Learning Java™ with JBuilder™

VERSION 5

Borland®
JBuilder™

볼랜드 코리아 주식회사
서울특별시 강남구 삼성동 159-1 ASEM 타워 30 층
연락처 :(02)6001-3162
www.borlandkorea.co.kr

볼랜드와 볼랜드 코리아는 이 문서에 포함된 모든 내용에 대한 특허를 가지고 있습니다.
이 문서를 공급함에 있어 사용자에게 특허권에 대한 어떠한 라이센스도 주어지지 않습니다 .

COPYRIGHT © 1997, 2001 Borland Software Corporation. All rights reserved. All Borland brand and product names are trademarks or registered trademarks of Borland Software Corporation in the United States and other countries. Other product names are trademarks or registered trademarks of their respective holders.

제품 구입 문의: Tel:02-6001-3194, 02-6001-3191
Fax:02-6001-3199
볼랜드 코리아
서울시 강남구 삼성동 159-1 ASEM 타워 30 층

Chapter

1

서문

*Learning Java with JBuilder*은 JBuilder 및 Java 프로그래밍 언어를 소개하는 자료입니다. 이 매뉴얼은 다음 세 가지 부분으로 구성됩니다.

- I부, "Quick Start"

  개발 환경에 대한 정보를 제공하고 프로젝트를 만들어 관리하며 사용자 인터페이스를 디자인하고 Java 프로그램을 컴파일 및 디버깅하는 방법에 대해 설명합니다. 또한 JBuilder와 관련 문서에 대한 일반적인 정보도 제공합니다.

- 2부, "Getting Started with Java"

  스레딩 방식 직렬화 및 원시 코드 인터페이스 사용 등의 자바의 프로그래밍 원리에 대해 살펴봅니다.

- 3부, "Tutorials"

  Jbuilder 통합 개발 환경(IDE)을 설치, 실행하고 생산적으로 사용하도록 디자인된 여러 가지 단계별 자습서를 제공합니다.

  - "자습서: 애플리케이션 구축"

    간단한 "Hello World" 애플리케이션을 생성합니다 .

  - "자습서:애플릿 만들기"

    AWT 애플릿 생성 절차를 설명합니다 .

Borland 개발자 지원 문의

Borland는 다양한 지원 옵션을 제공합니다.여기에는 인터넷의 무료 서비스가 포함되는데, 여기서 광범위한 정보 베이스를 찾고 Borland 제품을 쓰고 있는 다른 사용자를 접할 수 있습니다.그 외에도 Borland 제품의 설치 지원에서부터 유료 컨설팅 및 광범위한 지원에 이르기까지 여러 지원 방식 중에서 선택할 수 있습니다.

Borland의 개발자 지원 서비스에 대한 자세한 내용은 웹 사이트 http://www.borland.com/devsupport/를 참조하거나 Borland Assistant (800) 523-7070으로 전화하거나 Sales Department (831) 431-1064로 문의하십시오.

지원에 대하여 문의할 때는 사용자 환경에 대한 모든 정보, 사용 중인 제품 버전, 문제에 대한 상세한 설명 등을 준비해야 합니다.

협력 업체 툴 또는 설명서에 대해서는 툴 공급 업체에 문의하십시오.

온라인 리소스

다음과 같은 온라인 소스로부터 정보를 얻을 수 있습니다.

World Wide Web	http://www.borland.com/
FTP	ftp.borland.com 특정 ftp를 통해 사용 가능한 기술 문서
Listserv	전자 뉴스레터에 등록하려면 다음 주소의 온라인 양식을 사용하십시오. http://www.borland.com/contact/listserv.html 또는 Borland의 국제적인 listserver인 경우에 다음 주소의 온라인 양식을 사용하십시오. http://www.borland.com/contact/intlist.html
TECHFAX	1-800-822-4269(북미 지역) 팩스로 기술 문서를 받아 사용할 수 있습니다.

World Wide Web

www.borland.com를 정기적으로 확인합니다.JBuilder Product Team은 신규 및 기존 제품에 대한 백서, 경쟁사 분석, FAQ에 대한 답변, 샘플 애플리케이션, 업데이트된 소프트웨어, 업데이트된 설명서, 정보를 게시합니다.

구체적으로 다음 URL에서 확인할 수 있습니다.

- http://www.borland.com/jbuilder/(업데이트된 소프트웨어 및 기타 파일)
- http://www.borland.com/techpubs/jbuilder/(업데이트된 설명서 및 기타 파일)
- http://community.borland.com/(개발자용 웹 기반 뉴스 잡지 포함)

Borland 뉴스그룹

JBuilder를 등록하면 Jbuilder를 많이 사용하고 있는 여러 스레드 논의 그룹(threaded discussion group)에 참여할 수 있습니다.

http://www.borland.com/newsgroups/를 방문하면 JBuilder 및 다른 Borland 제품에 대한 사용자 지원 뉴스그룹을 찾을 수 있습니다.

Usenet 뉴스그룹

다음 Usenet 그룹은 Java 및 관련 프로그래밍 문제를 주로 다루는 그룹입니다.

- news:comp.lang.java.advocacy
- news:comp.lang.java.announce
- news:comp.lang.java.beans
- news:comp.lang.java.databases
- news:comp.lang.java.gui
- news:comp.lang.java.help
- news:comp.lang.java.machine
- news:comp.lang.java.programmer
- news:comp.lang.java.security
- news:comp.lang.java.softwaretools

참고 이러한 뉴스그룹은 사용자들에 의해 관리되며 공식적인 Borland 사이트는 아닙니다.

버그 보고

소프트웨어 버그라고 생각되는 점을 발견하면 JBuilder 개발자 지원 페이지 http://www.Borland.com/devsupport/jbuilder/에 알려주십시오.또한 이 사이트에서 기능 요청을 제출하거나 이미 보고된 버그 목록을 볼 수 있습니다.

버그를 보고할 때, 버그가 다시 발생할 수 있는 모든 단계, 이를테면 사용자가 사용했던 모든 특별한 환경 설정과 Jbuilder와 함께 사용한 다른 프로그램과 같은 내용을 포함해야 합니다.예상한 동작과 실제 발생한 것의 차이를 구체적으로 파악해야 합니다.

JBuilder 설명서에 대한 칭찬, 제안 사항 또는 문제 등과 관련된 의견이 있으면 jpgpubs@borland.com으로 메일을 보내 주십시오. 단 설명서에 관련된 문제만 해당됩니다.지원에 관한 문제는 개발자 지원팀에 알려야 합니다.

JBuilder는 개발자가 개발자를 위해 만든 것입니다.당사는 제품을 개선하는 데 도움을 주는 사용자의 제안을 가치 있게 생각합니다.

설명서 규칙

JBuilder용 Borland 설명서는 다음 표에 설명된 글꼴 및 기호를 사용하여 특수한 텍스트를 나타냅니다.

Table 1.1 글꼴 및 기호 규칙

글꼴	의미
Monospace type	다음은 고정 폭 형식을 나타냅니다. • 화면에 보이는 것과 같은 텍스트 • "애플리케이션 마법사의 Title 필드에 Hello World를 입력하는 것"과 같이 입력해야 하는 모든 것 • 파일 이름 • 경로 이름 • 디렉토리 및 폴더 이름 • SET PATH, CLASSPATH와 같은 명령 • Java 코드 • boolean, int, long과 같은 Java 데이터 타입. • 변수, 클래스, 인터페이스, 컴포넌트, 속성, 메소드, 이벤트 등과 같은 Java 식별자 • 패키지 이름 • 인수 이름 • 필드 이름 • void와 static과 같은 Java 키워드
Bold	Bold는 자바 툴, bmj(Borland Make for Java), bcj(Borland Compiler for Java) 및 컴파일러 옵션에 사용됩니다.예를 들면, 다음과 같습니다.**javac**, **bmj**, **-classpath**.
Italics	이탤릭체 단어는 정의되는 새로운 용어, 책 제목, 그리고 간혹 강조를 위해 사용됩니다.
Keycaps	이 글꼴은 키보드에 있는 특정 키를 나타냅니다.예를 들면, 다음과 같습니다. "메뉴를 종료하려면 *Esc*를 누릅니다."
[ ]	텍스트나 구문 목록에서 대괄호는 옵션 항목을 묶습니다.괄호를 입력하지 마십시오.
< >	텍스트 또는 구문 목록에서 각괄호는 변수 문자열을 나타내고 사용자의 코드에 적합한 문자열을 입력합니다.각괄호를 입력하지 마십시오.각괄호는 HTML 태그에도 사용됩니다.
...	코드 예제에서 생략 부호는 예제에서 생략된 코드를 나타냅니다. 버튼에서 생략 부호는 버튼이 선택 대화 상자에 연결되었음을 나타냅니다.

JBuilder는 다중 플랫폼에서 사용할 수 있습니다.설명서에서 사용된 플랫폼 및 디렉토리 규칙에 대한 설명은 아래 표를 참조하십시오.

Table 1.2 플랫폼 규칙 및 디렉토리

Item	의미
Paths	설명서에서 모든 경로는 슬래시(/)로 나타냅니다. Windows 플랫폼에서는 백슬래시(₩)를 사용합니다.
홈 디렉토리	홈 디렉토리의 위치는 플랫폼에 따라 다양합니다. • UNIX와 Linux의 경우 홈 디렉토리가 다를 수도 있습니다. 예를 들면, 홈 디렉토리가 /user/[username] 또는 /home/[username]일 수 있습니다. • Windows 95/98에서 홈 디렉토리는 C:₩Windows입니다. • Windows NT에서 홈 디렉토리는 C:₩Winnt₩Profiles₩[username]입니다. • Windows 2000에서 홈 디렉토리는 C:₩Documents and Settings₩[username]입니다.
.jbuilder5 디렉토리	JBuilder 설정이 저장되어 있는 .jbuilder5 디렉토리는 홈 디렉토리에 위치합니다.
jbproject 디렉토리	프로젝트, 클래스, 소스 파일을 포함하는 jbproject 디렉토리는 홈 디렉토리에 위치합니다.JBuilder는 파일을 이 기본 경로에 저장합니다.
스크린 샷	스크린 샷은 다양한 플랫폼에서 JBuilder의 Metal Look & Feel을 반영합니다.

Part

I

Quick Start

Chapter

2

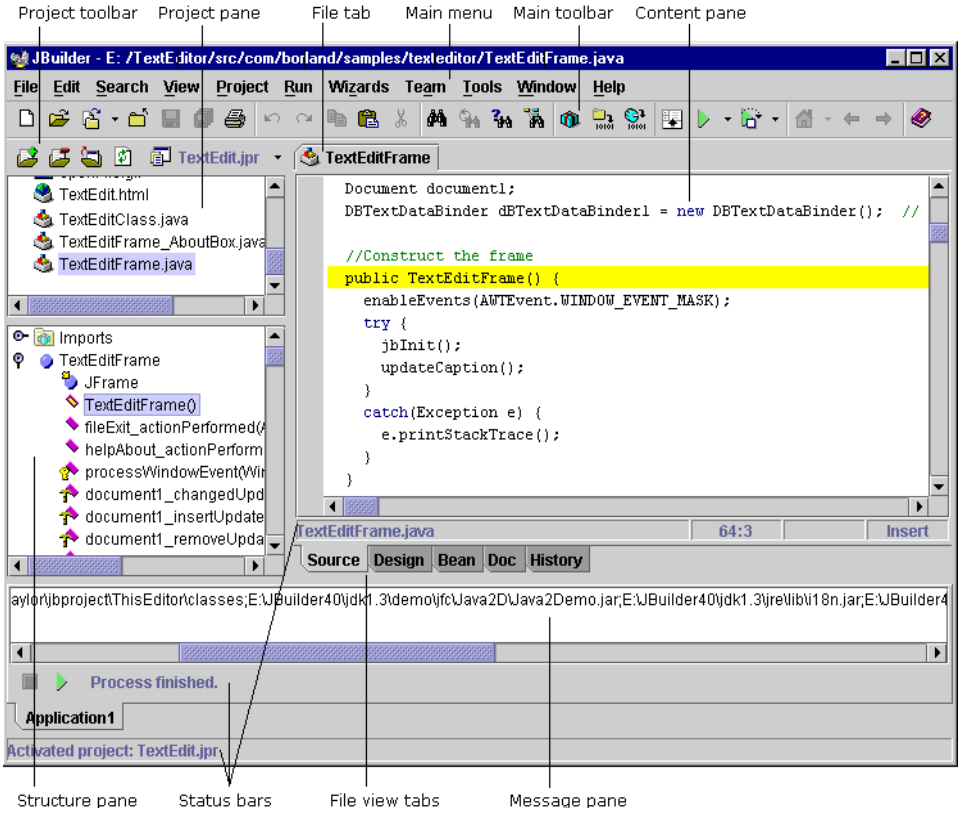
JBuilder 소개

JBuilder를 애용해 주셔서 감사합니다. 이 입문서는 JBuilder 통합 개발 환경(IDE)에 대한 개요를 제공합니다. *Quick Start*는 제품을 즉시 사용할 수 있도록 도와 주며 JBuilder의 Java 프로그래밍에 대한 자세한 정보가 들어 있는 위치를 알려 줍니다.

AppBrowser 소개

JBuilder 통합 개발 환경(IDE)은 많은 개발 기능을 처리하는 기술을 갖춘 단일 창을 제공합니다. 이러한 창을 AppBrowser라고 합니다. AppBrowser에서 버튼, 툴바, 메뉴 및 다른 사용자 인터페이스 요소를 사

용하여 파일 및 프로젝트를 생성, 편집 및 관리하며 애플리케이션을 시각적으로 디자인, 컴파일, 디버깅 및 실행할 수 있습니다.



AppBrowser는 다른 UI 요소에서 다양한 기능을 제공합니다.

Table 2.1 AppBrowser 요소의 기능

AppBrowser 요소	설명
메인 menu bar	File, Edit, Search, Run 및 Wizards와 같은 많은 메뉴에 대한 액세스를 제공합니다.
메인 툴바	명령의 단축키를 제공합니다. 버튼이 기능별로 그룹화됩니다. 툴바에서 마우스 오른쪽 버튼으로 클릭하고 컨텍스트 메뉴에서 선택하여 표시/숨김 사이를 토글합니다.
Project 창	프로젝트 드롭다운 목록에서 현재 선택한 프로젝트의 컨텐트를 표시합니다. 프로젝트 창에서 파일을 더블 클릭하거나 마우스 오른쪽 버튼으로 클릭하여 파일을 엽니다. 파일을 열지 않고도 프로젝트 트리를 탐색하고 처리할 수 있습니다.

Table 2.1 AppBrowser 요소의 기능 (continued)

AppBrowser 요소	설명
Project 툴바	현재 열려 있는 프로젝트의 드롭다운 목록을 제공하고 프로젝트 창에서 파일을 추가 및 제거하고 프로젝트를 닫거나 프로젝트 파일을 새로 고치는 버튼을 제공합니다.
Structure 창	아이콘, 정렬 옵션 및 오류 표시를 표시합니다. Javadoc @todo 태그를 지원합니다. 구조 창은 컨텐트 창에서 현재 선택한 파일의 구조를 보여 줍니다. Java 파일의 경우 이 구조는 파일에 정의된 모든 메소드, 속성 및 이벤트를 보여주는 트리 형식으로 표시됩니다. 구조 창은 드릴다운 기능을 제공합니다. 클래스나 인터페이스를 더블 클릭하여 그 조상을 봅니다. 파일 타입에 따라 그 구조가 다르게 표시될 수 있습니다.
컨텐트 창	열려 있는 파일의 컨텐트를 표시합니다. 열려 있는 각 파일에는 파일 이름을 표시하는 탭(파일 탭)이 있습니다. 이 창의 아래에는 사용 가능한 다른 보기(파일 보기 탭)에 대한 탭이 있습니다.
File view 탭	컨텐트 창의 보기를 소스, 디자인, 빈, doc 또는 사용 가능한 다른 보기로 변경할 수 있습니다.
File 탭	열려 있는 파일의 이름을 표시합니다. 활성 프로젝트의 파일 탭만 표시됩니다. 현재 프로젝트의 열려 있는 파일을 보려면 해당 파일 탭을 선택합니다. 파일 탭을 마우스 오른쪽 버튼으로 클릭하여 기본 파일 명령에 액세스합니다.
Message 창	디자이너, 검색 결과 및 컴파일러 같은 하위 시스템, 디버거 및 런타임 프로세스의 메시지를 표시합니다. 탭 모양의 메시지 창은 이러한 하위 시스템이 활성화될 때 표시됩니다. 또한 디버거 사용자 인터페이스도 저장합니다.
Status bar	모든 프로세스와 그 결과에 대한 정보를 계속적으로 알려줍니다. 상태 표시줄은 다음과 같이 세 가지가 있습니다. • 메인 상태 표시줄은 AppBrowser 창의 아래에 표시됩니다. • 파일 상태 표시줄은 컨텐트 창의 소스 보기로 열려 있는 파일의 아래에 표시됩니다. • 메시지 상태 표시줄은 메시지 탭 위의 메시지 창 아래에 표시됩니다.

AppBrowser는 OpenTools API를 사용하여 사용자 지정할 수 있습니다.

AppBrowser에 대한 자세한 내용은 Help|JBuilder Environment의 "JBuilder 환경" 항목에서 "Welcome Project" 및 "AppBrowser" 항목을 참조하십시오

AppBrowser 탐색

다음 키보드 단축키를 사용하여 AppBrowser 내에서 커서를 이동합니다.

Table 2.2 탐색 키보드 단축키

단축키	Action
Ctrl+Tab	다음 AppBrowser 창으로 이동합니다. 회전 순서는 프로젝트 창, 컨텐트 창, 메시지 창 탭, 메시지 창 텍스트 영역 및 구조 창입니다.
Shift+Ctrl+Tab	회전한 순서대로 이전 AppBrowser 창으로 이동합니다.
위쪽/아래쪽 화살표	트리에서 선택 커서를 위 아래로 이동합니다.
Enter 또는 왼쪽/오른쪽 화살표	프로젝트 및 구조 창의 상위 수준 노드를 확장하고 축소합니다.
Enter	더블 클릭처럼 작동합니다. 프로젝트 창에서 선택한 소스 파일을 열고 소스 보기에 커서를 놓습니다. 구조 창에서 선택한 클래스의 인터페이스나 수퍼클래스로 드릴다운합니다.

자세한 내용은 *Building Applications with JBuilder*의 "JBuilder 환경"장에서 "AppBrowser 탐색 및 검색" 항목을 참조하십시오.

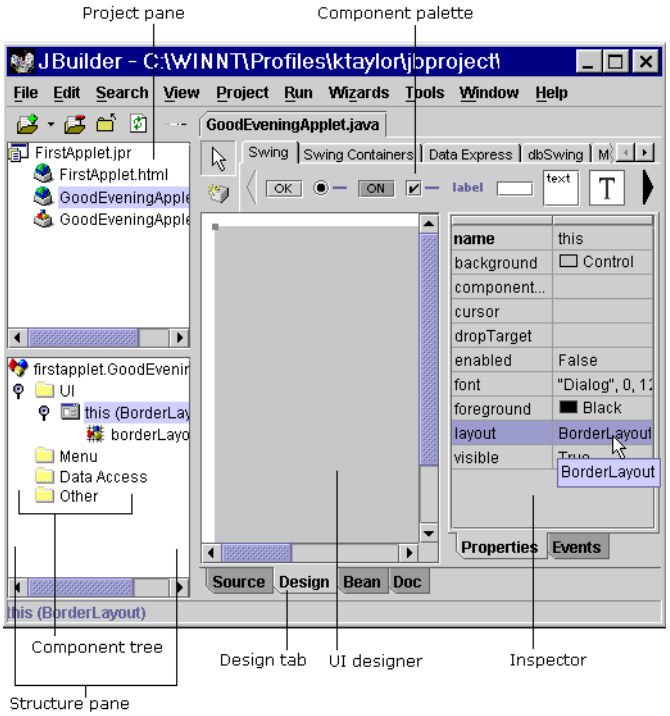
AppBrowser 디자인 뷰

UI 디자이너를 사용하여 시각적으로 애플리케이션을 디자인할 수 있습니다.

UI 디자이너에서 파일을 보려면 컨텐트 창의 아래에 있는 Design 탭을 선택합니다. 파일의 디자인 뷰가 표시됩니다. 컴포넌트 팔레트는 컨텐트 창의 상단에 나타나고 Inspector는 오른쪽에 나타납니다.

UI를 생성하려면 컴포넌트 팔레트에서 디자이너 또는 구조 창의 해당 노드로 컴포넌트를 드래그 앤 드롭합니다. 결과 코드가 자동으로 생성되어 파

일에 삽입됩니다. Inspector를 사용하여 선택한 컴포넌트의 속성을 조정합니다.



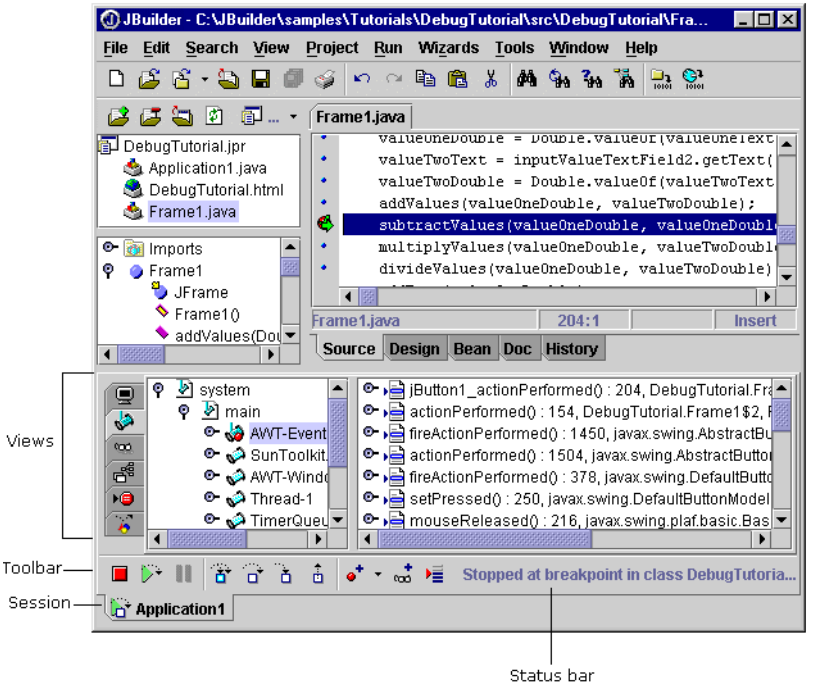
자세한 내용은 *Building Applications with JBuilder*의 "사용자 인터페이스 디자인"을 참조하십시오.

AppBrowser 아이콘 목록과 그 의미를 보려면 *Building Applications with JBuilder*의 *"JBuilder 환경"에서 "AppBrowser 아이콘" 단원을 참조하십시오.*

디버거 뷰의 AppBrowser 메시지 창

디버거를 실행하면 메시지 창에 나타납니다. 여러 디버깅 세션이 AppBrowser의 하단 부에 탭으로 표시됩니다.

디버거를 사용하려면 Run|Debug Project를 선택합니다.



자세한 내용은 *Building Applications with JBuilder*의 *"Java 프로그램 디버깅"을 참조하십시오.*

JBuilder의 언어 지원

JBuilder는 XML 및 Java 개발에 필요한 포괄적인 언어를 지원합니다.

JBuilder의 XML 기능은 다음 범주에 해당합니다.

- XML 문서의 프레젠테이션, 변환, 확인
- XML 문서의 데이터바인딩과 프로그램 처리
- 데이터베이스의 기업용 데이터 인터페이스

XML에 대한 자세한 내용은 Chapter 10, "JBuilder의 XML 지원" 및 *Building Applications with JBuilder*의 "JBuilder의 XML 기능 사용"을 참조하십시오. XML 기능은 JBuilder 에디션에 따라 다양합니다.

JBuilder의 자바 언어 지원은 다음과 같습니다.

- 재사용할 수 있는 컴포넌트에 필요한 JavaBeans
- Java 사용자 인터페이스 개발에 필요한 JFC/Swing 컴포넌트

JBuilder 전문가용에도 다음에 대한 언어 지원이 포함됩니다.

- JDBC
- 서블릿
- 여러 JDK(Java Development Kit)

  JBuilder 는 RMI 레지스트리를 시작하고 RMIC 를 사용하여 컴파일할 수 있습니다 .

  JBuilder 를 사용하면 작동 중인 JDK 1.1x 에서 다양한 버전의 JDK 에 대한 애플리케이션 및 애플릿을 구축할 수 있습니다. 기존의 모든 Java 2 100% 호환 프로그램은 JBuilder 환경에 추가될 수 있으며 JBuilder 환경에서 실행하고 작업할 수 있습니다 . Project|Project Properties 와 Paths 탭을 선택하여 JDK 버전을 변경하고 컴파일합니다 .

  자세한 내용은 http://www.javasoft.com 을 참조하십시오 .

JBuilder 기업용은 다음 내용에 대한 언어 지원을 추가합니다.

- 서버측 컴포넌트 아키텍처에 대한 EJB(Enterprise JavaBeans)
- CORBA
- 웹 기반 애플리케이션에 대한 JSP(JavaServer Pages)

JavaBeans에 대한 자세한 내용은 *Building Applications with JBuilder*의 "BeansExpress로 JavaBeans 생성"을 참조하십시오.

Enterprise JavaBeans에 대한 자세한 내용은 *Enterprise Application Developers Guide*를 참조하십시오.

JBuilder 학습

JBuilder 설명서

설명서는 다음과 같은 방법으로 사용할 수 있습니다.

문서	설명	인쇄	PDF	도움말	HTML
모든 에디션					
입문서	JBuilder 현재 버전의 새로운 기능에 대해 설명하고 개발 환경을 소개하며 JBuilder를 이용하여 처음으로 애플리케이션 및 애플릿을 만드는 단계별 자습서를 제공하고 각 항목에 대한 자세한 정보를 볼 수 있는 위치를 알려 줍니다.	X	X	X	X

문서	설명	인쇄	PDF	도움말	HTML
Java 시작하기	Java 프로그래밍 언어, 클래스 라이브러리 및 에디션에 대한 개요를 제공합니다.	X	X	X	X
JBuilder를 이용한 애플리케이션 구축	프로젝트를 만들고 관리하는 방법을 설명합니다. 버전 처리 및 명령줄 툴에 대한 정보가 들어 있습니다. 프로그램을 컴파일, 디버깅, 배포 및 국제화하는 방법 뿐만 아니라 사용자 인터페이스를 디자인하고 레이아웃 매니저를 사용하는 방법에 대해 설명합니다.		X	X	X
자습서(다양한 책으로 인쇄됨)	컴파일과 디버깅, 간단한 텍스트 에디터 구축 및 중첩 레이아웃 사용에 대한 자습서를 제공합니다.		X	X	X
JBuilder용 OpenTools 개발	고유한 추가 기능을 구축하는 OpenTools API 및 몇 가지 JBuilder 하위 시스템의 아키텍처에 대한 개념 정보를 제공합니다.		X	X	X
OpenTools API Reference	JBuilder OpenTools API에 대한 참조 설명서를 제공합니다.			X	X
JDK 1.3 설명서	Sun JDK(Java Development Kit)에 대한 API 참조 설명서입니다. 다음 두 가지 방법 중 하나로 이 설명서에 액세스할 수 있습니다. • Help|Java Reference를 선택합니다. • JDK 파일을 볼 때 컨텐트 창에서 Doc 탭을 선택합니다.			X	*
Java Language Specification	Java Language Specification의 2.0 버전이 포함되어 있습니다.			X	*
문맥에 따른 온라인 도움말	특히 현재의 JBuilder 사용자 인터페이스 요소와 관련된 정보를 제공합니다.				X
전문가용 및 기업용 에디션					
데이터베이스 애플리케이션 개발자 안내서	JBuilder의 DataExpress 데이터베이스 아키텍처에 대해 설명합니다. 주요 DataExpress 데이터 컴포넌트와 클래스 및 이 컴포넌트와 클래스를 사용하여 데이터베이스 애플리케이션을 만드는 방법 사이의 관계에 대해 설명합니다.	X	X	X	X
JdataStore 개발자 안내서	JDataStore 기능을 효율적으로 사용하는 방법에 대해 설명합니다. JDataStore는 고성능의 작은 풋프린트이며 전체적인 Java 데이터베이스입니다.	X	X	X	X
컴포넌트 라이브러리 참조	borland.com의 모든 부가 가치, data-aware 컴포넌트, 클래스, 속성, 메소드 및 이벤트에 대한 자세한 정보를 온라인으로 제공합니다.			X	X
웹 애플리케이션 개발자 안내서	애플릿, JSP(JavaServer Page) 및 서블릿을 사용하여 웹 기반 애플리케이션을 개발하고 디버깅하는 방법을 설명합니다.	X	X	X	X

문서	설명	인쇄	PDF	도움말	HTML
기업용 에디션					
기업용 JavaBeans 개발자 안내서	Enterprise JavaBeans 애플리케이션을 개발하고 디버깅하는 방법을 설명합니다.	X	X	X	X
분산 애플리케이션 개발자 안내서	CORBA와 RMI를 사용하여 분산 Java 및 웹 애플리케이션을 개발하고 디버깅하는 방법을 설명합니다.	X	X	X	X
JBuilder를 이용한 팀 개발	버전 관리에 대해 설명하고 JBuilder의 고유한 버전 처리 도구 뿐만 아니라 세 가지 주요 버전 제어 시스템과 함께 JBuilder의 인터페이스 사용 방법을 설명합니다.	X	X	X	X

* 이러한 설명서는 JavaSoft 웹 사이트 http://developer.java.sun.com/developer/infodocs/index.shtml에서도 구할 수 있습니다.

추가 리소스

다음과 같은 방법으로 Builder 및 Java에 대한 추가 정보를 볼 수 있습니다.

- JBuilder 웹 사이트 http://www.borland.com/jbuilder/ 및 Borland Community 웹 사이트 http://community.borland.com/을 방문하십시오.
- JBuilder 관련 도서를 보려면 http://www.borland.com/jbuilder/books/를 방문하십시오.
- Java 관련 도서 목록 및 용어집에 대해서는 "Java 학습"을 참조하십시오.

Java 학습

다양한 자바 언어에 대한 리소스입니다. 다음 목록은 일부에 불과하며 JBuilder 개발자의 라이브러리 및 북마크를 일정 수준까지 반영합니다.

온라인 용어집

다음은 Sun Microsystem의 온라인 Java 용어집입니다.

- Sun Microsystem의 HTML *Java Glossary*는 아래의 주소에 있습니다. http://java.sun.com/docs/glossary.nonjava.html#top
- Sun Microsystem의 Java용 *Java Glossary*는 아래의 주소에 있습니다. http://java.sun.com/docs/glossary.html

Sun에는 특정 Java 용어에 대한 더 많은 설명을 담고 있는 웹 페이지가 있으며 더 자세한 설명이 들어 있는 다른 웹 페이지와도 링크되어 있습니다.

http://developer.java.sun.com/developer/onlineTraining/new2java/programming/learn/unravelingjava.html

관련 도서

Java 프로그래밍에 대해 더 자세히 설명하는 도서 목록은 다음과 같습니다. 목록의 상반부는 쉬운 것부터 순서대로 정렬되어 있습니다. 하반부는 네트워크 프로그래밍 및 JavaBeans와 같은 특수 항목에 대해 설명합니다. 도서에 관한 자세한 내용을 보거나 Fatbrain.com에서 책을 구입하려면 http://www1.fatbrain.com/&from%3Dswp279로 이동하십시오.

책	저자	대상
Java for the World Wide Web: Visual Quickstart Guide (Peachpit Press)	Dori Smith	프로그래밍 백그라운드 없음
Java: First Contact (Course Technology)	Roger Garside와 John Mariani	프로그래밍 백그라운드 없음
A Little Java, A Few Patterns (MIT Press)	Mattias Felleisen과 Daniel P. Friedmens	초보자부터 고급 *
Beginning Java 2 (Wrox Press)	Ivor Horton	초보자
Learning Java (O□ eilly)	Patrick Niemeyer와 Jonathan Knudsen	초보자
Core Java 2, Volume 1: Fundamentals (Prentice Hall)	Cay S. Horstmann과 Gary Cornell	중간부터 고급
Just Java 2 (Prentice Hall)	Peter van der Linden	중간부터 고급
Thinking in Java (Prentice Hall)	Bruce Eckel	중간부터 고급
The Java Programming Language (Addison-Wesley)	Ken Arnold, James Gosling과 David Holmes	중급 상위부터 고급
The Complete Java 2 Certification Study Guide (Sybex, Inc.)	Simon Roberts, et al.	고급
Data Structures and Algorithms in Java (Waite Group Press)	Mitchell Waite와 Robert Lafore	고급

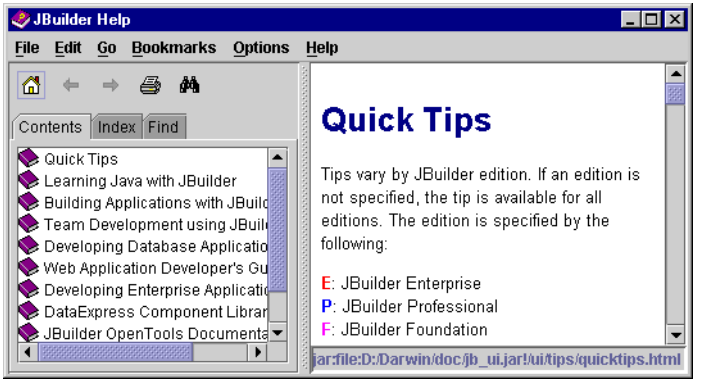
** 내용이 다소 철학적입니다. 개념 이해에는 도움이 되지만 "방법"을 이해하는 데에는 적합하지 않습니다.*

책	저자	주제
Java in a Nutshell (O□ eilly)	David Flanagan과 Daniel P. Friedmens	빠른 참조용
Java Class Libraries Reference (Addison-Wesley)	Patrick Chan, Rosanna Lee와 Doug Kramer	Java 클래스

책	저자	주제
Graphic Java 2: Mastering the JFC, Volume 2:Swing (Prentice Hall)	David M. Geary	Swing
Developing JavaBeans (O□ eilly)	Robert Englander	JavaBeans
Enterprise JavaBeans (O□ eilly)	Richard Monson-Haefel	네트워크 JavaBeans
Java 2 Networking (McGraw Hill)	Justin Couch	네트워크 프로그래밍
The Java Virtual Machine Specifications (Addison-Wesley)	Tim Lindholm과 Frank Yellin	네트워크 프로그래밍
Java Programming with CORBA (John Wiley and Sons, Inc.)	Andreas Vogel과 Keith Duddy	네트워크 프로그래밍
JDBC Database Access with Java: a Tutorial and Annotated Reference (Addison-Wesley)	Graham Hamilton, Maydene Fisher와 Rick Cattell	JDBC
Inside Servlets: Server-Side Programming for the Java Platform (Addison-Wesley)	Dustin R. Callaway	서블릿
Java: Servlet Programming (O□ eilly)	Jason Hunter와 William Crawford	서블릿

JBuilder의 온라인 도움말 사용

JBuilder는 Help Viewer에 온라인 도움말 항목을 표시합니다. AppBrowser나 웹 브라우저에 항목을 표시할 수도 있습니다.



도움말 보는 방법

다음과 같은 방법으로 JBuilder 내에서 도움말을 볼 수 있습니다.

- IDE 사용 시
  - Help viewer: JBuilder 메인 메뉴에서 Help|Help Topics를 선택하여 Help Viewer를 엽니다.
  - 컨텍스트 도움말: 대화 상자에 표시된 Help 버튼을 클릭하거나 Help 버튼이 있는 대화 상자, 창 또는 메뉴에서 *F1*을 누릅니다.
  - 클래스 참조: Search|Browse Symbol을 선택하고 클래스 이름을 입력한 다음 Doc 탭을 클릭합니다.
- 다음과 같은 두 가지 다른 방법으로도 AppBrowser에서 클래스 참조 정보를 볼 수 있습니다.
  - 구조 창에서 클래스 이름을 더블 클릭한 다음 Doc 탭을 선택합니다.
  - 소스 창에서 클래스 이름을 더블 클릭하고 Browse Symbol을 선택한 다음 Doc 탭을 선택합니다
- Inspector에서 속성이나 이벤트를 선택하고 *F1*을 누릅니다.

자세한 내용은 온라인 *Quick Start의 "JBuilder 학습" 장에서 "JBuilder의 온라인 도움말 사용"을 참조하십시오.*

Chapter

3

에디터 사용

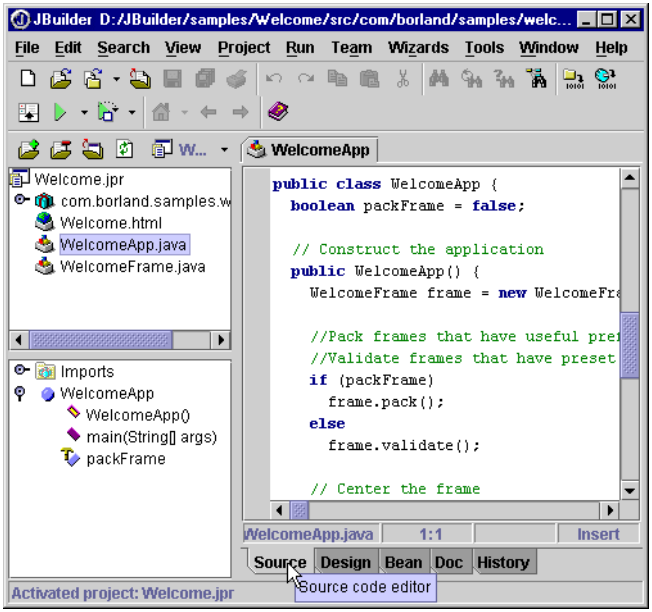
JBuilder에는 Java 코드를 작성하는 데 사용할 수 있는 모든 기능을 갖춘, 사용자 지정할 수 있는 에디터가 포함됩니다. Two-Way-Tools□사용하여 에디터에서 변경한 코드는 디자인 보기에 동시 반영됩니다. 전문가용 및 기업용 에디션에서 에디터는 @todo, @param 등과 같은 Javadoc 태그의 사용을 지원합니다.

그밖의 에디터의 생산성을 향상시키는 기능은 다음과 같습니다.

Table 3.1 에디터 기능

에디터 기능	설명
텍스트 검색	텍스트를 찾아 바꾸고 여러 파일을 검색하며 증분 검색합니다. 첫 번째 검색에 실패하면 파일의 맨 위에서부터 검색을 다시 시작할 수 있습니다.
구문 강조	.java, .c, .cpp, .html, .jsp, .xml, .xsl, .dtd, .sql 및 .idl 파일에서 지정한 구문 요소를 강조합니다.
코드 템플릿	사용자 정의 템플릿의 확장된 목록에서 나온 코드를 삽입합니다.
코드 스타일	중괄호 위치, 이벤트 핸들링 및 인스턴스 변수의 가시성을 설정합니다.
CodeInsight	코드를 완성하는 데 도움말을 제공하는 팝업 창을 에디터에 표시합니다. 디버깅할 때 값을 나타내는 툴팁 표현식을 표시합니다. .java 및 .jsp 파일에서 사용할 수 있습니다.
Browse Symbol	클래스 이름, 인터페이스 이름, 식별자, 메소드 이름, 속성 이름 및 패키지 이름을 찾습니다.

에디터에 액세스하려면 열려 있는, 텍스트 기반 파일의 컨텐트 창 아래에 있는 Source 탭을 선택합니다.



자세한 내용은 *JBuilder를 이용한 애플리케이션 구축*의 "JBuilder 환경"에서 "에디터"를 참조하십시오.

여러 가지 방법으로 편집 환경을 사용자 지정할 수 있습니다. 두 가지 메뉴, Tools|IDE Options와 Tools|Editor Options가 적용됩니다.

Tools|IDE Options의 Browser 탭에서는 컨텐트 창의 룩앤필(look and feel)과 키매핑 구성표(keymapping scheme) 및 파일 탭 방향을 변경할 수 있습니다. File Types 탭에서는 파일 타입과 관련 확장명을 추가할 수 있습니다. Run/Debug 탭에서는 런타임 업데이트 간격과 디버거 업데이트 간격을 설정할 수 있습니다.

Tools|Editor Options의 탭을 사용하면 다음을 설정할 수 있습니다.

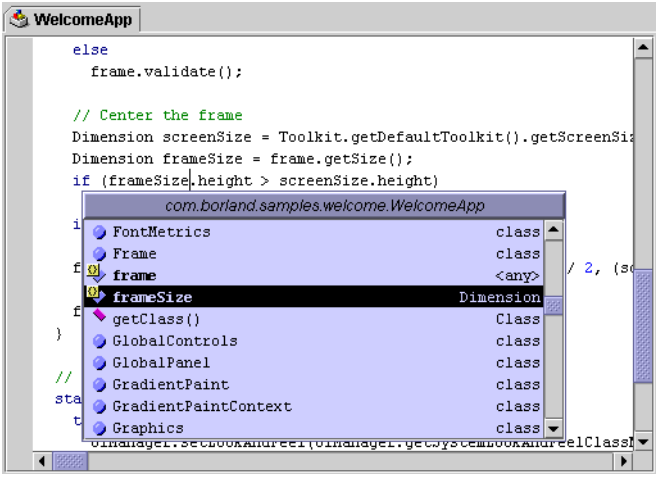
- Editor: 스마트 키, 탭 크기 및 블록 들여쓰기, 캐럿 표시, 유지하려는 백업 수, 검색 옵션 및 저장 옵션을 설정할 수 있습니다. 또한 이것으로부터 에디터 에뮬레이션을 변경하거나 사용자 지정할 수 있습니다.
- Display: 여백 및 글꼴을 변경합니다.
- Color: 수행 중인 작업이나 결과를 나타내기 위해 텍스트와 화면 요소의 색을 지정하고 코드를 강조하는 방법을 선택합니다.
- CodeInsight: 자동 팝업 옵션 및 지연 시간을 설정하고 사용할 CodeInsight의 부분을 설정합니다. Advanced Options를 사용하면 매개변수를 설정할 수 있습니다. 또한 CodeInsight를 호출하는 데 사용된 디스플레이 옵션 및 키스트로크를 설정할 수도 있습니다. 이 항목에 대한 자세한 내용은 "CodeInsight로 코드 완성" 참조하십시오.

- Templates: 코드 템플릿을 선택합니다. JBuilder 전문가용 및 기업용에서 코드 템플릿을 추가하고, 편집하며, 삭제할 수 있습니다.
- Java Structure: 구문 분석 지연 및 구조 순서를 조정합니다. 큰 파일의 에디터 속도를 향상시킬 수 있습니다.

자세한 내용을 보려면 Editor Options 또는 IDE Options 대화 상자 페이지(Tools|Editor Options, Tools|IDE Options)에서 Help 버튼을 클릭하십시오. 키매핑에 대한 자세한 내용을 보려면 Help|Keyboard Mappings를 선택하십시오.

CodeInsight로 코드 완성

JBuilder의 CodeInsight는 에디터 내에서 문맥에 맞는 팝업 창을 표시하여 코드를 완성하도록 도와줍니다.



CodeInsight는 다음을 표시합니다.

- 액세스 가능한 데이터 멤버 및 현재 컨텍스트의 메소드 목록(MemberInsight)
- 코딩할 메소드의 매개변수 목록(ParameterInsight)
- 현재 클래스 경로를 통해 액세스할 수 있는 클래스 목록(ClassInsight)
- 구조 창에 플래그로 표시되는 현재 코드의 오류(ErrorInsight)
- 전문가용 및 기업용: 디버깅할 때의 변수 값을 표시하는 툴팁 표현식 계산(ExpressionInsight).

JBuilder 전문가용 및 기업용에서 사용 가능

코딩할 때 원하는 정보 타입을 제공하도록 CodeInsight를 구성할 수 있습니다.

1. Tools|Editor Options를 선택하여 Editor Options 대화 상자를 엽니다.
2. CodeInsight 탭을 선택하고 해당 옵션을 변경합니다.
3. Display Options 버튼을 선택하여 팝업 창에 표시된 코드를 사용자 지정합니다.

CodeInsight 페이지에서 Keystrokes 버튼을 선택하여 CodeInsight 키를 구성합니다. 기본 키보드 단축키 목록은 Help|Keyboard Mappings에 있습니다.

자세한 내용은 *JBuilder를 이용한 애플리케이션 구축*의 "JBuilder 환경"에서 "CodeInsight"항목을 참조하십시오.

Browse Symbol 사용

Browse Symbol은 새 API를 배울 때 특히 유용합니다. 커서가 있는 메소드, 클래스, 인터페이스 또는 다른 기호에 대한 선언으로 이동합니다. 소스 코드를 사용할 수 있으면 소스로 이동합니다. 그렇지 않으면 디컴파일러 스텁 소스로 이동합니다. 커서가 기호의 이름에 있으면 기호를 호출하는 방법은 두 가지입니다.

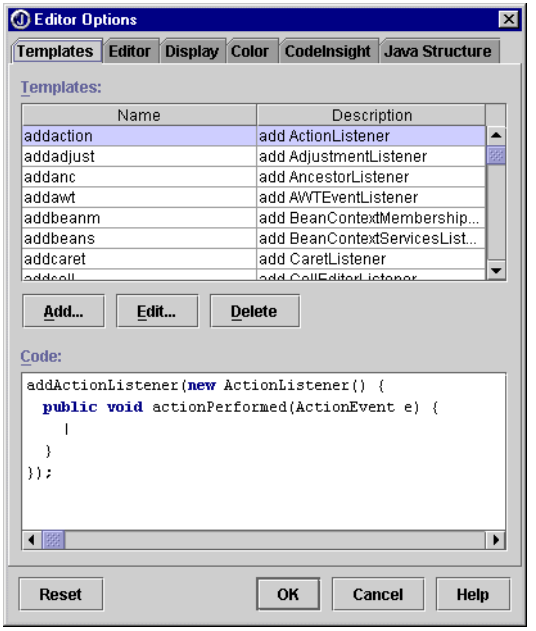
- 에디터의 컨텍스트(마우스 오른쪽 버튼) 메뉴에서 호출합니다.
- *Ctrl + Enter*를 눌러 호출합니다.

참고 클래스는 이런 방법으로 찾은 import 경로에 있어야 합니다.

코드 템플릿 사용

JBuilder에는 클래스 선언, if, if else, try/catch 및 while 문 같은 기본 코드 템플릿이 포함되어 있습니다.에디터에서 코드 템플릿을 사용하면 코딩 속도를 높일 수 있습니다. 코드 템플릿 이름을 입력하면 에디터에서 자동으로 템플릿을 표시합니다. *Ctrl + j* 를 눌러 표시된 템플릿을 확장하거나 템플릿 메뉴에 액세스합니다.

JBuilder 전문가용 및 기업용에서 코드 템플릿은 완벽하게 사용자 지정할 수 있습니다. Editor Options 대화 상자(Tools|Editor Options)의 Templates 페이지에서 코드 템플릿을 편집, 추가 및 삭제합니다.



자세한 내용은 *JBuilder를 이용한 애플리케이션 구축*의 "JBuilder 환경"에 있는 "코드 템플릿 사용"을 참조하십시오.

에디터 에뮬레이션의 키매핑

JBuilder 환경을 사용자 지정하여 즐겨쓰는 에디터를 에뮬레이트할 수 있습니다. JBuilder는 다음과 같은 에디터 에뮬레이션 키매핑을 제공합니다.

- 기본값/CUA
- Emacs
- Brief
- Visual Studio®

Visual Studio® 에디터 에뮬레이션은 Microsoft의 Visual Studio® 에디터와 호환되는 키스트로크입니다.

Keymap Editor를 사용하여 개별 키바인딩을 보고 변경할 수 있습니다. Tools|IDE Options 또는 Tools|Editor Options를 선택하고 첫 페이지에서 Customize를 클릭합니다. Keymap Editor는 JBuilder의 모든 에디션에서 볼 수 있습니다.

이것은 JBuilder 전문가용 및 기업용 버전의 기능입니다.

모든 에디터 에뮬레이션은 쉽게 사용자 지정할 수 있습니다. JBuilder의 에디터 에뮬레이션을 변경하거나 사용자 지정하려면

- Tools|IDE Options를 선택합니다. Browser 탭을 선택합니다 .Customize를 클릭합니다.

  또는

- Tools|Editor Options를 선택합니다. Editor 탭을 선택합니다 .Customize를 클릭합니다.

Keymap Editor를 사용하는 방법에 대해 자세히 알아보려면 Keymap Editor 대화 상자에서 Help 버튼을 클릭하십시오.

이러한 키매핑 대부분은 예제로 사용하여 새로운 키매핑을 만드는 일반 모델로 사용할 수도 있습니다.

Chapter

4

애플리케이션 개발 자동화

JBuilder는 애플리케이션 개발 시간을 절약하는 마법사를 제공합니다. 마법사를 사용하여 파일, 설정 및 환경 설정을 빠르게 만들거나 수정할 수 있습니다. 마법사가 파일이나 애플리케이션의 프레임워크를 만들어 주므로 사용자는 개발에만 집중할 수 있습니다.

마법사 사용

JBuilder는 많은 마법사를 제공하여 프로젝트, 파일, 설정, 빈, 그룹, 도구, 환경 설정 등을 빠르게 만들거나 수정합니다. 마법사를 사용하면 사용자는 많은 시간을 절약할 수 있고 논리적인 관리 대신 프로그램 개발에 집중할 수 있습니다.

각 마법사는 JBuilder IDE의 논리적 위치에서 액세스할 수 있습니다. 일부 마법사는 마법사가 영향을 주는 특정 영역에서 사용할 수 있고 다른 마법사는 보다 일반적인 영역에서 사용할 수 있다는 의미입니다.

JBuilder에서 사용할 수 있는 마법사는 에디션에 따라 다양합니다.

보유하고 있는 JBuilder 에디션에서 사용할 수 있는 마법사에 대한 자세한 내용은 메뉴를 마우스 오른쪽 버튼으로 클릭하면 나타나는 메뉴 컨텍스트 도움말에서 확인하십시오.

마법사를 찾을 수 있는 일반적인 영역은 다음과 같습니다.

- File|New. 객체 갤러리가 표시됩니다. 사용하려는 마법사의 종류에 해당하는 탭을 선택합니다. 기본 탭은 New 탭이며, Project, Interface, JavaBean 및 Panel 마법사 등과 같이 광범위하게 사용되는 마법사가 포함되어 있습니다. JBuilder 에디션에 따라 그 외의 다른 탭에는 Team, Web, XML, CORBA 및 기업용 마법사가 포함되어 있습니다.
- Wizards 메뉴. 사용 가능한 마법사는 에디션에 따라 다양합니다. 마법사를 사용할 수 없다면 그 이유는 에디션에 포함되어 있지 않거나 필요한 구조가 제 위치에 없기 때문일 것입니다.

주제별 마법사를 찾기에 좋은 위치는 다음과 같습니다.

- Team 메뉴. 객체 갤러리의 Team 탭에서 프로젝트를 불러오는 동안 메인 툴바의 Team 메뉴에서 지원되는 다른 버전 제어 작업을 수행할 수 있습니다.
- Tools 메뉴의 대화 상자에 있는 버튼. 예를 들어, Tools|Configure Library에서 New 버튼을 선택하여 New Library 생성 마법사에 액세스 합니다.

마법사에 대한 자세한 내용은 *JBuilder를 이용한 애플리케이션 구축*의 "JBuilder 환경" 장에서 "마법사로 시작하기"를 참조하십시오.

프로젝트 작업

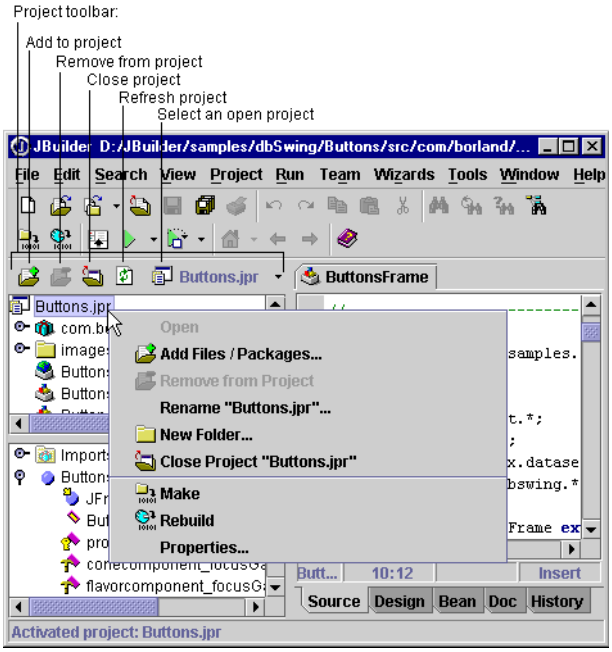
JBuilder 환경에서 프로그램을 개발하려면 먼저 프로젝트를 만들어야 합니다. 프로젝트는 파일 및 패키지를 구성하고 JBuilder 프로그램을 만드는 설정을 유지 관리합니다. 프로젝트는 프로젝트 창에서 보고 관리할 수 있습니다.

JBuilder 프로젝트는 조직 도구입니다. JBuilder 프로젝트에 파일을 넣어도 디렉토리 구조에서 해당 파일의 위치는 변경되지 않습니다. 파일을 이동하지 않고 많은 JBuilder 프로젝트에서 같은 파일을 사용할 수 있다는 의미입니다.

JBuilder 프로젝트에 대한 정보는 프로젝트 파일에 저장됩니다. 프로젝트 파일에는 프로젝트에 있는 파일 및 패키지 목록과 프로젝트를 정의하는 프로젝트 속성이 포함됩니다. JBuilder는 프로젝트를 로드하거나 저장, 구축 또는 배포할 때 이러한 정보를 사용합니다. 프로젝트 파일은 직접 편집할 수 없으며 JBuilder 개발 환경을 사용하여 파일을 추가 또는 제거하거나 경로나 연결 설정 같은 프로젝트 속성을 설정할 때마다 수정됩니다.

프로젝트 파일은 프로젝트 창의 노드로 표시됩니다.프로젝트의 모든 파일이 아래에 나열되어 있습니다. 프로젝트 파일마다 컨텍스트(마우스 오른쪽 버튼) 메뉴가 있습니다.

Figure 4.1 프로젝트 파일의 컨텍스트 메뉴가 표시된 프로젝트 창



프로젝트 저장

프로젝트에서 작업하는 동안 프로젝트를 기본 위치에 저장하거나 사용자가 선택한 디렉토리에 저장할 수 있습니다. 기본적으로 JBuilder는 홈 디렉토리의 jbproject 디렉토리에 프로젝트를 저장합니다.

각 프로젝트는 별도의 고유한 디렉토리에 저장됩니다. 프로젝트 디렉토리마다 프로젝트 파일(.jpr 또는 .jpx 타입), 프로젝트 노트의 옵션 .html 파일, 클래스 파일의 classes 디렉토리, 소스 파일의 src 디렉토리 및 파일 백업 복사본의 bak디렉토리가 포함됩니다.

기본적으로 자식 디렉토리가 있는 프로젝트 디렉토리는 jbproject에 저장됩니다. 모든 경로는 Project|Project Properties 대화 상자의 Project 마법사와 프로젝트 창의 프로젝트 파일 노드에서 사용할 수 있는 컨텍스트 메뉴에서 변경할 수 있습니다.

jbproject 및 홈 디렉토리의 위치에 대한 자세한 내용은 1- 4페이지의 "설명서 규칙"의 플랫폼 규칙 및 디렉토리 표를 참조하십시오.

Project 마법사 사용

JBuilder에는 프로젝트를 쉽게 만들 수 있는 Project 마법사가 있습니다. Project 마법사를 사용하여 새 프로젝트를 만들면 마법사는 자동으로 프로젝트에 대한 디렉토리 프레임워크를 설정하고 적절한 경로 및 JDK 버전 같은 프로젝트 속성에 대한 정보를 개발하고 저장합니다. 프로젝트 노트 파일을 만들 수도 있습니다. 이 파일의 정보는 프로젝트에 포함된 소스 파일의 Javadoc로 이동할 수 있습니다.

프로젝트를 열지 않고 Application 마법사나 Applet 마법사를 사용할 경우 먼저 Project 마법사가 시작되므로 새 프로젝트를 만들어 새 애플리케이션이나 애플릿을 저장할 수 있습니다.

Project 마법사를 사용하면 조정하려는 설정의 양에 따라 세 단계 중 어느 곳에서나 완료할 수 있습니다.

1. 첫 번째 단계에서 프로젝트 이름을 지정하고, 프로젝트 타입을 결정하며 루트 경로를 설정하고 프로젝트, 소스, 백업, 설명서 및 출력 디렉토리 이름을 지정합니다. 기본 경로에 친숙하여 경로를 변경할 필요가 없으면 이 단계에서 Finish를 클릭할 수 있습니다.
2. 두 번째 단계에서는 이러한 디렉토리의 경로를 설정하고 사용할 수 있는 JDK를 선택하며 필요한 라이브러리를 어떠한 것이라도 추가합니다. JBuilder에서 프로젝트 노트 HTML 파일을 생성할지 여부를 결정할 수도 있습니다. 이 옵션을 선택하지 않고 프로젝트 노트를 공백으로 남겨 놓으려면 Finish를 클릭합니다.
3. 프로젝트 노트 파일을 선택한 경우에는 세 번째 단계가 초기 정보를 입력하는 곳이 됩니다. 즉, 프로젝트 제목 및 작성자, 프로젝트를 사용할 회사, 프로젝트에 대한 설명 등을 입력합니다.마법사를 완료하려면 Finish를 클릭합니다.

프로젝트 창의 상단에 새로 만들어진 프로젝트 노드가 나타나고 그 아래에 프로젝트 노트 파일이 나타납니다. 경로, JDK 및 라이브러리를 보려면 프로젝트 노드를 마우스 오른쪽 버튼으로 클릭하고 Properties를 선택한 다음 Paths 탭을 선택합니다. Javadoc 클래스 필드로 프로젝트 노트 정보를 보려면 같은 대화 상자에서 General 탭을 선택합니다.

Project 마법사에 대한 자세한 내용은 *JBuilder를 이용한 애플리케이션 구축*의 "프로젝트 생성 및 관리" 장에서 "Project 마법사를 사용한 새 프로젝트 생성"을 참조하십시오.

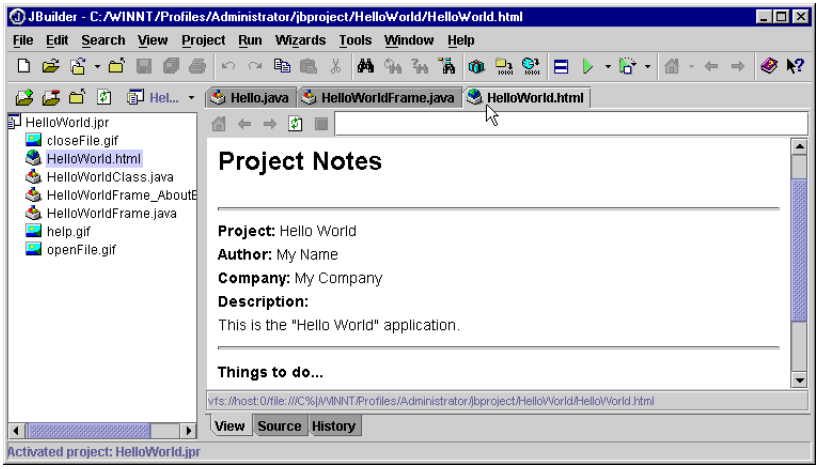
프로젝트 파일 사용

JBuilder는 AppBrowser의 프로젝트 창에 프로젝트 파일을 표시합니다. 프로젝트를 구성하는 파일은 그 아래에 나열됩니다. 컨텐트 창에서 파일을 열려면 프로젝트 창에서 파일 이름을 더블 클릭합니다.

참고 동시에 여러 파일을 보려면 AppBrowser의 여러 인스턴스를 엽니다. 원하는 각 인스턴스에 대해 Window|New Browser를 선택합니다.

컨텐트 창의 상단에 파일 이름을 가진 탭이 나타납니다. 여러 파일이 열려 있으면 각 파일에 대한 파일 탭이 만들어집니다. 파일 탭을 선택하거나 Window 메뉴에서 파일을 선택하면 열려 있는 다른 파일을 볼 수 있습니다. 파일 탭의 레이블과 위치를 사용자 지정할 수 있습니다. 파일 탭 사용자 지정에 대해 자세히 알아보려면 *JBuilder를 이용한 애플리케이션 구축*의 "JBuilder 환경"에서 "File Taps" 항목을 참조하십시오.

다음 그림은 프로젝트 파일 아래 소소 및 이미지 파일이 나열된 프로젝트 창의 프로젝트 파일, HelloWorld.jpr을 보여 줍니다. 컨텐트 창에는 프로젝트 노트 파일, HelloWorld.html이 선택되어 있습니다. 프로젝트 노트는 Project 마법사의 3 단계에서 입력한 정보로 만들어집니다.



프로젝트 속성 설정

프로젝트 속성을 설정하려면 Project|Project Properties를 선택하거나 프로젝트 창에서 프로젝트 파일을 마우스 오른쪽 버튼으로 클릭한 다음 Properties를 선택합니다. 모든 프로젝트의 기본 속성을 설정하려면 Project|Default Project Properties를 선택합니다.

참고 사용 가능한 프로젝트 속성 탭은 JBuilder 에디션에 따라 다양합니다.

Project Properties 대화 상자에는 다음과 같은 탭이 있습니다.

- Paths: 출력 경로, 소스 경로, 백업 경로, JBuilder 전문가용 및 기업용의 JDK 버전, 그리고 라이브러리 경로(다음 제목 참조)를 지정합니다.
- General: 인코딩, 자동 소스 패키징, 클래스 Javadoc 필드 등을 설정합니다.
- Run: 애플리케이션, 애플릿, JSP/Servlet이나 EJB을 선택하고 매개변수를 설정합니다.
- Debug: Smart Step 및 원격 디버깅 설정을 만듭니다.
- Build: 컴파일러 옵션, IDL, JSP 및 리소스 파일을 선택합니다.
- Code style: 중괄호 위치, 이벤트 핸들링 및 인스턴스화 옵션을 설정합니다.
- Editor: 줄 끝 문자를 선택합니다.
- WebServers/Servers: 전문가용의 웹 서버 및 기업용의 애플리케이션 서버를 선택하고 설정하며 기본 루트를 설정합니다.

프로젝트 속성에 대한 자세한 내용은 *JBuilder를 이용한 애플리케이션 구축*의 "프로젝트 생성 및 관리"에서 "프로젝트 속성 설정" 항목을 참조하십시오.

라이브러리

일반적으로 명령줄에서 Java 프로그램을 실행하면 명령줄에 classpath를 입력하여 프로그램이 필요로 하는 사항을 찾을 수 있습니다 . 라이브러리는 JBuilder 에서 classpath 정보를 저장하고 그룹화하는 방법입니다 . 라이브러리는 클래스 , 소스 파일 및 설명서 파일의 경로 모음입니다 . 라이브러리는 아카이브 파일 , classpath 및 다른 라이브러리를 포함할 수 있습니다 .

JBuilder는 라이브러리를 사용하여 프로젝트 실행, 코드 컴파일, 소스 검색, Javadoc 보기, 디자이너 사용 또는 CodeInsight 기능을 지원하는 클래스를 찾는 데 필요한 classpath를 찾습니다.

라이브러리에 대한 자세한 내용은 *JBuilder를 이용한 애플리케이션 구축*의 "프로젝트 생성 및 관리"에서 "라이브러리 사용"을 참조하십시오.

프로젝트 관리

AppBrowser에서 다음을 수행할 수 있습니다.

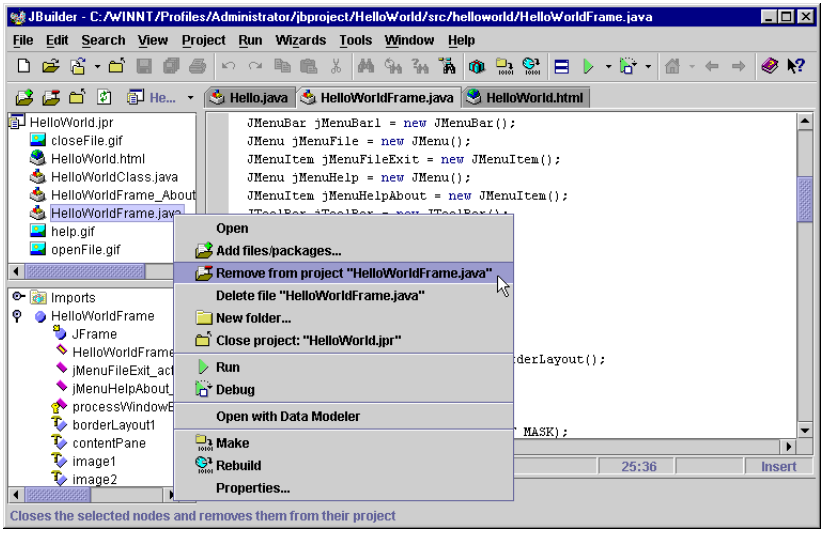
- 활성 프로젝트의 파일을 볼 수 있습니다.
- 경로와 이름을 포함하는 여러 파일 및 프로젝트를 열어 편집할 수 있습니다.
- 프로젝트에 소스 파일 및 패키지를 추가할 수 있습니다.
- 프로젝트 폴더를 추가할 수 있습니다.
- 패키지를 탐색할 수 있습니다.
- HTML 파일 및 웹 그래픽을 찾아볼 수 있습니다.
- 클래스, 메소드 및 코드 요소의 구조로 드릴다운할 수 있습니다.

프로젝트 열기

프로젝트를 열려면 File|Open Project 를 선택하고 원하는 프로젝트 파일을 찾아 봅니다 . 이전에 열어 본 프로젝트를 열려면 File|Reopen 을 선택하고 드롭다운 목록에서 프로젝트 파일을 선택합니다 . 또는 메인 툴바에서 Open 이나 Reopen 버튼을 클릭합니다 .

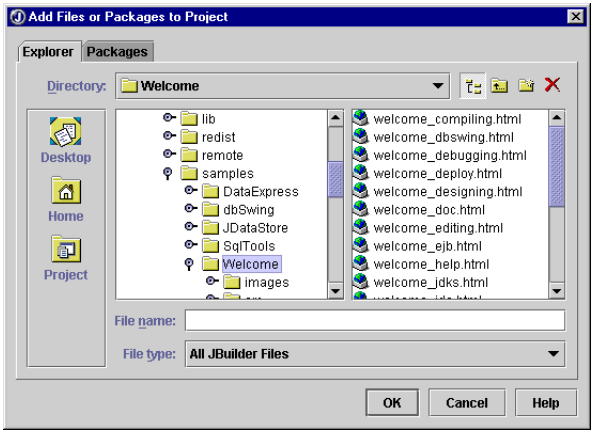
파일 추가 및 제거

프로젝트 툴바의 Add Files/Packages 또는 Remove From Project 버튼을 사용하거나 프로젝트 창에서 파일을 마우스 오른쪽 버튼으로 클릭하고 마우스 오른쪽 버튼 메뉴에서 Add Files/Packages 또는 Remove From Project 를 선택하여 프로젝트에서 파일 및 패키지를 추가하고 제거합니다 . 프로젝트 창에서 파일을 선택하고 Delete 를 선택하여 하드 드라



이브에서 파일을 삭제합니다 .

Add Files/Packages 대화 상자에는 두 가지 탭이 있습니다. Explorer 탭을 사용하여 디렉토리 및 파일을 찾아보고 Packages 탭은 사용 가능한 패키지 목록을 보여 줍니다.



팁 새 파일 이름을 입력하고 OK를 클릭하여 Explorer 페이지에서 새 파일을 만듭니다. 해당 파일을 만들지 묻는 메시지가 나타나면 OK를 클릭합니다.

프로젝트 저장 및 닫기

프로젝트를 저장하려면 File|Save All, File|Save Current Project 를 선택하거나 메인 툴바에서 Save All 버튼을 클릭합니다 .

프로젝트를 닫으려면 File|Close Project, File|Close Files를 선택하거나 프로젝트 툴바에서 Close Project 버튼을 클릭합니다.

프로젝트 및 파일의 이름 재지정

프로젝트 이름을 재지정하려면 프로젝트 창에서 프로젝트 파일을 선택합니다 . 그런 다음

1. Project|Rename을 선택하거나 마우스 오른쪽 버튼으로 클릭하고 Rename을 선택합니다.
2. Rename 대화 상자의 File Name 필드에 새로운 이름을 입력합니다.
3. OK를 클릭합니다.

다음과 같은 방법으로 열린 파일의 이름을 재지정합니다.

1. File|Rename을 선택하거나 컨텐트 창의 상단에 있는 파일 탭을 마우스 오른쪽 버튼으로 클릭하고 Rename을 선택합니다.
2. Rename 대화 상자의 File Name 필드에 새로운 이름을 입력합니다.
3. Save를 클릭합니다.

주의 프로젝트 및 파일 이름을 재지정해도 코드 내의 관련 패키지 및 파일 이름에 대한 참조는 변경되지 않습니다. 이 작업은 수동으로 수행해야 합니다.

레거시 파일 사용

JBuilder 파일을 여는 것과 같은 방식으로 JBuilder 에서 VisualAge 및 Forte 파일을 엽니다 . VisualAge 파일은 완전히 호환 가능합니다 . 디자이너에서 Forte 파일을 사용하려면 UI 핸들링 코드가 특정 조건을 만족하여 디자이너에서 코드를 제대로 해석할 수 있어야 합니다 . 자세한 내용은 *JBuilder 를 이용한 애플리케이션 구축*의 " 다른 Java IDE 에서 파일 이전 " 을 참조하십시오 .

Package Migration 툴을 사용하여 이전 버전의 Jbuilder에서 만들어진 파일을 JBuilder 5로 가져올 수 있습니다. Package Migration 툴은 클래스 이름, 패키지 이름, 경로 및 모든 관련 내부 참조를 현재 JBuilder 설치에 설정한 매개변수로 변환합니다. 이 툴을 사용하려면 Tools|Package Migration을 선택합니다.

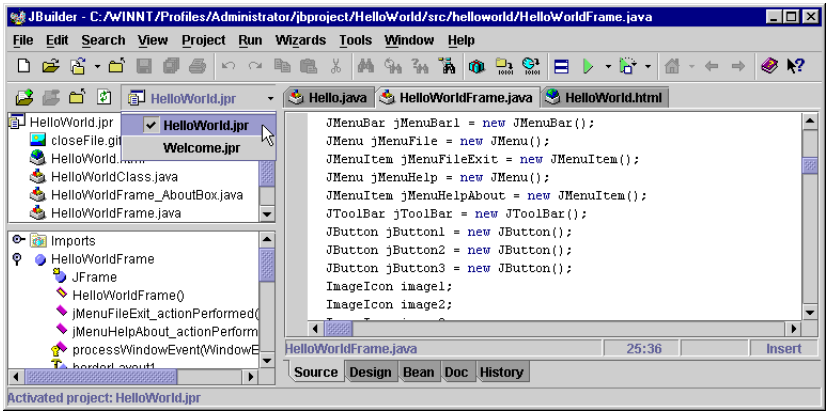
자세한 내용을 보려면 Package Migration 툴의 Help 버튼을 클릭하십시오.

여러 프로젝트 작업

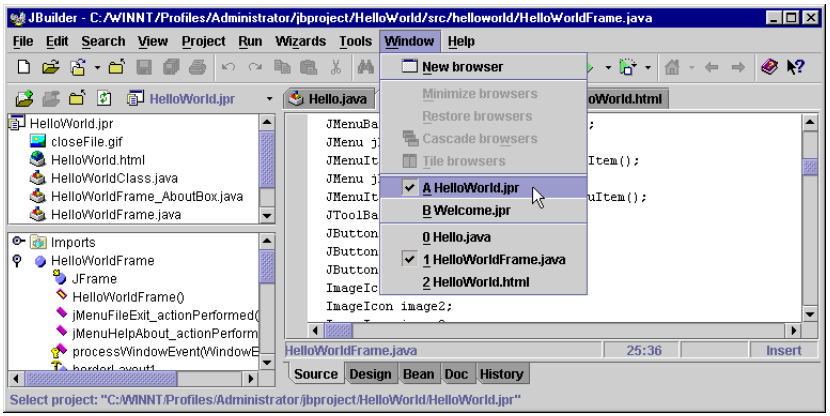
프로젝트를 AppBrowser 의 한 인스턴스나 다른 인스턴스에서 열 수 있습니다 . 열려 있는 모든 프로젝트는 열려 있는 어떤 AppBrowser 인스턴스에서든지 사용할 수 있습니다. 그러나 각 AppBrowser 인스턴스에서는 한 번에 한 프로젝트의 파일만 볼 수 있습니다 .

열려 있는 여러 프로젝트와 파일 사이를 전환하는 방법은 다음과 같이 여러 가지가 있습니다.

- 프로젝트 툴바의 드롭다운 목록에서 프로젝트를 선택합니다.



- Window 메뉴의 열려 있는 파일 목록에서 파일을 선택합니다.



- Window 메뉴에서 다른 AppBrowser를 인스턴스화하거나 열려 있는 AppBrowser 사이에서 전환할 수도 있습니다.

자세한 내용은 *JBuilder를 이용한 애플리케이션 구축*에서 "프로젝트 생성 및 관리"를 참조하십시오.

JavaBeans 생성

JavaBean은 독립적이며 재사용할 수 있는 컴포넌트 역할을 수행하는 하나 이상의 Java 클래스 모음입니다. JavaBean은 데이터 모듈이나 계산 엔진 같은 non-UI 컴포넌트나 사용자 인터페이스를 구축하는 데 사용되는 개별 컴포넌트일 수 있습니다.가장 간단한 JavaBean은 매개변수가 없는 생성자를 가진 **public** Java 클래스입니다. 일반적으로 JavaBeans에는 특정 이름 지정 규칙을 따르는 속성, 메소드 및 이벤트가 있습니다.

JavaBeans는 다른 컴포넌트에 비해 다음과 같은 고유한 장점이 있습니다.

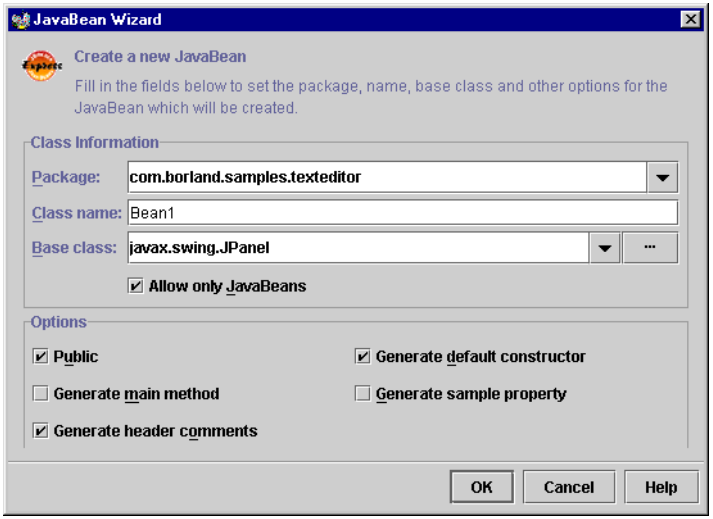
- 이것은 순수 Java, 크로스 플랫폼 컴포넌트입니다.
- JavaBeans를 JBuilder 컴포넌트 팔레트에서 설치하고 프로그램의 구성 및 디자인에 사용하거나 Java용 다른 애플리케이션 빌더 툴에서 사용할 수 있습니다.
- 이것은 JAR 파일로 배포할 수 있습니다.

전문가용 및 기업용에서 사용할 수 있는 JBuilder의 BeansExpress를 사용하면 JavaBeans를 가장 빨리 만들 수 있습니다.이것은 JavaBeans를 신속하고 쉽게 만드는 데 도움을 주는 마법사, 비주얼 디자이너, 코드 샘플의 집합으로 구성됩니다.일단 JavaBean을 가지면 BeansExpress를 사용하여 변경할 수 있습니다.또는 기존 Java 클래스를 취하여 JavaBean으로 바꿀 수 있습니다.

이것은 JBuilder 전문 가용 및 기업용 버전의 기능입니다.

다음과 같은 방법으로 JBuilder의 JavaBean 마법사에 액세스하여 JavaBean을 만듭니다.

1. File|New Project를 선택하고 Project 마법사를 사용하여 새 프로젝트를 만듭니다.
2. File|new를 선택하여 객체 갤러리를 표시합니다.
3. 객체 갤러리의 New 페이지에서 JavaBean 아이콘을 더블 클릭하여 JavaBean 마법사를 엽니다.



자세한 내용은 *JBuilder를 이용한 애플리케이션 구축*의 "BeansExpress를 이용한 JavaBeans 생성"을 참조하십시오.

Chapter

5

사용자 인터페이스 구축

JBuilder의 비주얼 디자인 툴을 사용하면 Java 프로그램을 위한 UI(User Interface)를 빠르고 쉽게 생성할 수 있습니다. 버튼, 텍스트 영역, 목록, 대화 상자, 메뉴 같은 컴포넌트가 들어 있는 팔레트를 사용하여 UI를 구축할 수 있습니다. 그런 다음 컴포넌트 속성의 값을 설정하고 컴포넌트 이벤트에 이벤트 핸들러 코드를 추가하여 UI 이벤트에 어떻게 반응할지를 프로그램에 알려 줍니다.

Figure 5.1 AppBrowser 및 UI 디자이너

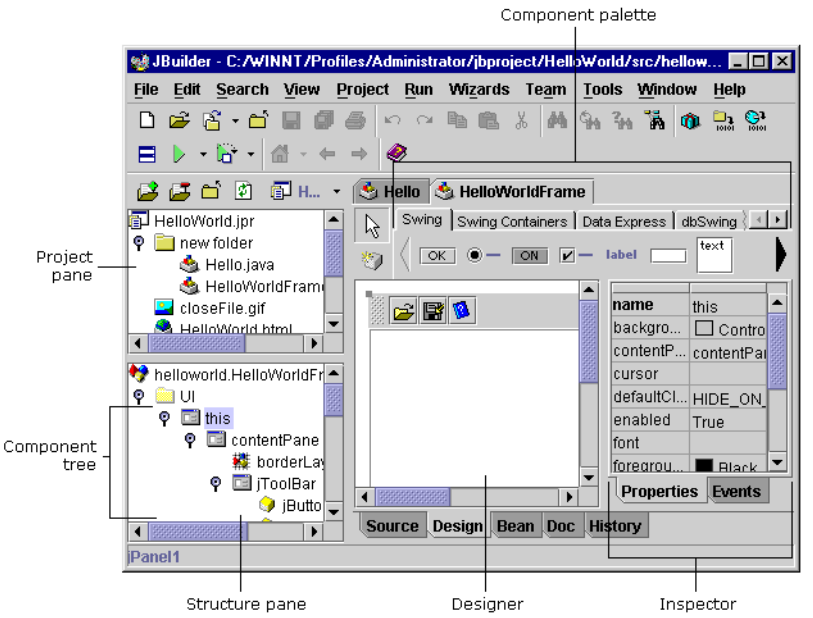


Table 5.1 JBuilder의 비주얼 디자인 툴

디자인 툴	설명
UI 디자이너	패널 및 기타 UI 컴포넌트를 배치하고 편집할 공간을 제공합니다. 열려 있는 파일에 대한 UI 디자이너를 보려면 컨텐트 창 하단의 Design 탭을 선택합니다.
컴포넌트 팔레트	비주얼 Java 컴포넌트와 넌비주얼 Java 컴포넌트가 들어 있습니다. 팔레트에 있는 컴포넌트는 JBuilder 에디션에 따라 다양합니다.
컴포넌트 트리	소스 파일에 있는 모든 컴포넌트 및 관계의 구조적 뷰를 표시합니다. AppBrowser 왼쪽 아래에 있는 구조 창에 나타납니다.
Inspector	컴포넌트 속성의 값을 검사하고 설정하여 메소드를 컴포넌트 이벤트에 연결하는 데 사용됩니다. Inspector에서 만들어진 변경 사항은 디자인에 비주얼하게 반영됩니다.
메뉴 디자이너	디자인 표면 상에서 메뉴를 디자인하는 데 사용됩니다. UI 디자이너에서 이를 호출하려면 컴포넌트 트리의 JMenuBar 또는 JPopupMenu 컴포넌트를 더블 클릭하거나 컴포넌트를 선택하고 *Enter*를 누릅니다.
열 디자이너	데이터 집합 컴포넌트를 비주얼하게 사용할 수 있게 합니다. 이를 호출하려면 컴포넌트 트리에서 데이터 집합을 더블 클릭합니다. JBuilder 전문가용 및 기업용에서 사용 가능합니다.

자세한 내용은 Chapter 23, "자습서: 애플리케이션 구축"을 참조하십시오. 또한 온라인 자습서, "Building a Java text editor", "Creating a UI with nested layouts", "GridBagLayout tutorial"도 참조할 수 있으며 *Building Applications with JBuilder*의 "Designing a user interface"도 참조하십시오.

UI 디자이너 사용

Java 클래스를 비주얼하게 디자인하고 프로그래밍할 수 있는 툴인 Jbuilder를 사용하여 복합적이거나 복잡한 새 컴포넌트를 만들 수 있습니다.

비주얼 디자인 툴을 사용하려면 파일이 다음과 같은 요구사항에 맞아야 합니다.

- Java 파일이어야 합니다(Inner 및 Anonymous 클래스는 예외).
- 구문 오류를 일으키지 않아야 합니다.
- 파일 이름과 일치하는 이름을 가진 클래스를 포함해야 합니다.

참고 JavaBeans는 java.awt.Container를 확장해야 합니다.

이러한 요구사항은 오브젝트 갤러리의 마법사를 사용하여 파일을 생성할 경우 모두 충족됩니다.

파일 보기

1. 프로젝트 창에서 Java 파일을 더블 클릭합니다. 파일은 컨텐트 창에 소스 편집기를 엽니다.
2. 컨텐트 창 하단의 Design 탭을 선택합니다. 파일은 디자인 보기 또는 디자이너로 변경됩니다. 이제 컴포넌트 팔레트와 Inspector가 사용 가능해집니다.

컴포넌트 추가 및 사용

- 컴포넌트 팔레트의 컴포넌트를 클릭하여 선택합니다.
- 구조 창의 해당 부모 또는 디자이너를 클릭하여 선택한 컴포넌트를 디자이너에 놓습니다.
- 구조 창의 컴포넌트 트리를 사용하여 UI 컴포넌트가 서로 관계를 맺고 있는 위치를 계속해서 추적합니다. 컴포넌트 트리의 컴포넌트를 자르기 또는 붙여넣기하여 원하는 형태로 쌓거나 중첩시킵니다.
- 레이아웃 매니저를 적용하려는 컨테이너를 선택한 다음 Inspector에서 layout을 선택하여 원하는 레이아웃 매니저를 선택 적용합니다.
- Inspector의 오른쪽 열 필드를 더블 클릭하여 사용 가능한 값을 보거나 텍스트 필드를 활성화합니다.

참고 JBuilder는 디자이너와 Java 소스 코드를 계속해서 동기화합니다. UI 디자이너에서 디자인을 변경하면 JBuilder는 자동으로 소스 코드를 업데이트하고 소스 코드를 변경하면 변경 사항을 UI 디자이너에 반영합니다.

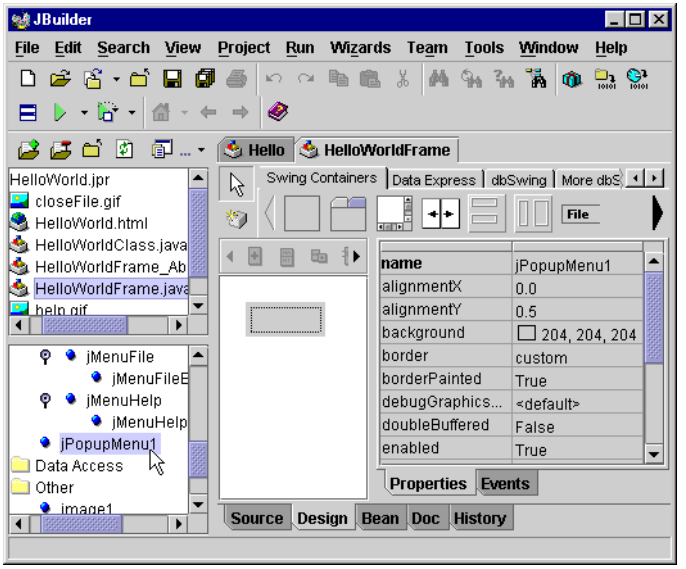
자세한 내용은 *JBuilder를 이용한 애플리케이션 구축*에서 "JBuilder의 비주얼 디자인 툴"의 "사용자 인터페이스 디자인"을 참조하십시오.

메뉴 디자인

JBuilder는 메뉴 생성을 쉽게 해주는 메뉴 디자이너를 포함합니다. 비주얼하게 드롭다운 및 컨텍스트 메뉴를 생성할 수 있습니다.

다음과 같은 방법으로 JBuilder의 메뉴 디자이너에 액세스합니다.

1. 프로젝트 창에서 Java 파일을 더블 클릭하여 엽니다.
2. 컨텐트 창 하단의 Design 탭을 선택하여 디자이너로 변경합니다.
3. 컴포넌트 팔레트에서 메뉴 컴포넌트를 클릭한 다음 디자인을 클릭하여 메뉴 컴포넌트를 추가합니다.
4. 컴포넌트 트리의 새 메뉴 컴포넌트를 더블 클릭하여 메뉴 디자이너를 활성화합니다.

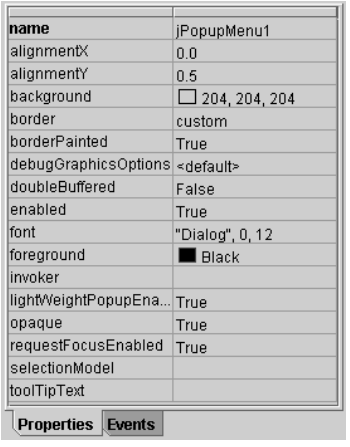


UI 디자이너로 되돌아가려면 컴포넌트 트리의 UI 폴더에 있는 임의의 컴포넌트를 더블 클릭합니다.

자세한 내용은 *JBuilder를 이용한 애플리케이션 구축*의 "메뉴 디자인"을 참조하십시오.

컴포넌트 속성 및 이벤트 설정

UI 디자이너의 Inspector를 통해 컴포넌트 속성을 비주얼하게 편집하고 컴포넌트 이벤트에 코드를 추가할 수 있습니다. Inspector에서 속성 및 이벤트에 변경 사항을 만들 수 있으며 해당 코드는 자동으로 소스 코드에 삽입됩니다.



자세한 내용은 *JBuilder를 이용한 애플리케이션 구축*의 "사용자 인터페이스 디자인"을 참조하십시오.

Inspector를 사용하여 다음을 수행할 수 있습니다.

- 컨테이너의 컴포넌트와 컨테이너 및 레이아웃 매니저에 대한 초기 속성 값을 설정합니다(초기화 코드).
- 컨테이너의 컴포넌트로부터 이벤트를 받는 컨테이너의 이벤트 리스너를 생성하고, 이름을 지정하며, 삭제합니다(이벤트 처리 코드).
- text 속성 String 값을 ResourceBundle에 저장하거나 리소스 String을 String 상수에 반환합니다.
- Inspector에 나타난 속성의 레벨을 변경합니다.
- Inspector에서 변경할 수 있도록 속성을 클래스 레벨 변수로 나타냅니다.

Inspector의 모든 변경 사항은 소스 코드와 UI 디자이너에 곧바로 반영됩니다.

자세한 내용은 *JBuilder를 이용한 애플리케이션 구축*의 "이벤트 처리"를 참조하십시오.

Inspector 열기

다음과 같은 방법으로 Inspector를 표시합니다.

1. 프로젝트 창에서 Java 파일을 선택하고 *Enter*를 눌러 컨텐트 창에 파일을 엽니다.
2. AppBrowser 하단의 Design 탭을 선택하여 디자이너에 액세스합니다. Inspector가 컨텐트 창의 오른쪽에 표시됩니다.
3. Inspector의 왼쪽 보더를 드래그하여 너비를 조정합니다.

자세한 내용은 *JBuilder를 이용한 애플리케이션 구축*의 "Inspector 사용"을 참조하십시오.

레이아웃 매니저를 이용한 레이아웃 디자인

Java로 작성된 프로그램은 하나 이상의 플랫폼에 배포될 수 있습니다. UI 컴포넌트에 대해 절대 위치와 크기를 지정하는 기존의 UI 디자인 기술을 사용한다면 모든 플랫폼에서 UI가 원하는 대로 표시되지 않을 수도 있습니다.자신의 배포 시스템에서는 잘 보여도 다른 플랫폼에서는 사용하지 못할 수 있습니다.이 문제를 해결하기 위해 Java는 이식가능한 레이아웃 매니저 시스템을 제공합니다.

레이아웃 매니저는 다음과 같은 이점을 제공합니다.

- 글꼴, 화면 해상도 및 플랫폼 차이점에 상관 없이 알맞게 위치하는 컴포넌트.
- 런타임 시 동적으로 크기가 조정되는 컨테이너에 대해 지능적인 컴포넌트 위치.
- 크기를 다르게 조정하는 문자열 변환의 용이성.문자열의 크기가 증가하면 컴포넌트는 적절하게 배치된 상태를 유지합니다.

JBuilder는 Java AWT 및 Swing으로부터 다음과 같은 레이아웃 매니저를 제공합니다.

- BorderLayout
- FlowLayout
- GridLayout
- CardLayout
- GridBagLayout
- Null

JBuilder 전문가용 및 기업용 버전은 또한 다음과 같은 사용자 지정 레이아웃을 제공합니다.

- XYLayout은 컨테이너에 넣은 컴포넌트의 원래 크기와 위치(x,y 좌표)를 유지합니다.
- PaneLayout은 컨테이너를 여러 창으로 나누는 데 사용됩니다.
- VerticalFlowLayout은 수평 대신 수직으로 컴포넌트를 배치한다는 것을 제외하고는 FlowLayout과 매우 유사합니다.
- BoxLayout2는 Swing의 BoxLayout에 대한 bean 랩퍼 클래스로서 Inspector에서 레이아웃으로 선택되도록 합니다.
- OverlayLayout2는 Swing의 OverlayLayout에 대한 bean 랩퍼 클래스로서 Inspector에서 레이아웃으로 선택되도록 합니다.

직접 사용자 지정 레이아웃을 생성하거나 java.awt 클래스, 새 레이아웃 매니저 또는 협력 업체의 레이아웃 매니저에서 그 밖의 다른 레이아웃을 시도해 볼 수 있습니다. 이 중 많은 레이아웃 매니저가 웹 상에서 퍼블릭 도메인이며, OpenSource 커뮤니티의 멤버로 액세스할 수 있습니다.UI 디자이너에서 사용자 지정 레이아웃을 사용하려면 UI 디자이너가 레이아웃을 사용하도록 Java helper 클래스 파일을 제공해야 합니다.

대부분의 UI 디자인에서는 레이아웃을 조합해서 사용하며 각각의 다른 레이아웃 패널을 중첩시킵니다.

자세한 내용은 *JBuilder를 이용한 애플리케이션 구축*의 "레이아웃 매니저 사용"을 참조하십시오.

Chapter

6

Java 프로그램 컴파일 및 실행

JBuilder 컴파일러는 내부 클래스 및 JAR(Java Archive) 파일을 포함하여 Java 언어를 완벽하게 지원합니다. IDE 내에서 컴파일(또는 "make")할 수 있습니다. JBuilder 전문가용 및 기업용에서는 **bmj**(Borland Maker for Java) 또는 **bcj**(Borland Compiler for Java)를 사용하여 명령 줄에서도 컴파일할 수 있습니다.

명령줄 도구에 대한 자세한 내용은 *JBuilder를 이용한 애플리케이션 구축*의 "명령줄 도구 사용"을 참조하십시오.

Run 명령은 프로젝트, 개별 파일(예: .java 파일 및 servlet), 애플릿을 컴파일하고 실행합니다.

JBuilder 전문가용 및 기업용에서는 실행할 파일 또는 프로그램의 종류(애플리케이션, 애플릿, JSP, servlet 또는 EJB)에 적합한 런타임 구성을 설정할 수 있습니다.

런타임 구성에 대한 자세한 내용은 *JBuilder를 이용한 애플리케이션 구축*의 "런타임 구성 설정"을 참조하십시오.

프로그램 컴파일 및 실행 시 JBuilder가 파일을 찾는 위치에 대한 자세한 내용은 *Building Applications with JBuilder*에서 "프로젝트 생성 및 관리" 장의 "How JBuilder constructs paths" 및 "Where are my files?"를 참조하십시오.

Java 프로그램 컴파일

Java 컴파일러는 Java 소스 파일을 읽어 들이고 컴파일에 필요한 추가 파일을 결정하며 Java Virtual Machine(VM)용 기계 코드인 바이트코드를 포함하는 .class 파일의 형태로 Java 프로그램을 생성합니다.

컴파일을 하면 소스 파일에 있는 각 클래스 선언과 인터페이스 선언에 대해 별도의 .class를 만듭니다. 특정 플랫폼에서 결과 Java 프로그램 실행 시 해당 플랫폼용 Java 인터프리터는 .class 파일에 포함된 바이트코드를 실행합니다.

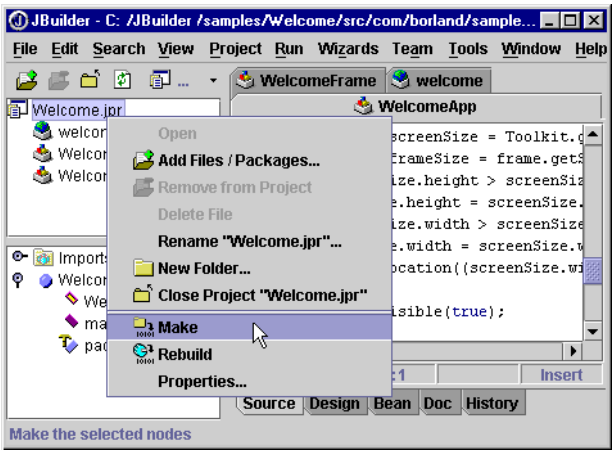
애플리케이션 또는 애플릿에 대한 소스 파일을 컴파일하기 위해서는 다음과 같은 단계를 따릅니다.

1 프로젝트를 엽니다.

2 다음 중 하나를 수행하십시오.

- Project|Make Project를 선택합니다.
- 프로젝트 창에서 프로젝트 파일(.jpr 또는 .jpx)를 마우스 오른쪽 버튼으로 클릭하고 Make를 선택합니다.

참고 프로그램의 경우 메인 클래스는 main() 메소드를 포함해야 하며 애플릿의 경우 메인 클래스는 init() 메소드를 포함해야 합니다. 실행할 메인 클래스를 설정하지 않은 경우 Project Properties 대화 상자(Project|Project Properties)의 Run 페이지가 나타납니다. 메인 클래스를 찾아서 선택한 다음 계속해서 컴파일합니다. 자세한 내용을 보려면 Run 페이지의 Help 버튼을 선택합니다.



컴파일러 오류 메시지가 있는 경우 AppBrowser 컨텐트 창 하단의 메시지 창에 표시됩니다.관련 소스 코드를 표시하려면 오류 메시지를 선택합니다.오류 메시지에 대한 도움말을 얻으려면 메시지 창에서 오류 메시지를 선택하고 F1을 누릅니다.JBuilder에서 오류 메시지에 대해 알아 보려면 *Jbuilder로 애플리케이션 구축*에서 "오류 및 경고 메시지"를 참조하십시오.

자세한 내용은 *JBuilder를 이용한 애플리케이션 구축*에서 "Java 프로그램 컴파일"을 참조하십시오.

Java 프로그램 실행

프로젝트를 실행하면 해당 프로젝트 파일의 메인 클래스가 실행됩니다. 메인 클래스가 선택되지 않은 경우에 실행을 시도하면 선택을 할 수 있는 대화 상자가 표시됩니다. JBuilder 마법사를 사용하여 파일을 생성한 경우에는 자동으로 메인 클래스가 설정됩니다. Project|Project Properties를 선택하고 Run 탭을 선택하여 메인 클래스를 선택 또는 변경할 수 있습니다.

참고 Project Properties(Project|Project Properties)의 Run 페이지에 어떤 항목이 설정되어 있는지에 관계없이 Run|Run Project 및 Run 버튼이 실행됩니다.

.java 파일을 실행하려면 main() 메소드를 포함해야 합니다.

다음과 같은 방법으로 JBuilder에서 프로그램을 실행합니다.

1. 실행하려는 프로그램 또는 파일을 저장합니다.
2. 프로젝트 창에서 프로그램 또는 파일을 선택합니다.
3. Project|Make Project를 선택하여 컴파일합니다.
4. 다음 방법 중 하나를 사용하여 프로그램을 실행합니다.
   - Project|Run Project를 선택합니다.
   - 프로젝트 창에서 파일을 마우스 오른쪽 버튼으로 클릭하고 드롭다운 메뉴에서 Run을 선택합니다.
   -  툴바에서 Run 버튼 을 클릭합니다.

애플릿은 다르게 동작하는데 일반적으로 main() 메소드를 포함하지 않기 때문입니다. HTML 파일은 <applet> 태그의 code 속성에서 발견되는 클래스를 호출합니다. 이 애플릿 클래스는 init() 메소드를 포함해야 합니다.

JBuilder에서 애플릿을 실행하는 데는 두 가지 방법이 있습니다.

- JBuilder의 appletviewer, AppletTestbed를 사용하여 메인 클래스에서 실행하는 방법
- Sun의 **appletviewer**를 사용하여 <applet> 태그를 사용하는 HTML 파일에서 실행하는 방법

  실행 클래스 또는 HTML 파일 설정에 대한 자세한 내용은 Project Properties 대화 상자 (Project|Project Properties) 에서 Run 페이지의 Help 를 선택합니다 .

JBuilder에서 애플릿을 실행하려면 다음 중 하나를 수행하십시오.

- Project|Run Project를 선택하여 Project Properties 대화 상자 (Project|Project Properties)에서 Run 페이지의 Applet 탭에 설정된 HTML 파일 또는 클래스 집합을 실행합니다.
-  툴바의 Run 버튼을 클릭하여 Project Properties 대화 상자(Project|Project Properties)에서 Run 페이지의 Applet 탭에 설정된 HTML 파일 또는 클래스 집합을 실행합니다.

- 프로젝트 창에서 <applet> 태그를 포함하는 애플릿 HTML 파일을 마우스 오른쪽 버튼으로 클릭하고 Run을 선택하면 Sun의 **appletviewer**에서 애플릿을 실행합니다.

일단 프로그램이 컴파일되었으면 컴파일하지 않고 실행할 수 있습니다. 기본적으로 변경된 파일은 재빌드되지만 다시 컴파일하지 않는 한 전체 빌드가 수행되지는 않습니다.

런타임 오류 메시지가 AppBrowser 컨텐트 창 하단의 메시지 창에 표시됩니다.관련 소스 코드를 표시하려면 오류 메시지를 선택합니다.오류 메시지에 대한 도움말을 얻으려면 메시지 창에서 오류 메시지를 선택하고 F1을 누릅니다.JBuilder에서 오류 메시지에 대해 알아 보려면 *Jbuilder를 이용한 애플리케이션 구축*에서 "오류 및 경고 메시지"를 참조하십시오.

프로그램 실행에 대한 자세한 내용은 *JBuilder를 이용한 애플리케이션 구축*의 "Java 프로그램 실행"을 참조하십시오.

애플릿에 대한 자세한 내용은 *웹 애플리케이션 개발자 안내서*의 "애플릿 작업"을 참조하십시오.

JSP 및 servlet 실행에 대한 내용은 *웹 애플리케이션 개발자 안내서*의 "JBuilder에서의 웹 애플리케이션 작업" 및 "서블릿 작업"을 참조하십시오.

Java 프로그램 디버깅

디버깅은 프로그램에서 오류를 찾고 고치는 작업입니다. JBuilder의 통합 디버거를 통해 JBuilder 환경 내에서 애플리케이션 및 애플릿을 디버깅할 수 있습니다. JBuilder 기업용은 JSP 디버깅, 교차 프로세스(cross-process) 디버깅, 원격 디버깅을 지원하는 반면 JBuilder 전문가용은 서블릿 디버깅도 지원합니다.

Run 메뉴를 통해 다양한 디버거 기능에 액세스할 수 있습니다. 다른 디버거들은 Search, View, Tools 메뉴에서 사용할 수 있습니다. CodeInsight(Tools|Editor Options) 및 구문 강조 기능은 소스 코드 디버깅을 더 쉽게 해줍니다.

디버깅에 대한 자세한 내용은 *JBuilder를 이용한 애플리케이션 구축*의 "Java 프로그램 디버깅" 또는 *Distributed Application Developer's Guide*의 "Debugging distributed applications"를 참조하십시오.

CodeInsight 및 구문 강조에 대한 자세한 내용은 *JBuilder를 이용한 애플리케이션 구축*의 "JBuilder 환경"을 참조하십시오.

파일 또는 전체 프로젝트를 디버깅할 수도 있는데 디버깅 전에 컴파일하거나 컴파일하지 않는 옵션이 있습니다.

디버깅 전의 컴파일 수행 여부를 선택하려면 Project|Project Properties를 선택하고 Run 탭을 선택합니다. 대화 상자의 하단에 있는 Compile Before Debugging 옵션을 선택합니다. JBuilder 전문가용에서는 Smart Step 사용 여부 및 사용 방법을 선택할 수 있습니다. JBuilder 기업용에서는 원격 디버깅 사용을 선택하여 적합하게 설정할 수 있습니다. 이 중 하나

를 수행하려면 Project|Project Properties를 선택하고 Debug 탭을 선택합니다.

파일을 디버그하려면 프로젝트 창의 파일을 오른쪽 마우스 버튼으로 클릭하고 컨텍스트 메뉴에서 Debug를 선택합니다. 소스 코드에 브레이크포인트를 설정하려면 Run|Add Breakpoint를 선택하고 소스 파일 코드의 실행 줄 왼쪽에 있는 회색 여백을 클릭하거나 선택한 편집기 에뮬레이션의 키스트로크 순서(Tools|Editor Options|Editor|Keymap)를 따릅니다.

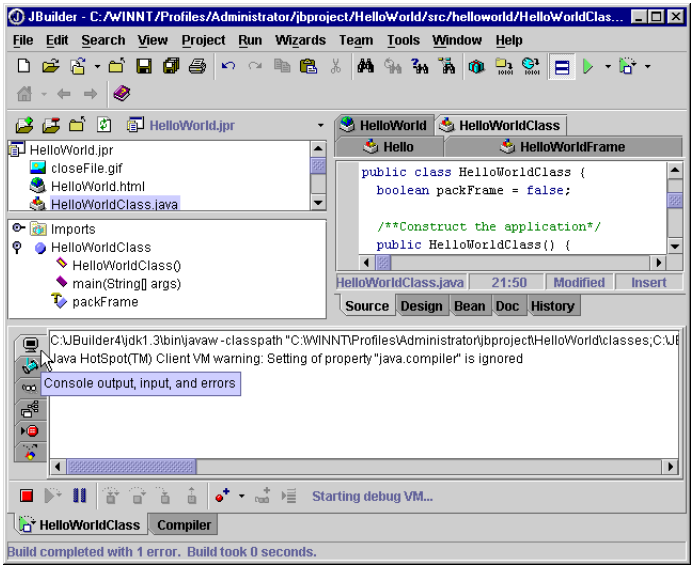
프로젝트를 디버깅하려면 다음 단계를 따르십시오.

1. 프로젝트를 엽니다.
2. Project|Project Properties를 선택합니다. Run 탭을 선택하고 JBuilder에서 디버깅 전에 컴파일할지 여부를 결정합니다.
3. 소스 코드에 브레이크포인트를 설정하려면 Run|Add Breakpoint를 클릭하고 소스 파일 코드의 실행 줄 왼쪽에 있는 회색 여백을 클릭하거나 선택한 편집기 에뮬레이션의 키스트로크 순서(Tools|Editor Options|Editor|Keymap)를 따릅니다.

4. Run|Debug Project를 선택하거나 툴바의 Debug 아이콘을 클릭합니다.

컴파일러 및 디버거는 파일 및 프로젝트 모두에 동일한 방식으로 작동합니다.

- 디버깅 전에 컴파일하도록 JBuilder를 설정한 경우 AppBrowser 하단의 메시지 창에 있는 컴파일러 페이지에 컴파일러 오류가 표시됩니다. 오류 메시지를 클릭하여 코드의 해당 줄로 이동합니다.
- 컴파일러를 사용하지 않거나 오류가 없는 경우 디버거가 메시지 창에 나타납니다. 왼쪽 탭의 툴팁을 사용하여 디버거가 제공하는 정보를 참조하십시오.



Java 프로그램 배포

Java 프로그램 배포는 프로젝트에서 필요로 하는 다양한 Java 클래스 파일, 이미지 파일 및 기타 파일을 번들로 구성하며 사용자가 액세스할 수 있는 서버 또는 클라이언트 컴퓨터에 이러한 파일을 복사합니다. 배포되는 프로그램은 압축 또는 압축 해제된 아카이브 파일(일반적으로 JAR 파일)의 형태로 전달할 수 있습니다. JBuilder 기업용 및 전문가용 에디션은 아카이브를 생성하는 Archive Builder를 제공합니다. Sun에서도 아카이브 생성용 도구인 jar 도구를 제공합니다.

참고 JBuilder 제품 라이센스에서 재배포할 수 있거나 없는 항목에 대한 정보는 jbuilder 디렉토리의 license.txt 파일 및 redist/deploy.txt를 참조하십시오.

자세한 내용은 *JBuilder를 이용한 애플리케이션 구축*의 "Java 프로그램 배포"를 참조하십시오.

JAR 파일로 프로그램을 배포하고 실행하는 것에 대한 자습서는 http://java.sun.com/docs/books/tutorial/jar/basics/index.html의 "Tutorial:Building a Java text editor," 및 Chapter 24, "자습서:애플릿 만들기"을 참조하십시오.

Archive Builder 사용

이것은 JBuilder 전문가용 및 기업용 버전의 기능입니다.

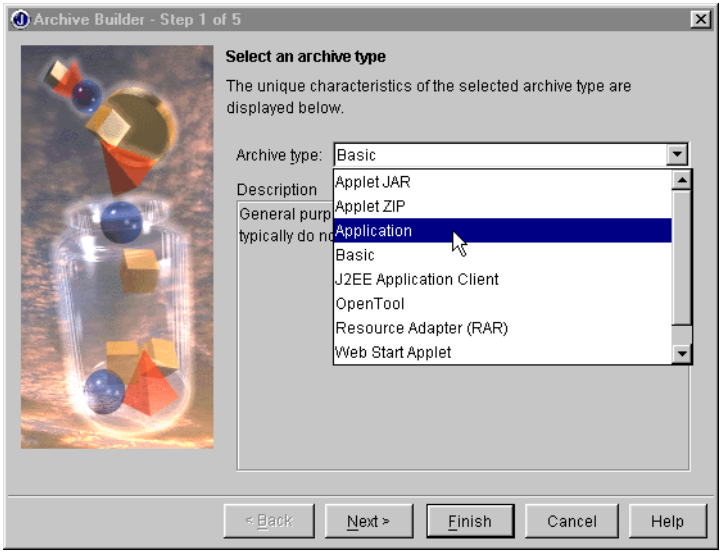
JBuilder Archive Builder는 프로그램이 필요로 하는 클래스 및 파일을 자동으로 모읍니다. 그런 다음 파일을 압축 또는 압축 해제된 아카이브로 번들로 구성합니다. 또한 JBuilder에서 수정할 수 있는 아카이브의 목록 파일을 생성할 수 있습니다.

Archive Builder는 또한 프로젝트에 아카이브 노드를 생성하여 아카이브 파일에 쉽게 액세스할 수 있도록 합니다.개발 도중에 언제라도 아카이브 파일을 만들고 다시 구축하거나 그 속성을 재설정할 수 있습니다.또한 목록(manifest) 파일의 내용은 물론 아카이브 파일의 내용도 볼 수 있습니다.

다음과 같은 방법으로 프로그램을 배포합니다.

1. JBuilder에서 코드를 작성하고 컴파일합니다.
2. Archive Builder(Wizards|Archive Builder)를 실행하여 아카이브 파일을 생성합니다.
3. 설치 프로시저를 만듭니다.

4 JAR 파일, 재배포할 수 있는 모든 필요한 JAR 파일 및 설치 파일을 배포합니다.



Archive Builder에 대한 자세한 내용은 *JBuilder를 이용한 애플리케이션 구축*의 "Archive Builder 사용"을 참조하십시오.

CORBA 애플리케이션 배포

이것은 JBuilder 기업용 버전의 기능입니다.

JBuilder 기업용을 이용하여 CORBA 애플리케이션을 배포 시 Archive Builder는 스텁 및 뼈대 코드를 JAR 파일에 모읍니다. 클라이언트, 중간 계층, 서버 CORBA 프로그램을 실행하는 각 시스템에 ORB를 설치해야 합니다.

자세한 내용은 VisiBroker를 사용하는 경우 *VisiBroker for Java Programmer's Guide*의 "Deploying Applications with VisiBroker"를 참조하거나 애플리케이션 서버의 *사용자 안내서*를 참조하십시오.

웹 기반 애플리케이션 배포

이것은 JBuilder 전문가용 및 기업용 버전의 기능입니다.

웹 기반의 다계층 애플리케이션은 웹 서버에 배포됩니다. 웹 애플리케이션 배포에 대한 내용은 웹 서버 설명서를 참조하십시오.

서블릿, JSP, 전체 웹 애플리케이션에 관한 내용은 *웹 애플리케이션 개발자 안내서*의 "웹 애플리케이션 배포"를 참조하십시오.

*웹 애플리케이션 개발자 안내서*의 "애플릿 작업" 및 *JBuilder를 이용한 애플리케이션 구축*의 "Java 프로그램 배포"를 참조하십시오.

배포된 프로그램 실행

JDK 명령줄 도구를 사용하여 명령 줄에서 배포된 프로그램을 실행할 수 있습니다. 명령 줄에서 프로그램을 실행하려면 **java** 명령과 함께 **-jar** 옵션을 사용합니다.

자세한 내용은 *JBuilder를 이용한 애플리케이션 구축*의 "JAR 파일에서의 프로그램 실행", "명령줄 도구 사용", "Java 프로그램 배포"를 참조하십시오.

JAR 파일에서의 프로그램 실행에 대한 자습서는 http://java.sun.com/docs/books/tutorial/jar/basics/index.html의 "Tutorial:Building a Java text editor," 및 Chapter 24, "자습서:애플릿 만들기"을 참조하십시오.

명령줄 도구 사용

명령줄 도구를 통해 명령줄 창에서 전역 명령을 실행할 수 있습니다. 표준 명령줄 도구를 사용하여 애플리케이션을 컴파일하고 시작할 수 있으며 JAR 파일을 관리하고 웹 브라우저 외부의 애플릿을 볼 수 있으며, 코드에 포함된 주석을 추출할 수 있습니다. JBuilder는 확장 및 개선된 기능을 제공하는 추가의 명령줄 도구를 제공합니다.

JDK에는 다음과 같은 명령줄 도구가 들어 있습니다.

- **javac** - Java 프로그램 언어용 컴파일러
- **java** - Java 애플리케이션용 런처.
- **jar** - Java Archive(.jar) 파일을 관리합니다.
- **javadoc** - 코드 주석을 추출하여 HTML 문서를 생성합니다.
- **appletviewer** - 웹 브라우저 컨텍스트 외부의 애플릿을 실행할 수 있게 해줍니다.
- **native2ascii** - 인코딩된 원시 문자 파일을 유니코드 이스케이프 시퀀스를 가진 파일로 변환합니다.

Sun의 명령줄 도구에 대한 내용은 http://java.sun.com/j2se/1.3/docs/tooldocs/tools.html#basic을 참조하십시오.

모든 JBuilder 에디션은 다음과 같은 옵션을 포함하는 명령줄 인터페이스를 사용합니다.

- 프로젝트 구축
- 구성 정보 표시
- 라이센스 관리자 표시
- 시작 화면 비활성화
- OpenTools 작성자를 위한 verbose 디버깅 모드 활성화

참고 이러한 옵션은 에디션에 따라 다양합니다.

JBuilder의 편집에 사용할 수 있는 옵션 목록에 액세스하려면 명령줄 창을 열고 JBuilder bin 디렉토리로 이동하여 jbuilder -help를 입력합니다.

JBuilder는 실행되는 플랫폼에 따라 쉘 스크립트, 배치 파일, 실행 파일 등의 자신의 고유한 런처에서 실행됩니다.이러한 각 런처들은 JBuilder에 인수를 전달할 수 있습니다.

JBuilder 전문가용 및 기업용에는 추가적인 명령줄 도구가 포함됩니다.

- **bmj** 명령줄 make
- **bcj** 명령줄 compiler

Sun 및 JBuilder의 명령줄 도구 사용에 대한 자세한 내용은 *JBuilder를 이용한 애플리케이션 구축*의 "명령줄 도구 사용"을 참조하십시오.

JBuilder 명령줄 인수에 대한 자세한 내용은 *JBuilder를 이용한 애플리케이션 구축*의 "JBuilder 명령줄 인수"를 참조하십시오.

Chapter

7

JBuilder를 이용한 팀 개발

개발자 또는 개발자 그룹은 파일 개정 추적 및 저장으로부터 이점을 얻을 수 있습니다. 그렇게 하면 전체 개발 수명 주기에 걸쳐 코드가 보호되고 필요 시 이전 코드를 참조할 수 있습니다. 이러한 작업을 수행하는 도구를 버전 제어 시스템(VCS) 또는 코드 관리(CM) 도구라고 합니다.

JBuilder의 버전 제어 시스템 통합은 문맥에 맞아야 합니다. 이는 실제로 다음 두 가지를 의미합니다.

1. IDE에는 메시지 대화 상자, 마법사 및 유용한 공지가 풍부하므로 필요 시 도움을 얻을 수 있으며 작업 진행을 확인할 수 있습니다.
2. 현재 상황에서 수행 가능한 경우에만 옵션을 사용할 수 있습니다.

버전 제어 도구

우수한 버전 제어 시스템은 개정 정보 유지 이상의 일을 합니다. 버전 제어 시스템은 코드 관리를 위한 강력한 기능과 유연성을 제공해야 합니다. 대부분의 VCS에서는 다음을 할 수 있습니다.

- 개발 주기의 어느 시점에서든지 전체 프로젝트의 스냅샷을 얻습니다.
- 코드의 여러 스트림을 동시에 개발할 수 있습니다.

다음 예제를 살펴봅니다.

소프트웨어 회사가 ReallyCoolSoftware라는 제품을 개발한다고 가정합니다. ReallyCoolSoftware의 각 베타 버전 출하 시 제품 관리자는 프로젝트의 스냅샷을 얻는 *버전 레이블*을 적용합니다. 이를 통해 서로 다른 파일이 각기 다른 횟수로 개정되었어도 각 버전 출하 시 프로젝트에서 ReallyCoolSoftware가 가졌던 코드베이스를 누구든지 즉시 참조할 수 있습니다. 개발된 제품을 릴리스할 때 관리자는 코드를 *분기*하여 두 개의 스트림을 동시에 개발할 수 있도록 합니다. 이를 통해 하나의 분기에서는 필

요 시 패치를 개발하고 다른 분기에서는 성숙한 제품을 계속 개발할 수 있습니다.

또한 버전 제어 시스템은 두 명이상의 개발자에 의해 변경 중인 파일을 처리할 수 있게 해줍니다. 대부분의 버전 제어 시스템은 모든 개정의 기록 및 각 개정이 발생한 파일 위치와 함께 각 파일의 현재 마스터 복사본을 유지 관리합니다. 하나의 마스터 복사본을 저장하고 각 변경 기록을 유지함으로써 VCS는 가능한 적은 공간을 차지하면서 필요한 모든 정보를 보유합니다.

JBuilder 기업용 에디션은 세 가지 주요 VCS인 Concurrent Versions System(CVS), Visual SourceSafe, Rational ClearCase를 가진 인터페이스를 제공합니다. CVS는 JBuilder 디렉토리에 자동으로 설치됩니다.

JBuilder의 인터페이스를 통해 대부분의 일반적인 버전 제어 작업을 JBuilder에서 바로 수행할 수 있습니다. JBuilder의 Version Control OpenTool과 확장 가능한 개방형 아키텍처를 통해 다른 버전 제어 시스템도 통합할 수 있습니다.

지원되는 버전 제어 시스템에서 프로젝트를 끌어오려면 File|New를 선택하고 Team tab을 선택한 후 프로젝트를 끌어오려는 버전 제어 시스템을 선택한 다음 OK를 클릭하거나 Enter를 누릅니다. 그러면 연결을 구성할 마법사를 불러 오고 버전 제어 하에 있는 프로젝트를 작업하는 데 필요한 모든 것을 설정합니다.

JBuilder의 버전 제어에 대한 자세한 내용은 Part I, "Team Development using JBuilder" of the *Enterprise Application Developers Guide*의 "Welcome to team development with JBuilder"를 참조하십시오.

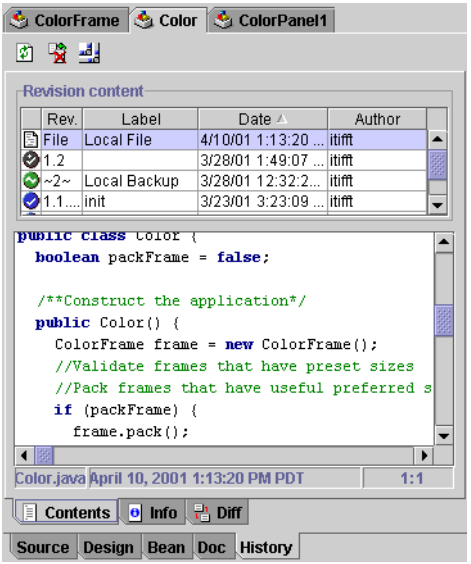
히스토리 뷰

JBuilder는 통합된 버전 제어 시스템 사용 여부에 상관 없이 파일 개정을 유지할 수 있게 해주는 히스토리 뷰의 개정 처리 기능을 제공합니다. JBuilder는 백업 시스템을 사용하여 이 작업을 수행합니다. 유지할 백업 레벨 수와 JBuilder가 백업을 저장할 위치를 설정할 수 있습니다. History 페이지에서 사용할 수 있는 기능은 에디션에 따라 다릅니다.

Contents 페이지

모든 JBuilder 에디션

사용 가능한 모든 버전의 활성 파일을 표시합니다.상단에 있는 개정 목록을 통해 버전 타입, 개정 번호, 레이블, 바뀐 날짜 또는 작성자 별로 파일 버전을 정렬할 수 있습니다. 개정 목록의 버전을 선택하여 소스 뷰어에서 해당 소스를 봅니다. Contents 페이지에서 Refresh Revision Info 및 Revert To Previous Revision 버튼에 액세스할 수 있습니다.

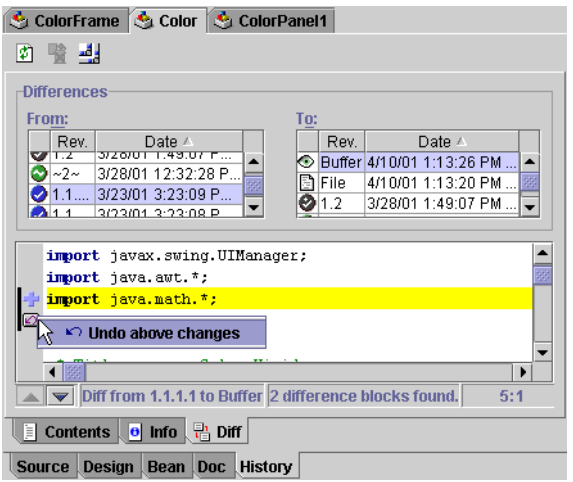


Diff 페이지

JBuilder 전문가용 및 기업용

활성 파일의 선택한 두 버전 간의 차이점을 표시합니다.상단에 있는 개정 목록을 통해 버전 타입, 개정 번호 또는 바뀐 날짜별로 파일 버전을 정렬할 수 있습니다. 이 페이지에서 Refresh Revision Info 버튼에 액세스할 수 있습니다.

Diff 페이지에는 From과 To에 두 개의 개정 목록이 있습니다. From 목록에서 버전을 선택하고 To 목록에서 다른 버전을 선택합니다. 목록 아래에 있는 소스 뷰어에 차이점이 표시됩니다.

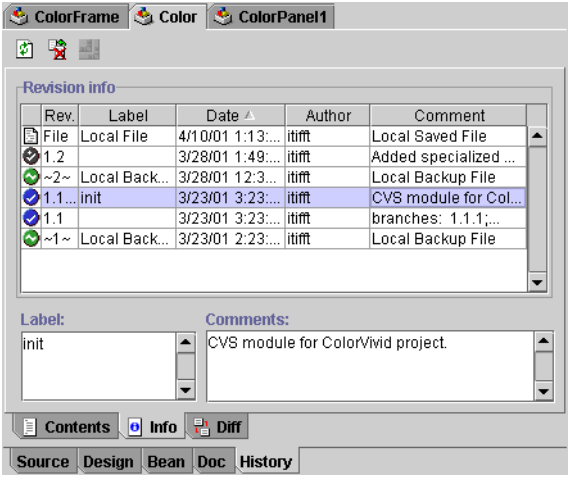


Info 페이지

JBuilder 기업용

레이블의 전체 텍스트와 활성 파일의 선택한 버전에 대한 로그를 표시합니다.상단에 있는 개정 목록을 통해 버전 타입, 개정 번호, 레이블, 바뀐 날짜, 작성자 또는 주석별로 파일 버전을 정렬할 수 있습니다. 이 페이지에서 Refresh Revision Info 및 Revert To Previous Revision 버튼에 액세스할 수 있습니다.

Info 페이지를 사용하려면 개정 목록에서 버전을 선택합니다. 페이지의 하단에 버전 레이블과 개정 주석이 표시됩니다. 스크롤 바를 사용하여 주석과 레이블의 전체 텍스트를 볼 수 있습니다.



History 페이지에 대한 자세한 내용은 *JBuilder를 이용한 애플리케이션 구축*의 "히스토리 뷰 사용"을 참조하십시오.

Chapter

8

웹 애플리케이션 구축

웹 개발은 JBuilder 전문가용과기업용의 기능입니다.

애플릿 개발은 JBuilder의 모든 에디션에 해당하는 기능입니다.

JBuilder는 웹 애플리케이션을 개발하고 배포하는 작업을 매우 쉽게 만듭니다. 웹 애플리케이션은 HTML/XML 문서, 웹 컴포넌트(servlets 및 JavaServer Pages), 디렉토리 구조 또는 Web ARchive(WAR) 파일 저장 형식의 그 외 리소스로 이루어진 컬렉션입니다. 웹 애플리케이션은 중앙 서버에 있으며 다양한 클라이언트에게 서비스를 제공합니다.

웹 애플리케이션은 HTML, XML, Java servlets, JavaServer Pages(JSP) 및 애플릿 같은 기술을 일부 또는 전부 포함합니다. JBuilder는 이러한 모든 기술의 사용을 위한 도구를 제공합니다.

또한 JBuilder는 data-aware 웹 애플리케이션인 InternetBeans Express를 쉽게 개발할 수 있도록 컴포넌트 집합을 제공합니다. InternetBeans Express는 servlet 또는 JSP 기술을 사용하여 정적 HTML 페이지에서 제공하는 템플릿에 기반한 데이터의 동적 표현을 생성합니다.

웹 애플리케이션 개발에 대한 설명의 주요 출처는 *웹 애플리케이션 개발자 안내서*입니다.

웹 서버

웹 애플리케이션은 웹 서버 내에서 실행됩니다. Tomcat 웹 서버는 JBuilder 전문가용 및 기업용과 함께 제공됩니다. JBuilder를 Tomcat과 함께 통합하면 매끄럽게 웹 애플리케이션을 컴파일, 실행 및 테스트할 수 있습니다. 또한 JBuilder는 여러 다른 웹 서버에 플러그인을 제공합니다. 자세한 내용은 *웹 애플리케이션 개발자 안내서*의 "웹 서버 구성"을 참조하십시오.

애플릿 작업

애플릿은 인터넷/인트라넷 웹 서버에 저장되는 Java 프로그램입니다. 애플리케이션과 달리 애플릿은 독립형 프로그램이 아니며 웹 브라우저와 같은 실행할 뷰어를 필요로 합니다. 애플릿은 <applet> 태그를 포함하는 HTML 웹 페이지에서 실행해야 합니다.

애플릿을 개발하기 전에 브라우저와 JDK 호환 문제를 완전히 이해하는 것이 중요합니다. 이러한 문제에 대한 내용은 *웹 애플리케이션 개발자 안내서*의 "애플릿 작업"과 Chapter 24, "자습서:애플릿 만들기"를 참조하십시오.

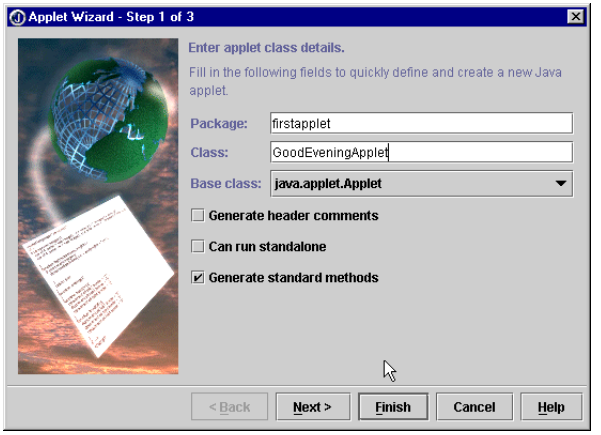
Applet 마법사 사용

JBuilder는 애플릿 생성을 돕는 Applet 마법사를 제공합니다. Applet 마법사는 다음 두 개의 파일로 구성된 애플릿을 만들고 기존 프로젝트에 추가합니다.

- 애플릿 클래스를 참조하는 <applet> 태그를 포함하는 HTML 파일. 이 파일은 애플릿을 실행하거나 디버그하는 데 선택해야 하는 파일입니다.
- Applet 또는 JApplet을 확장하는 Java 클래스. 이 클래스는 UI 디자이너를 사용하여 UI 컴포넌트를 추가하는 메인 UI 컨테이너입니다.

다음과 같은 방법으로 Applet 마법사를 엽니다.

1. File|New Project를 선택하고 Project 마법사를 완료하여 애플릿의 새 프로젝트를 만듭니다.
2. File|New를 선택하여 객체 갤러리를 엽니다.
3. 객체 갤러리의 Web 탭을 선택하고 Applet 아이콘을 더블 클릭합니다. (JBuilder 개인용의 New 페이지에 Applet 아이콘이 있습니다.)
4. Applet 마법사를 완료하여 애플릿을 만듭니다. 애플릿 HTML과 .java 파일이 프로젝트에 추가됩니다.



Applet 마법사에 대한 자세한 내용은 *웹 애플리케이션 개발자 안내서*의 "애플릿 작업" 장에서 "Applet 마법사를 이용한 애플릿 생성"을 참조하십시오.

애플릿 배포에 대한 자세한 내용은 *JBuilder를 이용한 애플리케이션 구축*의 "Java 프로그램 배포"와 *웹 애플리케이션 개발자 안내서*의 "애플릿 작업"을 참조하십시오.

서블릿

Java 서블릿은 웹 기반 애플리케이션 구축을 위한 프로토콜과 플랫폼 무관한 메소드를 제공합니다. 서블릿은 애플릿과 달리 웹 서버 내에서 실행되며 그래픽 사용자 인터페이스를 필요로 하지 않습니다. 서블릿은 요청과 응답을 통해 웹 서버에서 실행되는 서블릿 엔진과 상호 작용합니다. 모든 프로그램 언어로 작성될 수 있는 클라이언트 프로그램은 웹 서버에 액세스하고 요청을 만듭니다. 그런 다음 요청은 서블릿에 응답을 반환하는 웹 서버의 서블릿 엔진에 의해 처리되며 서블릿은 클라이언트에 응답을 보냅니다.

서블릿에 대한 자세한 내용은 *웹 애플리케이션 개발자 안내서*의 다음과 같은 장을 참조하십시오.

- "서블릿 작업"
- "JBuilder에서 서블릿 생성"
- "자습서: 간단한 서블릿 생성"
- "자습서: 방명록을 업데이트하는 서블릿 생성"

JSP

JavaServer Page(JSP) 기술을 통해 웹 개발자와 디자이너는 기존 비즈니스 시스템에 영향을 주는 풍부한 정보의 동적인 웹 페이지를 신속하게 개발하고 쉽게 유지 관리할 수 있습니다. Java 패밀리에 속하는 JSP 기술은 플랫폼과 무관한 웹 기반 애플리케이션을 신속하게 개발할 수 있게 해줍니다. JBuilder는 JSP를 쉽게 만들 수 있게 해 주는 마법사를 제공합니다. JSP에 대한 자세한 내용은 *웹 애플리케이션 개발자 안내서*의 "JavaServer Page(JSP) 개발"을 참조하십시오. JSP 생성에 대해 단계별로 설명하는 자습서를 이용할 수도 있습니다."자습서: JSP 생성."

InternetBeans

InternetBeans Express 기술은 서블릿 및 JSP 기술과 통합하여 애플리케이션에 값을 추가하고 서블릿 및 JSP 개발 작업을 단순화시킵니다. InternetBeans Express는 웹 애플리케이션의 프리젠테이션 레이어를 생성하고 그에 응답하기 위한 컴포넌트와 JSP 태그 라이브러리의 집합입니다. InternetBeans Express는 정적 템플릿 페이지를 취하며 라이브 데이터 모델의 동적 컨텐트를 삽입하고 이 컨텐트를 클라이언트에게 제공한 다

음 클라이언트로부터 게시된 변경 내용을 데이터 모델에 다시 씁니다. InternetBeans에 대한 자세한 내용은 *웹 애플리케이션 개발자 안내서*의 "InternetBeans Express 사용"을 참조하십시오. 웹 애플리케이션 개발자 안내서에는 다음 두 가지의 InternetBeans 자습서가 들어 있습니다.

- 자습서: InternetBeans Express를 이용한 서블릿 생성
- 자습서: InternetBeans Express를 이용한 JSP 생성

웹 애플리케이션 배포

웹 애플리케이션을 개발할 때 애플리케이션의 컨텍스트, WebApp 생성을 고려해야 합니다. WebApp는 웹 애플리케이션의 구조를 나타내는, 애플리케이션에 사용한 웹 컨텐트를 포함하는 디렉토리 트리입니다. Web ARchive(WAR) 파일에는 웹 서버에서 제공하는 파일이 들어 있으며 web.xml Deployment Descriptor는 이러한 파일에 대한 정보를 설명합니다. 배포 문제에 대한 자세한 내용은 *웹 애플리케이션 개발자 안내서*의 "WebApps와 WAR 파일 사용"을 참조하십시오.

Chapter

9

Enterprise JavaBeans 개발

이것은 JBuilder 기업용 버전의 기능입니다.

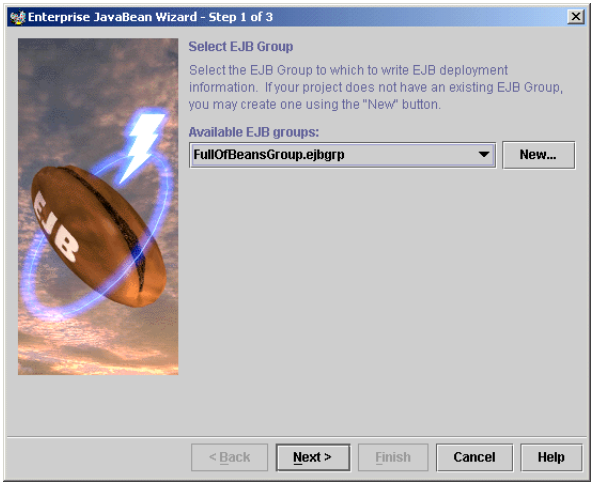
JBuilder를 통해 "Enterprise JavaBeans 1.1 사양"(http://java.sun.com/products/ejb/docs.html)을 순응하는 Enterprise JavaBeans (EJBs)를 쉽게 만들 수 있습니다. JBuilder를 통해 다음과 같은 애플리케이션 서버에서 실행되는 EJB를 만들 수 있습니다.

- Borland AppServer 4.5
- Borland Application Server 4.1
- BEA WebLogic Server 6.0
- BEA WebLogic Server 5.1
- IBM WebSphere 3.5

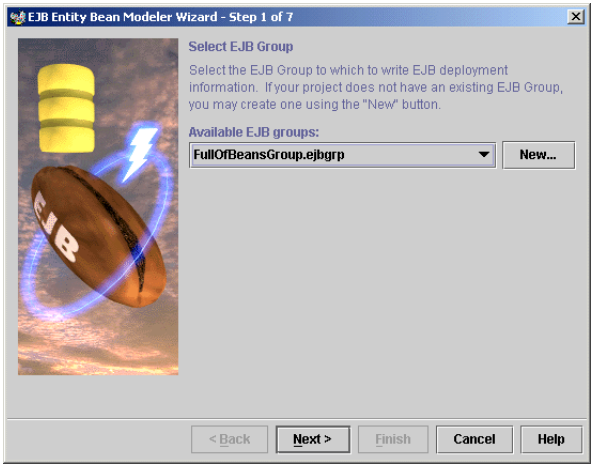
EJB 마법사를 이용한 기업용 bean 생성

JBuilder에서 기업용 bean을 생성하는 세 가지 마법사에는 Enterprise JavaBean 마법사, EJB Entity Bean Modeler와 EJB Bean Generator가 있습니다.

Enterprise JavaBean 마법사는 세션 bean 또는 엔티티 bean을 생성하는 데 사용됩니다.

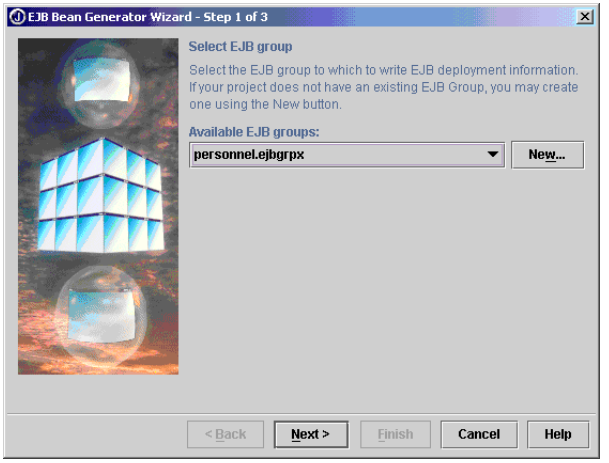


EJB Entity Modeler는 데이터베이스 테이블의 행에 매핑하는 엔티티 bean을 생성하는 데 사용됩니다.



마법사를 사용하여 bean 클래스를 생성한 다음 생성된 코드에 비즈니스 로직을 추가할 수 있습니다. JBuilder의 Bean 디자이너를 사용하여 속성을 추가하고 bean의 원격 인터페이스를 통해 코드에 추가한 비즈니스 메소드를 노출 할 수 있습니다.

EJB Bean Generator는 기존 원격 인터페이스에서 기업용 bean을 생성하는 데 사용됩니다.



JBuilder의 EJB 마법사를 사용해 기업용 bean을 생성하는 것에 대한 자세한 내용은 Part II, "Enterprise JavaBeans Developers Guide" of the *Enterprise Application Developers Guide*의 "기업용 bean 개발"을 참조하십시오.

deployment descriptor 생성

deployment descriptor는 특정 JAR 파일에서 번들되고 배포되는 각 EJB에 대한 정보를 제공합니다. JBuilder의 EJB 마법사를 사용하여 기업용 bean을 생성할 때 deployment descriptor가 자동으로 생성됩니다.

그런 다음 deployment descriptor 에디터를 사용하여 이러한 deployment descriptor를 편집할 수 있습니다. deployment descriptor 에디터의 사용에 대한 내용은 Part II, "Enterprise JavaBeans Developers Guide" of the *Enterprise Application Developers Guide*의 "deployment descriptor 에디터 사용"을 참조하십시오.

EJB 그룹 사용

JBuilder에서 생성한 각 EJB는 단일 JAR 파일에 배포되는 하나 이상의 bean으로 이루어진 논리 그룹인 EJB 그룹에 속합니다. EJB 그룹에는 JAR 파일의 deployment descriptor를 만드는 데 사용되는 모든 정보가 들어 있습니다.

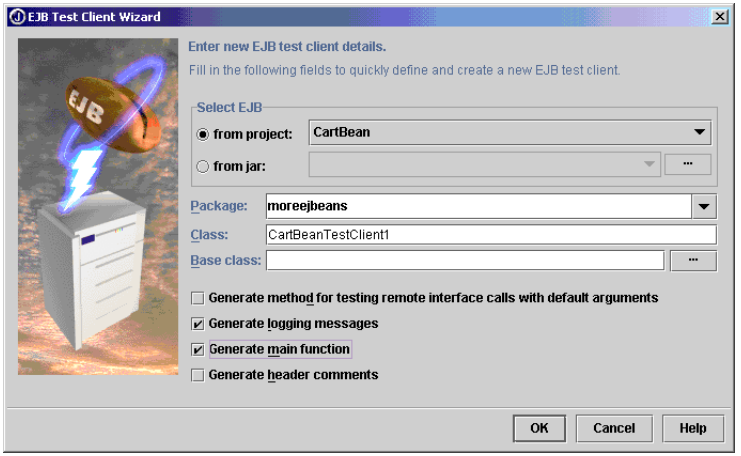
EJB 그룹을 사용하여 한 번에 여러 개의 기업용 bean을 배포할 수 있습니다. 기존 deployment descriptor를 그룹에 추가할 수도 있습니다.

Empty EJB 그룹 마법사 또는 Descriptor 마법사의 EJB 그룹을 사용하여 EJB 그룹을 만듭니다. EJB 그룹 및 EJB 그룹의 생성에 대한 자세한 내용은 Part II, "Enterprise JavaBeans Developers Guide" of the *Enterprise*

*Application Developers Guide*의 "JBuilder를 사용한 기업용 bean 생성"을 참조하십시오.

기업용 bean 테스트

기업용 bean을 생성한 후 EJB Test Client 마법사를 사용하여 bean의 기능을 테스트하는 데 유용한 클라이언트 애플리케이션을 빨리 만들 수 있습니다.



로컬 컨테이너에서 완료된 기업용 bean을 빨리 실행하려면 프로젝트 창에서 해당 EJB 그룹을 마우스 오른쪽 버튼으로 클릭하고 Run을 선택하면 됩니다. 또는 서버 및 컨테이너 런타임 구성을 만듭니다. 두 가지 메소드 중 어느 것을 선택하든 자동으로 로컬 컨테이너가 시작되고 기업용 bean이 실행됩니다.

JBuilder를 사용하여 기업용 bean을 테스트하고 실행하는 것에 대한 자세한 내용은 Part II, "Enterprise JavaBeans Developers Guide" of the *Enterprise Application Developers Guide*의 "기업용 bean 테스트"를 참조하십시오.

기업용 bean 배포

기업용 bean을 배포할 준비가 되면 Tools|EJB Deployment를 선택하여 원하는 애플리케이션 서버에 배포할 수 있습니다.

애플리케이션 서버를 실행한 후 개발 주기 동안 빨리 배포 또는 재배포하거나 배포 해제하기를 원할 경우 JBuilder의 핫 배포 옵션을 사용할 수 있습니다.

기업용 bean 배포에 대한 자세한 내용은 Part II, "Enterprise JavaBeans Developers Guide" of the *Enterprise Application Developers Guide*의 "기업용 bean 배포"를 참조하십시오.

Chapter

10

JBuilder의 XML 지원

XML 지원은 JBuilder 전문가용과 기업용 버전의 기능입니다.

JBuilder는 여러 기능을 제공하며 다양한 툴을 통합하여 Extensible Markup Language(XML)를 지원합니다. XML은 정보를 구성하기 위한 플랫폼 독립적인 메소드입니다. XML은 구조와 문서 내용을 분리하므로 데이터를 교환하는 데 유용하게 사용될 수 있습니다. 예를 들어, XML을 사용하여 데이터베이스와 Java 프로그램 간에 데이터를 전송할 수 있습니다. 또한 내용과 구조가 분리되어 있으므로 스타일 시트를 적용하여 웹 브라우저 등에서 표시할 때 Portable Document Format(PDF)이나 HTML과 같이 다른 형식으로 동일한 내용을 표시할 수 있습니다.

JBuilder의 XML 기능은 다음 범주에 해당합니다.

- XML 문서의 프레젠테이션, 변환, 확인
- XML 문서의 데이터바인딩과 프로그램 처리
- 데이터베이스의 기업용 데이터 인터페이스

JBuilder는 또한 객체 갤러리의 XML 페이지(File|New)에서 사용할 수 있는 여러 XML 마법사를 제공합니다.

자세한 내용은 *JBuilder를 이용한 애플리케이션 구축*의 "Using JBuilder's XML features "를 참조하십시오.

프레젠테이션, 변환, 확인

JBuilder는 XML 문서의 프레젠테이션, 변환, 확인을 수행하기 위해 다음과 같은 툴을 사용합니다.

- 프레젠테이션 레이어로서의 코쿤
- 변환용 Xalan
- XML 문서 확인용 Xerces

데이터 바인딩과 프로그램 처리

이러한 도구들은 XML 문서를 데이터바인딩하고 프로그램을 처리하는 데 사용됩니다.

- DOM과 JAXP의 SAX 마법사 및 라이브러리 정의
- DTD에서 Java 소스를 생성하기 위한 BorlandXML
- 스키마에서 Java 소스를 생성하기 위한 Castor

데이터베이스 컴포넌트와 툴

JBuilder는 데이터베이스 안팎으로 데이터를 전송하기 위해 이러한 컴포넌트와 툴을 사용합니다.

- 모델 기반 데이터베이스 지원용 XML-DBMS
- XML-DBMS 모델 기반 컴포넌트
- 템플릿 기반 컴포넌트

Chapter

11

비즈니스 애플리케이션 개발

JBuilder는 실제적인 비즈니스 애플리케이션을 개발할 수 있는 툴과 기술을 제공합니다. 데이터베이스와 분산 기술을 사용하여 전체 기업 내에서 데이터 및 애플리케이션을 액세스, 수정, 공유할 수 있습니다. JBuilder는 또한 전세계 고객이 쉽게 사용할 수 있으며 국제적인 언어 및 데이터를 지원하는 애플리케이션을 보다 쉽게 개발할 수 있도록 해주는 툴을 제공합니다.

데이터베이스 애플리케이션 구축

이것은 JBuilder 전문가용 및 기업용 버전의 기능입니다.

JBuilder의 DataExpress 컴포넌트를 사용하여 모든 Java 클라이언트 서버 애플리케이션과 인터넷이나 인트라넷용 애플릿을 구축할 수 있습니다. JBuilder로 구축한 애플리케이션은 런타임 시 모든 Java이며 클로스 플랫폼합니다.

JBuilder를 통해 com.borland.dbswing 패키지와 DataExpress Component Library의 com.borland.dx 패키지에서 정의된 속성, 메소드 및 이벤트를 사용하여 데이터를 액세스하고 처리할 수 있습니다. dbSwing 컴포넌트를 사용하면 Swing 컴포넌트의 기능을 확장하고 애플리케이션에 data-aware 기능을 제공할 수 있습니다.

자세한 내용은 Help 메뉴의 *DataExpress Reference* 및 *dbSwing Reference*를 참조하십시오.

JBuilder의 모듈 DataExpress 아키텍처에는 다음의 기능에 대한 지원을 포함하는 여러 가지 장점이 있습니다.

- 네트워크 컴퓨팅
- 모바일 컴퓨팅
- 포함 애플리케이션
- 사용자 인터페이스의 빠른 개발

디자이너를 사용하면 컴포넌트를 컴포넌트 팔레트에서 디자인으로 드래그 앤 드롭하여 데이터베이스 애플리케이션을 빠르게 만들 수 있습니다.

JBuilder 애플리케이션은 JDBC(Java Database Connectivity™) API, JavaSoft 데이터베이스 연결 사양을 통해 데이터베이스 서버와 통신합니다. JDBC는 데이터베이스 데이터에 액세스하고 이를 처리하는 모든 Java 산업 표준 API입니다. JBuilder 데이터베이스 애플리케이션은 JDBC 드라이버를 사용하여 모든 데이터베이스와 연결될 수 있습니다.

JBuilder는 데이터베이스 애플리케이션 개발에 필요한 다음과 같은 툴을 추가로 제공합니다.

- 데이터 캐싱 및 압축 지속성을 위한 JDataStore
  - 트랜잭션 및 작동 중지 복구 지원
  - 향상된 애플리케이션 성능에 대한 고급 동시성 제어
  - JDBC 2.0 Type-4 드라이버(로컬 및 원격)
  - DataStores를 시각적으로 관리하는 JDataStore Explorer
- JDBC 데이터베이스 툴
  - JDBC 데이터 소스에 대한 SQL 쿼리를 시각적으로 만들고 편집하는 SQL Builder
  - SQL 애플리케이션을 모니터하는 JDBC Monitor
  - 데이터 모듈
  - 데이터 모듈 디자이너
  - 데이터 모듈러(Data Modeler)
  - Connection URL Builder
  - 데이터베이스 데이터, 스키마를 보고 URL에 대한 연결을 설정하는 계층 데이터베이스 브라우저

자세한 내용은 *데이터베이스 애플리케이션 개발자 안내서*, Help 메뉴의 *JDataStore Reference*, *JDataStore Developer's Guide*를 참조하십시오.

기술적인 질문이 있을 경우 Borland 웹 사이트 http://www.borland.com/newsgroups/의 데이터베이스 뉴스그룹을 방문하십시오.

분산 애플리케이션 구축

이것은 JBuilder 전문가용 및 기업용 버전의 기능입니다.

JBuilder 개발 환경은 분산 애플리케이션 구축 과정을 매우 쉽게 만들어 다계층, 분산 애플리케이션을 만드는 데 필요한 파일, 구조, 설정 및 경로를 생성합니다. JBuilder는 Java RMI(Remote Method Invocation)나 CORBA(Common Object Request Broker Architecture)를 사용하여 분산 애플리케이션 개발을 완벽하게 지원합니다.

애플리케이션이 생성되면 생성된 코드에 필요한 비즈니스 로직을 추가할 수 있습니다. JBuilder의 개발 환경에서 분산 애플리케이션 개발은 RAD(rapid application development)가 됩니다.

JBuilder는 다음과 같은 기술을 이용합니다.

- RMI(Remote Method Invocation)(JBuilder 전문가용 및 기업용)

  RMI 를 사용하여 분산 Java-to-Java 애플리케이션을 만들 수 있습니다 . 자세한 내용은 Part III, "Distributed Application Developers Guide" in the *Enterprise Application Developers Guide* 의 "Exploring Java RMI-based distributed application in JBuilder" 를 참조하십시오 .

- CORBA(Common Object Request Broker Architecture)(JBuilder 기업용 버전)

  CORBA 는 모든 플랫폼에서 CORBA 가 지원하는 모든 언어로 클라이어트와 서버에서 작성할 수 있는 분산 애플리케이션 개발의 개방형 표준 기반 솔루션입니다 . 자세한 내용은 Part III, "Distributed Application Developers Guide" in the *Enterprise Application Developers Guide* 에서 "Exploring CORBA-based distributed applications in JBuilder" 를 참조하십시오 .

- Java의 CORBA 인터페이스(JBuilder 기업용 버전)

  JBuilder 기업용에 포함된 VisiBroker 는 Java 로 CORBA 인터페이스를 정의할 수 있도록 해주는 두 개의 컴파일러를 통합합니다 . 자세한 내용은 Part III, "Distributed Application Developers Guide" in the *Enterprise Application Developers Guide* 의 "Defining interfaces in Java" 를 참조하십시오 .

JBuilder에는 분산 애플리케이션 디버깅을 지원하는 여러 가지 디버거 기능이 포함됩니다. 특히, 크로스 프로세스 디버깅 및 원격 디버깅 지원이 포함됩니다. 자세한 내용은 Part III, "Distributed Application Developers Guide" in the *Enterprise Application Developers Guide*에서 "Debugging distributed applications"를 참조하십시오.

국제적인 애플리케이션 개발

JBuilder의 특수한 기능을 통해 Java의 국제화 기능을 쉽게 이용할 수 있으므로 코드를 번거롭게 변경하지 않고도 다양한 국가 또는 언어에 따라 애플리케이션을 사용자 지정할 수 있습니다.

이것은 JBuilder 전문가용 및 기업용 버전의 기능입니다.

JBuilder에는 국제적인 시장에 맞게 Java 애플릿 및 애플리케이션을 쉽게 만들 수 있도록 디자인된 다음과 같은 기능이 포함됩니다.

- 다국어 예제 애플리케이션("IntlDemo.jpx" 프로젝트는 JBuilder 설치의 samples/dbSwing/MultiLingual 디렉토리에 있습니다.
- 하드 코딩된 문자열을 제거하는 Resource Strings 마법사
- dbSwing 국제화 아키텍처 및 기능
- UI 디자이너 국제화 지원
- 유니코드에 대한 디버거 완전 지원
- 모든 JDK 원시 코드 변환기용 IDE 및 컴파일러 지원

자세한 내용은 http://java.sun.com/j2se/1.3/docs/guide/intl/index.html의 Java 설명서 및 *Building Applications with JBuilder*의 "Internationalizing programs with JBuilder "를 참조하십시오.

Part

II

Getting Started with Java

Chapter

12

Java 소개

Java는 *객체 지향* 프로그래밍 언어입니다. 다른 프로그래밍 패러다임에서 객체 지향 프로그래밍(OOP)으로의 전환은 어려운 작업입니다. Java는 프로그램에서 액세스하고 처리할 수 있는 객체(데이터 구조 또는 동작)를 만드는 데 포커스를 맞춥니다.

다른 프로그래밍 언어처럼 Java는 다른 입력 및 출력 장치에서 데이터를 읽고 쓰는 기능을 지원합니다. Java는 입/출력 효율을 높이고 국제화를 쉽게 하며 NON-UNIX 플랫폼에 대한 지원을 강화하는 프로세스를 사용합니다. Java는 실행되는 프로그램에 주목하여 더 이상 필요하지 않은 메모리를 해제합니다. 다시 말해서 Java는 메모리 포인터를 추적하고 수동으로 메모리를 해제할 필요가 없습니다. 이 기능을 사용함으로써 프로그램이 작동 중지하는 경우가 없어지며 의도적으로 메모리를 남용할 수도 없게 됩니다.

이 설명서는 다른 언어를 사용하는 프로그래머에게 Java 프로그래밍 언어에 대한 일반적인 내용을 소개하려는 의도에서 만들어졌습니다. 이 설명서는 사용자에게 Java 프로그래밍의 주요 요소에 대해 소개하고 관련 링크 및 읽을 수 있는 자료를 제공하여 사용자가 해당 프로그래밍을 보다 자세히 연구할 수 있게 해줍니다. Fatbrain(http://fatbrain.com/)에서 대부분의 책 제목을 찾아 주문할 수 있습니다. 연결되지 않는 제목을 인쇄할 수 없는 경우도 있으나 일반적으로 유용하게 사용할 수 있습니다. 포괄적인 리소스 목록을 보려면 2- 9페이지 "Java 학습"을 참조하십시오.

이 설명서는 다음과 같은 장으로 구성됩니다.

- Java 구문: Chapter 13, "자바 언어 요소"와 Chapter 14, "자바 언어 구조" Chapter 15, "자바 언어 제어"

  다음의 세 개의 장에서는 Java의 기본 구문을 정의하고 기본적인 객체 지향 프로그래밍의 개념을 소개합니다. 또한 다른 리소스도 제공합니다. 이러한 제안은 시작에 불과합니다. 설명서는 각기 서로 다른 부분을 강조하고 있고 언어에 대해 완벽하게 설명할 수 없기 때문에 설명서를 보며

Java를 학습하려는 사용자에게 서너 개의 설명서가 필요할 수도 있습니다.

각 단원은 "용어" 및 "개념 적용"과 같은 두 가지 주요 부분 으로 구분됩니다. "용어" 부분에서는 이미 이해한 개념에 추가할 새로운 개념을 소개하여 어휘를 늘려줍니다. "개념 적용" 부분에서는 현재까지 제공된 개념의 실제적인 사용에 대해 설명하여 반복적이고 더 복잡한 방법으로 개념을 접할 수 있도록 해줍니다. 계속 복잡해지는 수준으로 인해 어떤 개념은 여러 번 다시 찾아보게 됩니다. 이러한 반복적인 방법을 통해 이해력과 기억력이 향상됩니다.

- Chapter 16, "Java 클래스 라이브러리"

  이 장에서는 Java 2 클래스 라이브러리와 Java 2 플랫폼 에디션에 대한 개요를 제공합니다.

- Chapter 17, " Java의 객체 지향 프로그래밍"

  이 장은 Java의 객체 지향 기능을 소개합니다. 짧은 자습서에서 사용자는 Java 클래스를 만들고 객체를 인스턴스화하며 멤버 변수를 액세스합니다. 사용자는 또한 상속을 사용하여 새 클래스를 만들고, 인터페이스를 사용하여 클래스에 새 기능을 추가하며, 다형성을 사용하여 관련 클래스가 같은 메시지에 여러 가지 방식으로 응답하게 하고, 패키지를 사용하여 관련 클래스를 그룹화하는 것에 대해 학습합니다.

- Chapter 18, "스레드 기법"

  스레드는 프로그램 안에서 하나의 순차적인 제어 흐름입니다. 자바 언어의 가장 강력한 측면 중 하나는 실행의 여러 스레드를 프로그램화하여 같은 프로그램 내에서 동시에 실행할 수 있다는 점입니다. 이 장에서는 다중 스레드 프로그램을 만드는 방법에 대해 설명하고 깊이 있는 정보를 볼 수 있는 다른 리소스에 대한 링크를 제공합니다.

- Chapter 19, "직렬화"

  일련화는 Java 객체 상태를 저장하고 복원합니다. 이 장에서는 Java를 사용하여 객체를 일련화하는 방법에 대해 설명하고 또한 Serializable 인터페이스와 객체를 디스크에 쓰는 방법 및 객체를 메모리로 다시 읽어 오는 방법에 대해 설명합니다.

- Chapter 20, "Java Virtual Machine 소개"

  JVM은 특정 기계에서 Java 프로그램을 실행할 수 있도록 해주는 기본 소프트웨어입니다. 이 장에서는 JVM의 일반 구조 및 용도에 대해 설명하며 JVM의 주요 역할, 특히 Java 보안상 중요한 역할에 대해서도 설명합니다. 또한 세 가지 구체적인 보안 기능인 Java 인증자, Security Manager 및 Class Loader을 보다 자세히 다룹니다.

- Chapter 21, "Java Native Interface(JNI)의 사용"

  이 장에서는 Java Native Method Interface(JNI)를 사용하여 Java 애플리케이션에서 원시 메소드를 호출하는 방법에 대해 설명합니다. 먼저 JNI가 작동하는 방법에 대해 설명한 다음 native 키워드 및 Java 메소드가 native 메소드가 될 수 있는 방법에 대해 설명합니다.마지막으로 Java 클

래스용 C 헤더 파일을 생성하는 데 사용되는 JDK의 javah 툴을 검사합니다.

- Chapter 22, "자바 언어 빠른 참조"

  이 장에는 클래스 라이브러리의 일부 목록과 주요 해당 기능, Java2 플랫폼 에디션 목록, JDK 1.3에 대한 Java 키워드 전체 목록, 기본(primitive) 타입 및 참조 타입 사이의 데이터 타입 변환에 대한 포괄적인 표, Java 이스케이프 시퀀스, 연산자와 그 동작에 대한 포괄적인 표 등이 포함되어 있습니다.

Chapter

13

자바 언어 요소

이 단원에서는 이 장 전체에서 사용될 Java 프로그래밍 언어의 요소에 대한 기초적인 개념을 제공합니다.여기서는 사용자가 일반적인 프로그래밍 개념은 이해하지만 Java에 대한 경험이 거의 없다고 가정합니다.

용어

다음과 같은 용어 및 개념을 이 장에서 설명합니다.

- 식별자
- 데이터 타입
- 문자열
- 배열
- 변수
- 리터럴

식별자

식별자는 변수나 메소드 같은 요소를 호출할 때 선택하는 이름입니다. Java는 모든 유효한 식별자를 허용하지만 사용성의 측면에서 다음과 같은 요구 사항에 맞도록 수정된 일반 언어로 된 용어를 사용하는 것이 가장 좋습니다.

- 문자로 시작해야 합니다. 엄밀히 말해 유니코드 통화 기호 또는 밑줄(_)로 시작할 수 있긴 하지만 이러한 기호 중 일부가 import된 파일이나 내부 처리에 사용될 수도 있습니다. 따라서 사용하지 않는 것이 좋습니다.
- 그 다음에 모든 영숫자 문자(문자 또는 숫자), 밑줄 또는 유니코드 통화 기호(예: □또는 $ 등)를 사용할 수 있지만 다른 특수 문자는 사용할 수 없습니다.

- 공백이나 하이픈 없이 전체적인 하나의 단어를 사용해야 합니다.

식별자의 대문자 사용은 식별자의 종류에 따라 달라집니다. Java는 대/소문자를 구분하므로 대문자 사용에 주의해야 합니다. 올바른 대문자 스타일은 컨텍스트에서 언급됩니다.

데이터 타입

데이터 타입은 특정 Java 프로그래밍 요소가 포함할 수 있는 정보의 종류를 분류합니다. 데이터 타입은 다음과 같은 두 가지 주요 범주로 구분됩니다.

- 기본(primitive) 또는 기본(basic)
- 복합 또는 참조

물론 데이터 타입의 종류에 따라 다른 종류의 정보를 보유하며 그 양도 다릅니다. 변수의 데이터 타입을 다음과 같은 한계 내에서 다른 타입으로 변환할 수 있습니다. boolean 타입은 변환할 수 없고 관련되지 않은 클래스의 객체로 객체를 변환할 수 없습니다.

Java는 데이터를 위험하지 않게 합니다. 즉, 변수나 객체를 더 큰 타입으로 쉽게 변환할 수 있지만 더 작은 타입으로 변환할 수는 없다는 의미입니다. 더 큰 용량의 데이터 타입을 더 작은 용량으로 변경할 때는 *type cast* 라는 문장 타입을 사용해야 합니다.

기본(primitive) 데이터 타입

기본(primitive) 또는 기본(basic) 데이터 타입은 Boolean(on/off 상태 지정), character(단일 문자와 유니코드 문자), integer(정수) 또는 floating-point(실수)로 분류됩니다. 코드에서 기본(primitive) 데이터 타입은 모두 소문자입니다.

Boolean 데이터 타입은 boolean이라고 하며, true 또는 false의 두 값 중 하나를 취합니다. Java에서는 이러한 값을 수로 저장하지 않고 boolean 데이터 타입을 사용하여 이러한 값을 저장합니다.

character 데이터 타입은 char이라고 하며 최대 16비트 길이로 단일 유니코드 문자를 취합니다. Java에서 유니코드 문자(문자, 특수 문자 및 문장 부호)는 'b'처럼 작은따옴표 사이에 옵니다. Java의 유니코드 기본값은 ₩u0000이며 범위는 ₩u0000에서 ₩uFFFF 사이입니다.

유니코드 번호 지정 시스템은 0에서 65535 사이의 숫자를 취하지만 그 숫자는 이스케이프 시퀀스 ₩u가 그 앞에 나오는 16진수 표시법으로 지정되어야 합니다.

모든 특수 문자를 이러한 방식으로 나타낼 수 있는 것은 아닙니다. Java는 고유한 이스케이프 시퀀스 집합을 제공하는데, 22- 19페이지의 "이스케이프 시퀀스" 표에서 볼 수 있습니다.

Java에서 기본(primitive) 데이터 타입의 크기는 플랫폼에 종속적이지 않고 절대적입니다. 따라서 이식성이 향상됩니다.

숫자 데이터 타입에 따라 다른 종류와 크기의 숫자를 취합니다. 해당 이름 및 크기가 다음 표에 나열되어 있습니다.

타입	속성	범위
double	Java의 기본값. 대략 소수점 15자리까지의 8바이트 숫자를 취하는 부동 소수점 타입	+/- $9.00x10^{18}$
int	가장 일반적인 옵션.전체 4바이트를 취하는 정수 타입	+/- $2x10^{9}$
long	전체 8바이트를 취하는 정수 타입	+/- $9x10^{18}$
float	대략 소수점 7자리까지의 4바이트 숫자를 취하는 부동 소수점 타입	+/- $2.0x10^{9}$
short	전체 2바이트를 취하는 정수 타입	+/- 32768
byte	전체 1바이트를 취하는 정수 타입	+/- 128

복합 데이터 타입

위의 각 데이터 타입은 하나의 숫자, 하나의 문자 또는 하나의 상태를 받아들입니다. 복합 또는 참조 데이터 타입은 둘 이상의 요소로 구성됩니다. 복합 데이터 타입에는 클래스와 배열이라는 두 종류가 있습니다. 클래스와 배열 이름은 대문자로 시작되고 낙타 등처럼 대문자를 사용합니다. 즉, 이름에서 각 자연어의 첫 글자는 대문자로 시작됩니다. 예를 들어, NameOfClass와 같습니다.

클래스는 논리적으로 통합된 객체 및 그 동작 집합을 정의하는 완전한 코드입니다.클래스에 대한 자세한 내용은 Chapter 17, " Java의 객체 지향 프로그래밍"을 참조하십시오.

클래스를 만들고 프로그램으로 import하면 데이터 타입으로서 사용할 수 있습니다. String 클래스는 데이터 타입으로 가장 자주 사용되는 클래스이므로 이 장에서는 이 클래스에 포커스를 맞춥니다.

문자열

String 데이터 타입은 사실상 String 클래스입니다. String 클래스는 큰따옴표로 둘러싸인 영숫자 문자, 공백 및 일반 문장 부호(*문자열*)의 시퀀스를 저장합니다. 문자열에 모든 유니코드 이스케이프 시퀀스를 포함할 수 있고 문자열 안에 큰따옴표를 놓으려면 ₩"가 필요할 수 있지만 일반적으로 String 클래스 자체에서 문자를 올바르게 해석하는 방법을 프로그램에 알려 줍니다.

배열

배열은 동일한 데이터 타입을 가지는 값의 그룹을 포함하는 데이터 구조입니다. 예를 들어, 배열에 String 값의 그룹, int 값의 그룹 또는 boolean 값의 그룹을 사용할 수 있습니다. 동일한 데이터 타입의 모든 값은 모두 동일한 배열이 될 수 있습니다.

배열은 한 쌍의 대괄호로 나타냅니다. Java에서 배열을 선언하는 경우 식별자 다음이나 데이터 타입 다음에 대괄호를 넣을 수 있습니다.

```
int studentID[];
char[] grades;
```

배열의 크기가 지정되지 않습니다. 배열을 선언해도 해당 배열에 대한 메모리가 할당되지는 않습니다. 대부분의 다른 언어에서는 배열의 크기가 선언에 포함되어야 하지만 Java에서는 배열을 사용할 때까지 크기를 지정하지 않습니다. 그러면 적절한 메모리가 할당됩니다.

변수

변수는 프로그래머가 이름을 지정하고 정의하는 값입니다. 변수에는 식별자와 값이 필요합니다.

리터럴

리터럴은 숫자, 문자, 상태 또는 문자열의 실제 표현입니다. 리터럴은 식별자의 값을 나타냅니다.

영숫자 리터럴에는 큰따옴표 안의 문자열, 작은따옴표 안의 단일 char 문자, boolean true/false 값이 포함됩니다.

정수 리터럴은 10진수, 8진수 또는 16진수로 저장될 수 있지만 구문을 고려해야 합니다. 날짜에서처럼 0으로 시작되는 모든 정수는 8진수로 해석됩니다. 부동 소수점 리터럴은 10진수로만 표시될 수 있습니다. 타입을 지정하지 않으면 double로 처리됩니다.

리터럴 및 그 용량에 대한 자세한 내용은 Patrick Naughton의 *The Java Handbook*을 참조하십시오.

개념 적용

다음 단원에서는 이 장의 초반부에 소개된 용어 및 개념을 적용하는 방법에 대해 설명합니다.

변수 선언

변수 선언 작업은 선언한 변수에 대한 메모리를 할당합니다. 변수를 선언할 때는 데이터 타입과 식별자의 두 가지만 순서대로 필요합니다. 데이터 타입은 할당할 메모리 양을 프로그램에 알려 주고, 식별자는 할당된 메모리에 레이블을 지정합니다.

변수는 한 번만 선언합니다. 적절하게 변수를 선언했으면 메모리의 해당 블록에 액세스하기 위해서는 식별자만 참조하면 됩니다.

변수 선언은 다음과 같습니다.

boolean isOn;	데이터 타입 boolean은 true 또는 false로 설정될 수 있습니다. 식별자 isOn은 프로그래머가 이 변수에 대해 할당된 메모리에 지정한 이름입니다. isOn이라는 이름은 논리적으로 true/false 값을 적용하는 것으로서 코드를 읽어 보는 사용자를 위한 의미가 있습니다.
int studentsEnrolled;	int 데이터 타입은 10자리보다 적은 정수를 처리하는 것을 알려 줍니다. 식별자 studentsEnrolled는 나타낼 숫자를 의미합니다. 학생들의 수는 정수이므로 해당 데이터 타입은 정수를 요구합니다.
float creditCardSales;	일반적으로 화폐는 소수점 두 자리로 표시되므로 float 데이터 타입이 적합합니다. 프로그래머가 변수 이름을 creditCardSales로 유용하게 지정했기 때문에 화폐와 관련있다는 것을 알 수 있습니다.

메소드

Java의 *메소드*는 다른 언어의 함수 또는 서브루틴과 동일합니다. 메소드는 객체에서 수행할 작업을 정의합니다.

메소드는 다음과 같이 이름과 한 쌍의 괄호로 구성됩니다.

```
getData()
```

여기서 getData는 이름이고 괄호는 메소드임을 프로그램에 알려 줍니다.

메소드가 작업을 수행하는 데 있어 특정 정보가 필요하면 그것을 괄호 안에 넣습니다. 괄호 안에 들어가는 것을 *인수* 또는 짧게 *arg*라고 합니다. 메소드 선언에서 인수는 데이터 타입과 식별자를 포함해야 합니다.

```
drawString(String remark)
```

여기서 drawString은 메소드의 이름이고, String remark는 메소드를 수행하는 데 필요한 문자열의 데이터 타입과 변수 이름입니다.

메소드가 반환할 데이터 타입 또는 반환할 것을 프로그램에 알려 주어야 하는데 이를 *반환 타입*이라고 합니다. 기본(primitive) 타입의 데이터를 반환하는 메소드를 만들 수 있습니다. 작업을 수행하는 대부분의 메소드처럼 메소드가 아무것도 반환하지 않으면 반환 타입은 void가 되어야 합니다.

반환 타입, 이름 및 필요한 인수를 갖는 괄호를 가지고 다음과 같은 매우 기본적인 메소드 선언을 제공합니다.

```
String drawString(String remark);
```

사용자의 메소드는 이것보다 더 복잡할 수도 있습니다. 메소드를 입력하고 이름을 지정한 다음 필요한 경우 인수를 지정했으면 메소드를 완전하게 정의해야 합니다. 메소드 이름 아래에 작업을 정의하는 몸체 부분을 한 쌍의 중괄호로 묶어 표시합니다. 다음은 좀 더 복잡한 메소드 선언입니다.

```
String drawString(String remark) {              //Declares the method
  String remark = "My, what big teeth you have!" //Defines what's in the
method.
}                                               //Closes the method body.
```

메소드를 정의했으면 이름으로 해당 메소드를 참조하고 작업을 수행하는 데 필요한 인수를 전달하기만 하면 됩니다. drawString(remark);

Chapter

14

자바 언어 구조

이 단원에서는 이 장 전체에서 사용될 Java 프로그래밍 언어의 구조에 대한 기초적인 개념을 제공합니다.여기서는 사용자가 일반적인 프로그래밍 개념은 이해하지만 Java에 대한 경험이 거의 없다고 가정합니다.

용어

다음과 같은 용어와 개념들이 이 장에 설명되어 있습니다.

- 키워드
- 연산자
- 주석
- 문장
- 코드 블록
- 범위

키워드

키워드는 다른 구문 요소를 수정하는 예약된 Java용어입니다. 키워드는 객체의 액세스 가능성, 메소드의 흐름 또는 변수의 데이터 타입을 정의할 수 있습니다. 키워드는 식별자로 사용할 수 없습니다.

Java의 많은 키워드는 C/C++에서 가져옵니다. 또한 C/C++에서처럼 키워드는 항상 소문자로 작성됩니다. 일반적으로 Java의 키워드는 그 기능에 따라 다음과 같이 분류될 수 있습니다. 괄호 안은 예제입니다.

- 데이터 및 반환 타입과 용어(int, void, return)
- 패키지, 클래스, 멤버 및 인터페이스(package, class, static)
- 액세스 변경자(public, private, protected)
- 루프 및 루프 제어(if, switch, break)
- 예외 처리(throw, try, finally)

- 아직 사용되지 않았지만 사용할 수 없는 예약어(goto, assert, const)

일부 키워드는 이 장의 컨텍스트에 따라 설명됩니다. 전체 키워드 목록 및 그 의미에 대해서는 22- 2페이지의 시작 부분에 있는 "키워드" 표를 참조하십시오.

연산자

연산자를 사용하면 변수에서 클래스에 이르는 Java 언어 요소를 액세스하거나 처리하거나 관련시키거나 참조할 수 있습니다. 연산자에는 *우선 순위*와 *연관성*이라는 속성이 있습니다. 여러 연산자가 같은 요소 또는 *피연산자*에서 동작하면 연산자의 우선 순위에 의해 먼저 동작할 연산자가 결정됩니다. 둘 이상 연산자의 우선 순위가 같으면 연관성 규칙이 적용됩니다. 일반적으로 이 규칙은 수학적입니다. 예를 들어, 연산자는 왼쪽에서 오른쪽으로 사용되고 괄호 안의 연산자 표현식은 괄호 밖의 연산자 표현식보다 먼저 계산됩니다.

일반적으로 연산자는 다음과 같이 여섯 개의 범주, 할당, 산술, 논리, 비교, 비트 단위 및 3항으로 구분됩니다.

*할당*은 변수 내 =의 *오른쪽*에 있는 값을 *왼쪽*에 저장한다는 의미입니다. 변수를 선언할 *때* 나 변수를 선언한 *이후* 변수에 값을 할당할 수 있습니다. 기계는 프로그램 및 연습에 적합한 것으로서 사용자가 어떠한 방법을 선택할 것인지에 대해서는 관여하지 않습니다.

```
double bankBalance;          //Declaration
bankBalance = 100.35;        //Assigment

double bankBalance = 100.35;  //Declaration with assignment
```

두 가지 경우 모두 100.35라는 값이 bankBalance 변수의 선언으로 예약된 메모리 안에 저장됩니다.

할당 연산자를 통해 변수에 값을 할당할 수 있습니다. 또한 할당 연산자를 사용하면 결합된 표현식 하나를 사용하여 표현식에서 연산을 수행한 다음 오른쪽 피연산자에 새 값을 할당할 수 있습니다.

산술 연산자는 정수 및 부동 소수점 값 모두에 대해서 수학적 계산을 수행합니다. 일반 수학적 기호로 두 수에 대한 + 더하기, - 빼기, * 곱하기 및 / 나누기를 사용합니다.

논리 또는 부울 연산자를 통해 프로그래머는 유용한 방법으로 부울 표현식을 그룹화하여 특정 조건을 판단하는 방법을 정확하게 프로그램에 알려 줍니다.

비교 연산자는 코드의 다른 부분과 비교하여 단일 표현식을 계산합니다. 문자열 비교처럼 더 복잡한 비교는 프로그램에서 수행됩니다.

비트 단위 연산자는 이진수의 0과 1을 바탕으로 작동합니다. Java의 비트 단위 연산자는 원래 숫자의 부호를 유지할 수 있지만 모든 언어에서 부호가 유지되지는 않습니다.

3항 연산자 ?:은 매우 간단한 if-then-else 문을 작성하는 방법을 제공합니다. 첫 번째 표현식이 계산되고, 그 식이 true이면 두 번째 식이 계산되고, 두 번째 식이 false이면 세 번째 식이 사용됩니다.

다음은 다른 연산자의 일부 목록과 그 속성입니다.

연산자	피연산자	동작
.	객체 멤버	객체의 멤버에 액세스합니다.
(<type>)	데이터 타입	데이터 타입을 변환합니다.[1]
+	문자열	문자열을 결합합니다(연결자).
	숫자	더합니다.
–	숫자	이것은 숫자의 부호를 바꾸는 단항 연산자[2] – (minus)입니다.
	숫자	빼기.
!	부울	이것은 부울 NOT 연산자입니다.
&	정수, 부울	이것은 비트 단위(정수)이자 부울 AND 연산자입니다. 두 번 사용 될 경우(&&) 이것은 부울 조건부 AND입니다.
=	변수를 가진 요소 대부분	다른 요소에 요소를 할당합니다(예를 들어, 변수에 값 또는 인스턴스에 클래스를 지정)이것은 다른 연산자와 결합하여 다른 연산을 수행하고 결과 값을 할당할 수 있습니다.예를 들어, +=은 왼쪽값을 오른쪽에 더하여 그 값을 표현식의 오른쪽에 할당합니다.

1. 연산과 문장 부호를 구분하는 것이 중요합니다. 괄호는 인수 주위에서 문장 부호로 사용됩니다. 괄호는 괄호 내의 데이터 타입으로 변수의 데이터 타입을 변경하는 연산에서 데이터 타입 주위에 사용됩니다.
2. 단항 연산자에는 피연산자가 한 개 있고, 이항 연산자에는 두 개 있으며, 3항 연산자에는 세 개 있습니다.

주석

코드에 주석 처리하는 것은 아주 좋은 프로그래밍 방법입니다. 주석 처리를 잘 하면 코드를 더 빨리 검색하고, 복잡한 프로그램을 구축할 때 수행한 작업을 추적하고, 추가하거나 조정하려는 사항을 상기시킬 수 있습니다. 주석을 사용하여 특수한 상황에 대비해 저장하거나 충돌할 수 있는 작업을 수행하는 동안 충돌을 막기 위해 코드의 일부를 숨길 수 있습니다. 주석은 사용자가 다른 프로젝트에서 작업한 후 이전 프로젝트로 돌아오거나 휴가에서 돌아온 경우 수행 중이던 작업을 기억할 수 있도록 도와 줄 수 있습니다. 팀 개발 환경이나 프로그래머 사이에서 코드가 전달될 때마다 주석은 모든 부분을 구문 분석하여 이해하지 않고도 사용자가 주석 처리한 부분의 용도 및 연관을 다른 사람이 이해할 수 있도록 도와 줍니다.

Java는 세 종류의 주석, 한 줄 주석, 여러 줄 주석 및 Javadoc 주석을 사용합니다.

설명	Tag	용도
한 줄	// ...	문장이나 표현식의 기능이나 구조에 대한 간단한 설명에 적합합니다. 여는 태그만 필요하며, 새 줄을 시작하는 즉시 코드로 돌아갑니다.
여러 줄	/* ... */	코드의 내용에 대한 자세한 정보로 이동하거나 코드에 사용권에 대한 고지 사항을 포함하려는 경우처럼 두 줄 이상이 필요한 주석에 적합합니다.여는 태그와 닫는 태그가 모두 필요합니다.
Javadoc	/** ... */	JDK의 Javadoc 유틸리티에서 읽어 HTML 설명서로 변환할 수 있는 여러 줄 주석입니다. Javadoc에는 그 기능을 확장하는 데 사용할 수 있는 태그가 있습니다. API에 도움말을 제공하고, 할 일 목록을 생성하며, 코드에 플래그를 포함하는 데 사용됩니다.여는 태그와 닫는 태그가 모두 필요합니다. Javadoc 툴에 대해 자세히 학습하려면 Sun의 Javadoc 페이지 http://java.sun.com/products//jdk/1.2/docs/tooldocs/javadoc/로 이동하십시오.

몇 가지 예제를 보면 다음과 같습니다.

```
/* You can put as many lines of
   discussion or as many pages of
   boilerplate as you like between
   these two tags.
*/

/* Note that, if you really get carried away,
   you can nest single-line comments
   //inside of the multi-line comments
   and the compiler will have no trouble
   with it at all.
*/

/* Just don't try nesting
   /* multi-line types of comments
   */
   /** of any sort
   */
   because that will generate a
   compiler error.
*/

/**Useful information about what the code
   does goes in Javadoc tags. Special tags
   such as @todo can be used here to take
   advantage of Javadoc's helpful features.
*/
```

문장

문장은 단일 명령입니다. 한 명령이 여러 불의 코드에 사용될 수 있지만 컴파일러는 전체를 하나의 명령으로 읽습니다. 개별(한 줄) 문장은 세미콜론(;)으로 끝나고 그룹(여러 줄) 문장은 닫는 중괄호(}로 끝납니다. 일반적으로 여러 줄 문장을 *코드 블록*이라고 합니다.

기본적으로 Java는 문장이 작성된 순서대로 문장을 실행하지만 Java를 사용하여 아직 정의되지 않은 용어로 참조를 전달할 수 있습니다.

코드 블록

코드 블록은 중괄호 사이에 있는 모든 내용이며 중괄호 부분을 시작하는 표현식을 포함합니다.

```
class GettingRounder {
  ...
}
```

범위 이해

범위 규칙은 프로그램에서 변수가 인식되는 위치를 결정합니다. 변수는 다음과 같은 두 가지 주요 범위로 분류됩니다.

- 전역 변수: 전체 클래스 사이에서 인식되는 변수입니다.
- 지역 변수: 변수가 선언된 코드 블록에서만 인식되는 변수입니다.

범위 규칙은 코드 블록과 밀접한 관련이 있습니다. 한 가지 일반적인 범위 규칙은 다음과 같습니다. 코드 블록에서 선언된 변수는 해당 블록과 그 안에 중첩된 모든 블록에만 표시됩니다. 다음 코드는 이러한 규칙을 설명합니다.

```
class Scoping {
int x = 0;
  void method1() {
    int y;
    y = x;  // This works. method1 can access y.
  }
  void method2() {
int z = 1;
    z = y;  // This does not work:
            // y is defined outside method2's scope.
  }
}
```

이 코드는 scoping이라는 클래스를 선언하며 이 클래스에는 두 가지 메소드, method1()과 method2()가 있습니다. 클래스 자체는 메인 코드 블록으로 간주되며 두 가지 메소드는 클래스의 중첩 블록입니다.

x 변수는 메인 블록에서 선언되므로 method1()과 method2() 모두에서 *표시*됩니다. 즉, 컴파일러에서 인식됩니다. 그러나 변수 y와 z는 두 가지 독

립된 중첩 블록에서 선언되므로 method2()에서 y를 사용하려고 하면 y는 해당 블록에서 표시되지 않기 때문에 오류가 발생합니다.

참고 다음과 같은 두 가지 이유로 전역 변수에 의존하는 프로그램에서 오류가 발생하기도 합니다.

1. 전역 변수는 추적하기 어렵습니다.
2. 프로그램의 한 부분에서 전역 변수를 변경하면 프로그램의 다른 부분에서 예기치 않은 부작용이 발생할 수 있습니다.

지역 변수는 수명이 제한되어 있기 때문에 사용하기가 보더 안전합니다. 예를 들어, 메소드 내에서 선언된 변수는 해당 메소드에서만 액세스할 수 있으므로 프로그램의 다른 부분에서 잘못 사용할 위험이 없습니다.

간단한 모든 문장은 세미콜론으로 끝납니다. 모든 중괄호에는 짝이 있어야 합니다. 위의 예제에서처럼 중괄호를 일관된 방법으로 구성하여 그 쌍을 추적할 수 있습니다. JBuilder 같은 많은 Java IDE는 설정에 따라 자동으로 중괄호를 중첩합니다.

개념 적용

다음 단원에서는 이 장의 초반부에 소개된 용어 및 개념을 적용하는 방법에 대해 설명합니다.

연산자 사용

리뷰 여섯 가지 기본 연산자(산술, 논리, 할당, 비교, 비트 단위 및 3항)가 있으며 연산자는 하나, 둘 또는 세 개의 피연산자에 영향을 주어 단항, 이항 또는 3항 연산자를 만듭니다. 우선 순위와 연관성이라는 속성이 있으며, 이러한 속성은 연산자가 처리되는 순서를 결정합니다.

연산자는 우선 순위를 설정하는 할당된 숫자입니다. 숫자가 클수록 우선 순위의 순서가 높습니다. 즉, 다른 연산자보다 더 빨리 계산될 수 있습니다. 우선 순위가 1(가장 낮음)인 연산자가 마지막으로 계산되고 우선 순위가 15(가장 높음)인 연산자가 처음으로 계산됩니다.

일반적으로 우선 순위가 같은 연산자일 경우에는 왼쪽에서 오른쪽으로 계산됩니다.

우선 순위는 연관성보다 먼저 계산됩니다. 예를 들어, 표현식 a + b - c * d는 왼쪽에서 오른쪽으로 계산되는데 곱하기는 더하기보다 우선 순위가 높으므로 c * d가 먼저 계산됩니다. 더하기와 빼기는 우선 순위가 같으므로 연관성이 적용되어 a와 b가 먼저 더해진 다음 c * d의 값에서 그 합을 뺍니다.

우선 순위에 관계없이 먼저 계산할 수학적 표현식 주위에는 괄호를 사용하는 것이 좋습니다. 예: (a + b) - (c * d) 프로그램에서는 같은 방식으로 이 연산을 계산하지만 사용자의 경우에는 이 형식이 더 명확합니다.

산술 연산자

Java는 수학적 계산에 필요한 모든 연산자 집합을 제공합니다. 다른 언어와 달리 Java는 정수와 부동 소수점 값 모두에서 수학적 기능을 수행할 수 있습니다. 따라서 사용자는 이러한 연산자에 친숙해집니다.

산술 연산자는 다음과 같습니다.

연산자	정의	Prec.	Assoc.
++/--	자동 증가/감소:단일 피연산자에 하나를 더하거나 하나를 뺍니다. i의 값이 4이면 ++i는 5입니다. pre-increment(++i)는 값을 하나씩 증가시키고 원래 변수 i에 새 값을 할당합니다. post-increment(i++)는 값을 증가시키지만 원래 변수 i는 원래 값으로 그대로 유지합니다. 자세한 내용을 아래를 참조하십시오.	1	오른쪽
+/-	단항 +/-:단일 숫자의 양/음 값을 설정하거나 변경합니다.	2	오른쪽
*	곱하기.	4	왼쪽
/	나누기.	4	왼쪽
%	계수: 첫째 피연산자를 둘째 피연산자로 나누고 나머지를 반환합니다. 간단한 수학적 리뷰에 대해서는 아래를 참조하십시오.	4	왼쪽
+/-	더하기/빼기	5	왼쪽

새 값을 할당하려는 경우에 따라 pre- 또는 post-increment/decrement를 사용합니다.

```
int y = 3, x; //1. variable declarations
int b = 9;    //2.
int a;        //3.
x = ++y;      //4. pre-increment
a = b--;      //5. post-decrement
```

문장 4에서 pre-increment, y 변수의 값은 1씩 증가된 다음 그 새 값(4)은 x에 할당됩니다. 원래 x와 y의 값은 모두 3이고 이제 둘 다 4가 되었습니다.

문장 5에서 post-decrement, b의 현재 값(9)은 a에 할당되고 *그런 다음* b의 값은 8로 감소됩니다. 원래 b의 값은 9이고 a에는 할당된 값이 없었는데, 현재 a는 9이고 b는 8입니다.

계수 연산자에는 오래 전에 수학을 마지막으로 연구한 사람에 대한 설명이 필요합니다. 두 숫자를 나눌 때 똑같이 나누어지는 경우가 거의 없습니다. 소수점을 새로 추가하지 않고 숫자를 나눈 후 남는 부분을 *나머지*라고 합니다. 예를 들어, 5를 3으로 한 번 나누면 2가 남습니다. 이런 경우 나머지 2가 계수 연산자가 계산하는 부분입니다. 예를 들어, 한 시간의 계수가 60인 것처럼 예상 가능한 방식으로 나머지는 나누기 과정에서 반복하여 발생하므로 계수 연산자는 특정 간격으로 프로세스를 반복하도록 프로그램에 알려 주려는 경우 특히 유용합니다.

논리 연산자

논리 또는 부울 연산자를 사용하면 프로그래머는 부울 표현식을 그룹화하여 특정 조건을 판단할 수 있습니다. 이러한 연산자는 표준 부울 연산 AND, OR, NOT과 XOR을 수행합니다.

다음 표는 논리 연산자를 열거한 것입니다.

연산자	정의	Prec.	Assoc.
!	부울 NOT(단항) true를 false로 변경하거나 false를 true로 변경합니다.낮은 우선 순위 때문에 이 문장 주위에 괄호를 사용해야 합니다.	2	오른쪽
&	AND 연산(이항) 두 피연산자가 모두 true인 경우에만 true가 됩니다.항상 두 피연산자 모두 계산합니다. 논리 연산자로 사용되는 경우는 거의 없습니다.	9	왼쪽
^	XOR 연산(이항) 단 하나의 피연산자가 true인 경우에 true가 됩니다.두 피연산자 모두 계산합니다.	10	왼쪽
|	OR 연산(이항) 하나 또는 두 개의 피연산자가 모두 true인 경우에 true가 됩니다.두 피연산자 모두 계산합니다.	11	왼쪽
&&	조건부 AND(이항) 두 피연산자가 모두 true인 경우에만 true가 됩니다.첫 번째 피연산자가 true인 경우에 두 번째 연산자만 계산하기 때문에 "조건부"라고 합니다.	12	왼쪽
||	조건부 OR(이항) 하나 또는 두 개의 피연산자가 모두 true인 경우에 true가 되고 두 피연산자 모두 false인 경우에는 false가 됩니다.첫 번째 피연산자가 true인 경우에 두 번째 피연산자를 계산하지 않습니다.	13	왼쪽

계산 연산자는 항상 피연산자 두 개를 모두 계산합니다. 그러나 조건부 연산자는 항상 첫 번째 피연산자를 계산하고 그 결과로 전체 표현식의 값이 결정되면 두 번째 피연산자를 계산하지 않습니다.예를 들면, 다음과 같습니다.

```
if ( !isHighPressure && (temperature1 > temperature2)) {
  ...
}                                    //Statement 1: 조건부

boolean1 = (x < y) || ( a > b);          //Statement 2: 조건부

boolean2 = (10 > 5) & (5 > 1);           //Statement 3: evaluation
```

첫 번째 문장에서 !isHighPressure를 먼저 계산합니다. !isHighPressure이 false인 경우(즉, 압력*이 높은* 경우 !의 논리적 이중 부정 및 false}, 두 번째 피연산자, temperature1 > temperature2는 계산할 필요가 없습니다. &&는 반환할 값을 인지하기 위해 단 하나의 false 값을 필요로 합니다.

두 번째 문장에서 x가 y보다 작으면 boolean1 값은 true가 됩니다. x가 y보다 크면 두 번째 표현식이 계산되고 a가 b보다 작으면 boolean1 값은 여전히 true입니다.

그러나 세 번째 문장에서 &는 조건부 연산자가 아닌 계산 연산자이기 때문에 컴파일러는 boolean2에 true 또는 false를 할당하기 전에 두 가지 피연산자의 값을 계산합니다.

할당 연산자

기본 할당 연산자(=)를 사용하면 변수에 값을 할당할 수 있습니다. Java의 할당 연산자 집합을 사용하여 어떠한 피연산자라도 연산을 수행할 수 있으며 새 값을 1단계의 변수에 할당할 수 있습니다.

다음 표는 할당 연산자를 나열한 것입니다.

연산자	정의	Prec.	Assoc.
=	오른쪽 값을 왼쪽 변수에 할당합니다.	15	오른쪽
+=	오른쪽 값을 왼쪽 변수 값에 더하고 그 값을 원래 변수에 할당합니다.	15	오른쪽
−=	왼쪽 변수 값에서 오른쪽 값을 빼고 그 값을 원래 변수에 할당합니다.	15	오른쪽
*=	오른쪽 값을 왼쪽 변수 값에 곱하고 그 값을 원래 변수에 할당합니다.	15	오른쪽
/=	왼쪽 변수 값을 오른쪽 값으로 나누고 그 값을 원래 변수에 할당합니다.	15	오른쪽

첫 번째 연산자는 지금까지 친숙한 연산자입니다. 나머지 할당 연산자가 먼저 연산을 수행한 다음 연산의 결과를 표현식의 왼쪽에 있는 피연산자에 저장합니다.몇 가지 예제를 보면 다음과 같습니다.

```
int y = 2;
y *= 2;  //same as (y = y * 2)

boolean b1 = true, b2 = false;
b1 &= b2;  //same as (b1 = b1 & b2)
```

비교 연산자

비교 연산자를 사용하여 한 값을 다른 값과 비교할 수 있습니다.

다음 표는 비교 연산자를 나열한 것입니다.

연산자	정의	Prec.	Assoc.
<	작다	7	왼쪽
>	크다	7	왼쪽
<=	같거나 작다	7	왼쪽
>=	같거나 크다	7	왼쪽
==	같다	8	왼쪽
!=	같지 않다	8	왼쪽

동등 연산자를 사용하면 같은 타입의 두 객체 변수를 비교할 수 있습니다. 이런 경우 두 변수가 같은 객체를 참조하면 비교의 결과는 true입니다. 예를 들면 다음과 같습니다.

```
m1 = new Mammal();
m2 = new Mammal();
boolean b1 = m1 == m2;  //b1 is false

m1 = m2;
boolean b2 = m1 == m2;  //b2 is true
```

m1과 m2는 같은 타입이지만 서로 다른 객체를 참조하기 때문에 첫 번째 동등 테스트의 결과는 false입니다. 두 변수 모두 같은 객체를 나타내므로 두 번째 비교는 true입니다.

참고 그러나 대부분의 경우 비교 연산자 대신 Object 클래스의 equals() 메소드를 사용합니다. equals()를 사용하여 비교 클래스의 객체를 비교하기 전에 비교 클래스는 Object의 하위 클래스가 되어야 합니다.

비트 단위 연산자

비트 단위 연산자에는 시프트 연산자와 부울 연산자가 있습니다. 시프트 연산자는 정수의 이진수 숫자를 오른쪽이나 왼쪽으로 시프트하는 데 사용됩니다. 예제를 보면 다음과 같습니다. 간단하게 하기 위해 short 정수 타입이 int 대신 사용됩니다.

```
short i = 13; //i is 0000000000001101
i = i << 2;   //i is 0000000000110100
```

둘째 줄에서 비트 단위 왼쪽 시프트 연산자는 i 두 지점의 모든 비트 단위를 왼쪽으로 시프트합니다.

참고 *부호 있는* 정수를 사용하는 방법에 있어서 Java의 시프팅 연산과 C/C++의 시프팅 연산은 다릅니다.부호 있는 정수에서 맨 왼쪽 비트가 정수의 부호를 나타내는 데 사용됩니다. 정수가 음수이면 비트는 1이고 양수이면 0입니다.Java에서 정수는 항상 부호가 있지만 C/C++에서 정수는 기본 설정에 따라 부호를 가집니다. C/C++의 구현 대부분에서 비트 단위 시프트 연산은 정수의 부호를 유지하지 않지만 부호 비트는 시프트됩니다. 그러나 Java에서 >>>를 사용하여 *부호 없는 시프트*를 수행하지 않으면 시프트 연산자는 부호 비트를 유지합니다. 부호 비트가 복제되어 시프트된다는 의미입니다.예를 들어, 10010011을 1만큼 오른쪽으로 시프트하면 11001001이 됩니다.

다음 표는 Java의 비트 단위 연산자를 나열한 것입니다.

연산자	정의	Prec.	Assoc.
~	비트 단위 NOT 피연산자의 각 비트를 반전하므로 모든 0은 1이 되고 1은 0이 됩니다.	2	오른쪽
<<	부호 있는 왼쪽으로 시프트 왼쪽 피연산자의 비트를 오른쪽 피연산자에 지정된 자리수만큼 왼쪽으로 시프트하여, 0을 오른쪽으로부터 시프트합니다.상위 비트는 잃게 됩니다.	6	왼쪽
>>	부호 있는 오른쪽으로 시프트 오른쪽에 지정된 자리수에 따라 왼쪽 피연산자의 비트를 오른쪽으로 시프트합니다.왼쪽 피연산자가 음수이면 0은 왼쪽으로부터 시프트되고 양수이면 1이 시프트됩니다. 이렇게 하여 원래 부호를 유지합니다.	6	왼쪽
>>>	Zero-fill(0 채움) 오른쪽 시프트 오른쪽으로 시프트하지만 항상 0으로 채웁니다.	6	왼쪽
&	비트 단위 AND 값을 할당하기 위해 =와 함께 사용될 수 있습니다.	9	왼쪽
|	비트 단위 OR 값을 할당하기 위해 =와 함께 사용될 수 있습니다.	10	왼쪽
^	비트 단위 XOR 값을 할당하기 위해 =와 함께 사용될 수 있습니다.	11	왼쪽
<<=	할당과 함께 왼쪽 시프트	15	왼쪽
>>=	할당과 함께 오른쪽 시프트	15	왼쪽
>>>=	할당과 함께 Zero-fill(0 채움) 오른쪽 시프트	15	왼쪽

?:-3항 연산자

?:는 Java가 C에서 가져온 3항 연산자입니다. 매우 간단한 if-then-else 종류의 문을 만드는 매우 간단한 방법을 제공합니다.

다음 구문과 같습니다.

```
<expression 1> ? <expression 2> :<expression 3>;
```

expression 1이 먼저 계산됩니다. true이면 expression 2가 계산됩니다. expression 2가 false이면 expression 3이 사용됩니다.예를 들면, 다음과 같습니다.

```
int x = 3, y = 4, max;
max = (x > y) ? x :y;
```

이 때 max에는 더 큰 값에 따라 x 또는 y 값이 할당됩니다.

메소드 사용

메소드는 어떤 작업을 수행하는 것입니다. 메소드는 다른 메소드를 포함할 수 없지만 변수 및 클래스 참조는 포함할 수 있습니다.

다음은 리뷰할 간단한 예제입니다. 다음 메소드는 재고를 사용하여 레코드 가게를 돕습니다.

```
//Declare the method: return type, name, args:
public int getTotalCDs(int numRockCDs, int numJazzCDs, in numPopCDs) {
  //Declare the variable totalCDs. The other three variables are declared
elsewhere:
  int totalCDs = numRockCDs + numJazzCDs + numPopCDs;
  //Make it do something useful. In this case, print this line on the screen:
  System.out.println("Total CDs in stock = " + totalCDs);
}
```

Java에서는 메소드마다 다른 인수를 사용하는 경우 같은 이름으로 둘 이상의 메소드를 정의할 수 있습니다. 예를 들어 public int getTotalCDs(int numRockCDs, int numJazzCDs, in numPopCDs)와 public int getTotalCDs(int salesRetailCD, int salesWholesaleCD)는 모두 같은 클래스에서 유효합니다. Java는 다른 패턴의 인수(*메소드 시그너처*)를 인식하고 호출할 때 올바른 메소드를 적용합니다. 다른 메소드에 같은 이름을 지정하는 것을 *메소드 오버로드*라고 합니다.

프로그램의 다른 부분에서 메소드에 액세스하려면 다음과 같이 먼저 메소드가 상주하는 클래스의 인스턴스를 만든 다음 해당 객체를 사용하여 메소드를 호출합니다.

```
//Create an instance totalCD of the class Inventory:
Inventory totalCD = new Inventory();

//Access the getTotalCDs() method inside of Inventory, storing the value in
total:
int total = totalCD.getTotalCDs(myNumRockCDs, myNumJazzCDs,
myNumPopCDs);
```

배열 사용

배열의 크기는 선언의 일부가 아닙니다. 배열에 필요한 메모리는 배열을 초기화할 때까지 실제로 할당되지 않습니다.

배열을 초기화하고 필요한 메모리를 할당하려면 다음과 같이 new 연산자를 사용해야 합니다.

```
int studentID[] = new int[20];              // Creates array of 20 int elements.
char[] grades = new char[20];               // Creates array of 20 char
elements.
float[][] coordinates = new float[10][5];  // 2-dimensional, 10x5 array of
float
                                            // elements.
```

참고 2차원 배열을 만들 때 첫 번째 배열 숫자는 열 번호를 정의하고 두 번째 배열 숫자는 행 번호를 정의합니다.

Java는 0에서 시작하여 위치를 카운트합니다. 즉, 요소가 20개인 배열의 요소는 0에서 19까지의 번호가 지정됩니다. 첫 번째 요소는 0, 두 번째는 1, 등의 순서입니다. 배열 작업을 할 때는 카운트 방법에 주의해야 합니다.

배열이 만들어질 때 모든 요소의 값은 null 또는 0이며, 그 값은 나중에 할당됩니다.

참고 Java의 new 연산자 사용은 C의 malloc 명령 및 C++의 new 연산자 사용과 비슷합니다.

배열을 초기화하려면 중괄호 안에 있는 배열 요소의 값을 지정합니다. 다차원 배열의 경우에는 중첩된 중괄호를 사용하십시오.예를 들면, 다음과 같습니다.

```
char[] grades = {'A', 'B', 'C', 'D', 'F');
float[][] coordinates = {{0.0, 0.1}, {0.2, 0.3}};
```

첫 번째 문장에서는 grades라는 char 배열을 만듭니다. 'A'에서 'F' 사이의 값으로 배열 요소를 초기화합니다. 이 배열을 만드는 데 new 연산자를 사용할 필요가 없습니다. 배열을 초기화하면 배열에 충분한 메모리가 자동으로 할당되어 초기화된 모든 값을 저장합니다. 따라서 첫 번째 문장은 요소가 5개인 char 배열을 만듭니다.

두 번째 문장에서는 coordinates라는 2차원 float 배열을 만들며, 그 크기는 2 * 2입니다. 기본적으로 coordinates는 두 가지 배열 요소로 구성되는 배열입니다. 배열의 첫 행은 0.0과 0.1로, 두 번째 행은 0.2와 0.3으로 초기화됩니다.

생성자 사용

*클래스*는 논리적으로 일관된 변수, 속성 및 동작 집합을 정의하는 한 쌍의 중괄호로 둘러싸인 코드입니다. *패키지*는 논리적으로 연관된 클래스 집합입니다.

클래스는 지시어 집합에 불과합니다. 그 자체로는 아무것도 수행하지 않습니다. 그것은 요리법과 비슷합니다. 올바른 요리법으로 케익을 만들 수 있지만 요리법은 케익이 아닌 지시에 불과합니다. 케익은 요리법의 지시에 따라 만든 *객체*에 해당합니다. Java에서는 Cake 요리법을 보고 케익의 *인스턴스*를 만들었다고 말할 수 있습니다.

클래스의 인스턴스를 만드는 동작을 객체 *인스턴스화*라고 합니다. *클래스의 객체*를 *인스턴스화*합니다.

객체를 인스턴스화하려면 할당 연산자(=), 키워드 new 및 *생성자*라는 특수한 종류의 메소드를 사용합니다. 생성자를 호출하려면 인스턴스화된 클래스 이름과 그 뒤에 한 쌍의 괄호를 사용합니다. 메소드처럼 보이지만 클래스의 이름을 사용하는데 그것이 대문자를 사용하는 이유이기도 합니다.

```
<ClassName> <instanceName> = new <Constructor()>;
```

예를 들어 Geek 클래스의 새 객체를 인스턴스화하고 인스턴스 thisProgrammer의 이름을 지정하려면 다음과 같이 입력합니다.

```
Geek thisProgrammer = new Geek();
```

생성자는 클래스의 새 인스턴스를 설정합니다. 생성자는 해당 클래스의 모든 변수를 초기화하여 즉시 사용할 수 있게 합니다. 또한 객체에 필요한 모든 시작 루틴을 수행할 수도 있습니다.

예를 들어 차를 운전하려면 가장 먼저 문을 열고 차에 탄 다음 클러치를 밟고 엔진을 시작합니다. 그런 다음 기어를 넣고 액셀러레이터를 사용하는 것과 같은 일반적으로 운전과 관련된 일을 수행할 수 있습니다. 생성자는 차에 타고 시작하는 것과 관련된 객체 및 동작과 같은 부분을 프로그램에서 처리합니다.

인스턴스를 만들고 나면 인스턴스 이름을 사용하여 해당 클래스의 멤버에 액세스할 수 있습니다.

생성자에 관한 자세한 내용은 17- 4페이지 "사례 연구: 간단한 OOP 예제"를 참조하십시오.

멤버 액세스

액세스 연산자(.)를 사용하여 인스턴스화된 객체 안의 멤버에 액세스합니다. 기본 구문은 다음과 같습니다.

```
<instanceName>.<memberName>
```

멤버 이름의 정확한 구문은 멤버 종류에 따라 달라집니다. 이 구문에는 변수(<memberName>), 메소드(<memberName>()) 또는 하위 클래스(<MemberName>)가 포함될 수 있습니다.

멤버를 액세스하려는 다른 구문 요소 내부에서 이 연산을 사용할 수 있습니다. 예를 들면, 다음과 같습니다.

```
setColor(Color.pink);
```

이 메소드에는 자신의 작업을 수행할 색이 필요합니다. 프로그래머는 액세스 연산을 인수로 사용하여 Color 클래스안의 pink 변수에 액세스합니다.

배열

배열 요소는 배열 변수를 아래 첨자로 하거나 *인덱싱*하여 액세스합니다. 배열 변수를 인덱싱하려면 배열 변수 이름 다음에 대괄호로 둘러싸인 요소 번호(인덱스)를 사용합니다. *배열은 항상 0에서 시작하여 인덱싱됩니다.* 마치 1부터 번호를 지정하는 것과 같이 코딩하는 실수를 흔히 범합니다.

다차원 배열의 경우에는 인덱스를 사용하여 각 차원이 요소를 액세스하도록 해야 합니다. 첫 번째 인덱스는 행이고 두 번째 인덱스는 열입니다.

예를 들면, 다음과 같습니다.

```
firstElement = grades[0];   //firstElement = 'A'
fifthElement = grades[4];   //fifthElement = 'F'
row2Col1 = coordinates[1][0]; //row2Col1 = 0.2
```

코드의 다음 부분은 배열의 한 가지 사용을 보여 줍니다. intArray라는 5개 int 요소의 배열을 만든 다음 for 루프를 사용하여 정수 0에서 4를 배열의 요소에 저장합니다.

```
int[] intArray = new int [5];
int index;
for (index = 0; index < 5; index++) intArray [index] = index;
```

이 코드는 index 변수를 0에서 4로 증가시키고 모든 단계에서 변수 index로 인덱싱된 intArray 요소에 그 값을 저장합니다.

Chapter

15

자바 언어 제어

이 단원에서는 이 장 전체에서 사용될 Java 프로그램 언어의 제어에 대한 기본 개념을 설명합니다.여기서는 사용자가 일반적인 프로그래밍 개념은 이해하지만 Java에 대한 경험이 거의 없다고 가정합니다.

용어

다음과 같은 용어와 개념들이 이 장에 설명되어 있습니다.

- 문자열 처리
- 타입 변환
- 반환 타입 및 문장
- 흐름 제어문

문자열 처리

String 클래스는 메소드를 제공하여 부분 문자열을 얻거나 문자열 내에서 문자를 *인덱싱*할 수 있도록 해줍니다. 그러나 선언된 String 값은 변경할 수 없습니다. 해당 변수와 연결된 String 값을 변경해야 하는 경우에는 변수가 새 값을 가리키도록 합니다.

```
String text1 = new String("Good evening."); // Declares text1 and assigns a value.
text1 = "Hi, honey, I'm home!"              // Assigns a new value to text1.
```

인덱싱을 통해 문자열의 특정 문자를 가리킬 수 있습니다. Java는 0에서부터 시작하여 문자열 각각의 위치를 세기 때문에 첫번째 위치는 0이고 두번째 위치는 1, 이런 식이 됩니다. 따라서 문자열의 여덟번째 위치는 인덱스 7입니다.

StringBuffer 클래스는 해결책을 제공합니다. 또한 문자열의 내용을 처리하는 몇 가지 다른 방법을 제공합니다. StringBuffer 클래스는 크기를 명시

적으로 제어할 수 있는 버퍼(메모리의 특정 영역)에 문자열을 저장합니다. 따라서 String을 선언하고 문자열을 확정하기 전에 필요한 만큼 문자열을 변경할 수 있습니다.

일반적으로 String 클래스는 문자열 저장용이며 StringBuffer 클래스는 문자열 처리용입니다.

타입 변환

데이터 타입 값은 다른 타입으로 변환할 수 있습니다. 클래스 값은 동일한 클래스 계층에 있는 경우 이 클래스에서 다른 클래스로 변환할 수 있습니다. 변환은 한 연산에서 컴파일러의 값에 대한 인식을 변경할 뿐 해당 값의 원래 타입은 변경하지 않음에 유의하십시오.

명확한 논리 제한이 적용됩니다. 작은 타입에서 큰 타입으로의 확장 변환은 쉽지만 큰 타입(예: double 또는 Mammal)에서 작은 타입(예: float 또는 Bear)으로의 축소 변환은 데이터가 확실히 새 타입의 매개변수에 맞지 않는 한 데이터에 위험을 가져옵니다. 축소 변환은 *타입 변환*이라는 특수 연산을 필요로 합니다.

다음 표는 기본 값의 확대 변환을 보여 줍니다. 이러한 변환은 데이터에 위험을 초래하지 않습니다.

원래 타입	변환 타입
byte	short, char, int, long, float, double
short	int, long, float, double
char	int, long, float, double
int	long, float, double
long	float, double
float	double

데이터 타입을 변환하려면 괄호 안에 *변환하려는* 타입을 먼저 입력하고 그 뒤에 변환하려는 변수를 입력합니다. (int)x다음은 컨텍스트에서 보여지는 것으로, 여기서 x는 변환 중인 변수이고 float는 원래 데이터 타입이며 int는 대상 데이터 타입이고 y는 새 값을 저장한 변수입니다.

```
float x = 1.00;    //declaring x as a float
int y = (int)x; //casting x to an int named y
```

위에서 x의 값은 int의 내부에 꼭맞다고 가정합니다.x의 소수점 값이 변환 중 없어졌다는 것에 유의하십시오.Java는 소수를 가장 가까운 정수로 반올림합니다.

반환 타입 및 문장

메소드 선언은 변수 선언이 데이터 타입을 필요로 하는 것과 같이 반환 타입을 필요로 합니다. 반환 타입은 void를 제외하고는 데이터 타입(int, boolean, String 등)과 동일합니다.

void는 특수 반환 타입입니다. 이것은 void가 완료되었을 때 메소드가 어떤 것도 반환할 필요가 없다는 것을 의미합니다. void 반환 타입은 일반적으로 정보 전달외의 작업을 수행할 때 필요한 액션 메소드에 사용됩니다.

그밖의 모든 반환 타입은 메소드의 끝에서 return문을 필요로 합니다. void 메소드에서 return문을 사용하여 특정 지점에 메소드를 남겨둘 수 있지만 그밖의 다른 지점에서는 그럴 필요가 없습니다.

return 문은 return이라는 단어와 문자열, 데이터, 변수 이름 또는 필요한 연결 중의 하나로 구성됩니다.

```
return numCD;
```

일반적으로 연결을 위해 괄호를 사용합니다.

```
return ("Number of files: " + numFiles);
```

흐름 제어문

흐름 제어문은 프로그램에 명령 방법 및 사용자가 제공한 정보의 사용 방법을 알려 줍니다. 흐름 제어를 통해 사용자는 문장 반복, 문장 조건 지정, 재귀 루프 생성, 루프 동작 제어를 수행할 수 있습니다.

흐름 제어문은 루프를 생성하는 for, while 및 do-while과 같은 *반복* 문장과 문장 사용에 조건을 지정하는 switch, if, if-else, if-then-else 및if-else-if 사다리와 같은 *선택*문, 제어를 프로그램의 다른 부분으로 전환하는 break, continue 및return과 같은 *점프*문 등의 세 종류의 문장으로 나뉩니다.

흐름 제어의 특수 형태로 *예외 처리*가 있습니다. 예외 처리는 프로그램의 런타임 오류를 잡아내는 구조적 방법을 제공하며 이에 관한 상세 정보를 반환하도록 합니다. 또한 예외 핸들러를 설정하여 프로그램이 종료하기 전에 동작을 수행할 수 있습니다.

개념 적용

다음 단원에서는 이 장의 초반부에 소개된 용어 및 개념을 적용하는 방법에 대해 설명합니다.

이스케이프 시퀀스

문자 리터럴의 특수 타입을 *이스케이프 시퀀스*라고 합니다. C/C++와 같이 Java는 이스케이프 시퀀스를 사용하여 특수 제어 문자 및 인쇄할 수 없는 문자를 나타냅니다. 이스케이프 시퀀스는 백슬래시(₩) 뒤에 문자 코드를 붙여 나타냅니다. 다음 표는 이러한 이스케이프 시퀀스를 요약한 것입니다.

Character	Escape Sequence
백슬래시	₩₩
백스페이스	₩b
캐리지 리턴	₩r
이중 인용 부호	₩"
폼 피드	₩f
수평 탭	₩t
새 라인	₩n
8진수 문자	₩DDD
단일 인용 부호	₩'
유니코드 문자	₩uHHHH

10진수가 아닌 숫자 문자는 이스케이프 시퀀스입니다. 8진수 문자는 8진수 숫자 세 개로 나타내며 유니코드 문자는 뒤에 16진수 숫자 네 개가 붙는 소문자 u로 나타냅니다. 예를 들어, 십진수 57은 8진 코드 ₩071와 유니코드 시퀀스 ₩u0039로 표시됩니다.

다음 문장의 예제 문자열은 첫번째 줄에서 두 개의 탭으로 구분된 단어 Name과 "Hildegaard von Bingen"을 출력하며 역시 두번째 줄에서 두 개의 탭으로 구분된 ID와 "1098"을 출력합니다.

```
String escapeDemo = new
  String("Name₩t₩t₩"Hildegaard von Bingen₩"₩nID₩t₩t₩"1098₩"");
```

문자열 처리

String에 지정한 문자의 문자열은 리터럴이며 프로그램은 문자열을 변경하지 않고 정확히 사용자가 지정한 문자열을 사용합니다. 그러나 String 클래스는 문자열을 체인화(*문자열 연결*)하고 문자열에 무엇이 있는지 보고 사용하며(문자열 비교, 문자열 검색, 문자열에서 부분 문자열을 추출) 다른 종류의 데이터를 문자열로 변환할 방법을 제공합니다. 예를 들면, 다음과 같습니다.

- String 타입의 변수를 선언하며 값을 지정합니다.

```
String firstNames = "Joseph, Elvira and Hans";
String modifier = " really ";
String tastes = "like chocolate.";
```

- 9번째 열에서부터 문자열의 마지막까지 선택하여 문자열에서 부분 문자열을 얻습니다.

```
String sub = firstNames(9); // "Elvira and Hans"
```

- 부분 문자열 일부를 다른 문자열과 비교하며 문자열을 대문자로 변환한 다음 다른 문자열과 연결하여 반환 값을 얻습니다.

```
boolean bFirst = firstNames.startsWith("Emine");  // Returns false in this
case.
String caps = modifier.toUpperCase();             // Yields " REALLY "
return firstNames + caps + tastes;              // Returns the line:
                                  // Elvira and Hans REALLY like chocolate.
```

String 클래스 사용 방법에 관한 자세한 내용은 http://java.sun.com/j2se/1.3/docs/api/java/lang/String.html에 있는 Sun의 API 설명서를 참조하거나 Patrick Chan과 Rosanna Lee의 *Java Class Libraries : Annotated Reference*를 참조하십시오. 자세한 내용은 부록을 참조하십시오.

String 핸들러 목록에 대한 내용은 David Flanagan의 *Java in a Nutshell*을 참조하십시오. String 처리에 관한 상세 설명은 Patrick Naughton의 *Java Handbook*을 참조하십시오.

StringBuffer

문자열을 보다 효과적으로 제어하려는 경우 StringBuffer 클래스를 사용합니다. 이 클래스는 java.lang 패키지의 일부입니다.

StringBuffer는 문자열을 버퍼에 저장하므로 필요할 때까지는 확정 String을 선언할 필요가 없습니다. StringBuffer의 몇가지 이점은 내용이 변경되었을 경우에도 String을 다시 선언할 필요가 없다는 것입니다. 기존의 버퍼 크기보다 큰 버퍼 공간을 확보할 수 있습니다.

StringBuffer는 String의 메소드 외에도 다른 메소드를 제공하는데 이들 메소드를 통해 새로운 방법으로 문자열의 내용을 수정할 수 있습니다. 예를 들어, StringBuffer의 setCharAt() 메소드는 첫번째 매개변수에 지정된 인덱스에 있는 문자를 두번째 매개변수에 지정된 새 값으로 변경합니다.

```
StringBuffer word = new StringBuffer ("yellow");
word.setCharAt (0, 'b');  //word is now "bellow".
```

액세스 결정

기본적으로 클래스는 클래스 내의 모든 멤버에서 사용할 수 있으며 클래스의 멤버는 서로 사용할 수 있습니다. 그러나 이러한 액세스는 광범위하게 수정할 수 있습니다.

*액세스 변경자*는 클래스 또는 멤버의 정보가 다른 멤버 및 클래스에 보여지는 방법을 결정합니다. 액세스 변경자는 다음을 포함합니다.

- public: public 멤버는 부모 클래스가 보이는 한 public 멤버에 속하지 않는 멤버에게도 보입니다. public 클래스는 다른 모든 패키지의 기타의 모든 클래스에 보여집니다.
- private: private 멤버 액세스는 멤버의 클래스로 제한됩니다.
- protected: protected 멤버에는 멤버의 부모 클래스가 액세스 가능한 한 클래스의 다른 멤버와 동일한 패키지의 클래스 멤버가 액세스할 수 있지만 다른 패키지의 클래스 멤버에서는 액세스할 수 없습니다. protected 클래스는 동일한 패키지의 다른 클래스에서 사용할 수 있으나 다른 패키지의 클래스에서는 사용할 수 없습니다.
- 액세스 변경자가 선언되지 않은 경우 멤버는 부모 패키지의 모든 클래스에서 사용할 수 있으나 패키지 외부의 클래스에서는 사용할 수 없습니다.

컨텍스트에서 이를 살펴보면 다음과 같습니다.

```
class Waistline {
  private boolean invitationGiven = false;  // This is private.
  private int weight = 170                  // So is this.

  public void acceptInvitation() {          // This is public.
    invitationGiven = true;
  }

  //Class JunkFood is declared and object junkFood is instantiated
elsewhere:
  public void eat(JunkFood junkFood) {

    /*This object only accepts more junkFood if it has an invitation
    * and if it is able to accept. Notice that isAcceptingFood()
    * checks to see if the object is too big to accept more food:
    */
    if (invitationGiven && isAcceptingFood()) {

      /*This object's new weight will be whatever its current weight
      * is, plus the weight added by junkFood. Weight increments
      * as more junkFood is added:
      */
      weight += junkFood.getWeight();
    }
  }
```

```
    //Only the object knows if it's accepting food:
    private boolean isAcceptingFood() {
      // This object will only accept food if there's room:
      return (isTooBig() ? false :true};
    }

    //Objects in the same package can see if this object is too big:
    protected boolean isTooBig() {
      //It can accept food if its weight is less than 185:
      return (weight > 185) ? true : false;
    }
  }
```

isAcceptingFood()와 invitationGiven이 private임을 유의하십시오. 이 클래스의 멤버만이 이 객체가 음식을 받아들일 수 있는지 또는 초대장을 가지고 있는지 여부를 알 수 있습니다.

isTooBig()은 protected입니다. 이 패키지의 클래스만이 이 객체가 중량 제한을 초과했는지 여부를 알 수 있습니다.

외부에 공개되는 메소드는 acceptInvitation()과 eat()뿐입니다. 모든 클래스에서 이러한 메소드를 인지할 수 있습니다.

메소드 처리

main() 메소드는 특별한 주의를 필요로 합니다. 이 메소드는 프로그램으로의 엔트리 포인트로(애플릿 예외) 다음과 같이 나타납니다.

```
public static void main(String[] args) {
  ...
}
```

특정 변수가 괄호 안에 허용되지만 일반적인 형태는 정해져 있습니다.

키워드 static이 중요합니다. static 메소드는 항상 해당 클래스의 특정 인스턴스보다는 전체 클래스와 연결되어 있습니다. (키워드 static은 또한 클래스에 적용할 수 있습니다. static 클래스의 모든 멤버는 클래스의 전체 부모 클래스와 연결되어 있습니다.) static 메소드는 또한 *클래스 메소드*라고도 합니다.

main() 메소드가 프로그램의 시작 포인트이기 때문에 프로그램이 부모 클래스에서 생성하는 여러 객체의 독립성을 유지하려면 이 메소드는 static이어야 합니다.

static의 전체 클래스 연결은 static 메소드 호출 방법 및 static 메소드에서의 다른 메소드 호출 방법에 영향을 미칩니다. static 멤버는 메소드의 이름을 이용하여 다른 타입의 멤버에서 호출할 수 있으며 static 멤버는 동일한 방법으로 서로를 호출할 수 있습니다. 클래스의 static 메소드에 액세스하기 위해 클래스의 인스턴스를 생성할 필요는 없습니다.

static 메소드에서 비static 클래스의 비static 멤버에 액세스하려면 액세스하려는 멤버의 클래스를 인스턴스화해야 하며 다른 메소드 호출에서와 마찬가지로 해당 인스턴스를 액세스 연산자와 함께 사용해야 합니다.

main() 메소드에 대한 인수는 다른 인수를 허용하는 String 배열이어야 함을 유의하십시오. 이 메소드는 컴파일러가 작업을 시작하는 지점임을 유의해야 합니다. 명령 줄에서 인수를 전달할 경우 이 인수는 main() 메소드 선언의 String 배열에 문자열로 전달되며 해당 인수를 사용하여 프로그램 실행을 시작합니다. String이 아닌 데이터 타입을 전달할 경우에도 데이터 타입은 문자열로 수신됩니다. main() 메소드의 몸체에 String에서 필요한 데이터 타입으로 변환하는 데 필요한 코드를 코딩해야 합니다.

타입 변환 사용

리뷰 타입 변환은 특정 연산 과정 동안 변수의 데이터 타입을 변환하는 프로세스입니다. 축소 변환의 표준 형태를 타입 변환이라고 하는데 이 타입 변환은 데이터에 위험을 가져올 수 있습니다.

암시적 타입 변환

컴파일러가 암시적으로 타입 변환을 수행하는 시간이 있습니다. 예를 들면, 다음과 같습니다.

```
if (3 > 'a') {
  ...
}
```

이런 경우 'a' 값은 숫자 3과 비교하기 전에 정수 값(문자 a에 대한 아스키 값)으로 변환됩니다.

명시적 변환

확장 타입 변환 구문은 다음과 같이 간단합니다.

```
<nameOfOldValue> = (<new type>) <nameOfNewValue>
```

Java에서는 축소 변환 수행을 권장하지 않기 때문에 변환을 시행할 경우 다음과 같이 명시적이어야 합니다.

```
floatValue = (float)doubValue;   // To float "floatValue"
                                 // from double "doubValue".

longValue = (long)floatValue;    // To long "longValue"
                                 // from float "floatValue".
                                 // This is one of four possible constructions.
```

(10진수는 기본적으로 반올림합니다.) 이 프로세스는 복잡할 수 있기 때문에 타입 변환하려는 타입의 구문을 명확히 이해하고 있어야 합니다.

자세한 내용은 22-5페이지 "데이터 타입 변환 및 타입 변환"을 참조하십시오.

흐름 제어

리뷰 루프 문장에는 다음과 같이 세가지 종류의 루프 문장, 즉 루프를 만드는 반복문(for, while 및 do-while), 프로그램에 프로그램이 어떤 환경에서 문장

을 사용하게 되는지를 설명하는 선택문(switch 및 모든 if문)과 프로그램의 다른 부분으로 제어를 시프트하는 점프문(break, continue 및return)이 있습니다.

루프

프로그램의 각 문장은 한 번씩 실행됩니다. 그러나 조건을 만족할 때까지 문장을 여러 차례 실행해야 하는 경우도 있습니다. Java는 문장에 while, do 및 for와 같은 세 가지 루푸 방법을 제공합니다.

- **while 루프**

  while 루프는 특정 조건을 만족할 때까지 실행되는 코드 블럭을 생성하는 데 사용됩니다. 다음은 일반적인 while 루프 구문입니다.

```
while ( <boolean condition statement> ) {
  <code to execute as long as that condition is true>
}
```

  루프는 먼저 조건을 검사합니다. 조건 값이 true인 경우 전체 블록을 실행합니다. 그런 다음 다시 조건을 계산하고 조건이 false가 될 때까지 이 프로세스를 반복합니다. false가 되는 시점에서 루프는 실행을 중지합니다. 예를 들면, 다음과 같이 "Looping"을 10번 출력합니다.

```
int x = 0;                          //Initiates x at 0.
while (x < 10){                       //Boolean condition statement.
  System.out.println("Looping");      //Prints "Looping" once.
  x++;                              //Increments x for the next iteration.
}
```

  루프가 처음으로 실행을 시작하면 x 값이 10보다 작은지 검사합니다. 값이 10보다 작으면 루프의 몸체가 실행됩니다. 이런 경우 화면에 "Looping"이라는 단어가 출력된 다음 x 값이 증가됩니다. 이 루프는 루프의 실행이 중단되는 x 값이 10이 될 때까지 계속됩니다.

  무한 루프를 작성하려고 하지 않는 한 조건 값이 false가 되고 루프가 종료되는 지점이 루프에 있어야 합니다. 또한 return, continue 또는 break 문장을 사용하여 루프를 종료할 수 있습니다.

- **do-while 루프**

  do-while 루프는 문장의 앞이 아니라 *뒤에서* 조건을 계산한다는 점을 제외하고 while 루프와 유사합니다. 다음 코드는 do 루프로 변환된 앞의 while 루프를 보여 줍니다.

```
int x = 0;
do{
  System.out.println("Looping");
  x++;
}
while (x < 10);
```

  두 루프 구조의 가장 큰 차이점은 do-while 루프는 항상 최소 한 번은 실행되지만 while 루프는 초기 조건이 만족되지 않는 경우 한 번도 실행되지 않는다는 것입니다.

- **for 루프**

  for 루프는 가장 강력한 루프 구조입니다. 다음은 for 루프의 일반적인 구문입니다.

```
for ( <initialization> ; <boolean condition> ; <iteration> ) {
  <execution code>
}
```

  for 루프는 초기화 표현식, Boolean 조건 표현식 및 반복 표현식의 세 부분으로 구성됩니다. 세번째 표현식은 일반적으로 첫번째 표현식에서 초기화되는 루프 변수를 업데이트합니다. 다음은 앞의 while 루프와 동일한 for 루프입니다.

```
for (int x = 0; x < 10; x++){
  System.out.println("Looping");
}
```

  이 for 루프와 이와 동일한 while 루프는 매우 유사합니다. 거의 모든 for 루프의 경우 while 루프와 동일합니다.

  for 루프는 가장 많이 사용되는 루프 구조로 매우 효율적입니다. 예를 들면, while 루프와 for 루프 모두 1부터 20까지 숫자를 증가시키지만 for 루프는 한 줄에서 이를 수행할 수 있습니다.

  While:

```
int x = 1, z = 0;
while (x <= 20) {
  z += x;
  x++;
}
```

  For:

```
int z = 0;
for (int x=1; x <= 20; x++) {
  z += x;
}
```

for 루프에 여러 조건을 달아 루프의 실행 횟수를 반으로 줄일 수 있습니다.

```
for (int x=1,y=20, z=0; x<=10 && y>10; x++, y--) {
  z+= x+y;
}
```

이 루프를 다음과 같이 네 가지 기본 섹션으로 나누어 볼 수 있습니다.

1 초기화 표현식: int x =1, y=20, z=0
2 Boolean 조건: x<=10 && y>10
3 반복 표현식: x++, y--
4 실행 코드의 기본 몸체는 다음과 같습니다. z+= x + y

루프 제어문

이러한 문장은 루프 문장에 제어 기능을 추가합니다.

- **break 문**

  break 문을 통해 테스트 조건을 만족하기 전에 루프 구조를 빠져 나올 수 있습니다. break 문을 만나면 루프는 즉시 종료되고 남아있는 모든 코드를 생략합니다. 예를 들면, 다음과 같습니다.

```
int x = 0;
while (x < 10){
  System.out.println("Looping");
  x++;
  if (x == 5)
    break;
else
    ... //do something else
}
```

  이 예제에서 루프는 x가 5가 될 때 실행을 중지합니다.

- **continue 문**

  continue 문은 루프의 나머지를 생략하고 다음 루프 반복에서 실행을 재개하는데 사용됩니다.

```
for (int x = 0; x < 10; x++){
  if(x == 5)
    continue; //go back to beginning of loop with x=6
  System.out.println("Looping");
}
```

  이 예제는 x가 5인 경우 "Looping"을 출력하지 않지만 x가 6,7,8,9가 되는 경우를 위해 계속해서 출력합니다.

조건문

조건문은 코드에 의사 결정 기능을 제공하는 데 사용됩니다. Java에는 다음과 같은 if-else 문과 switch 문이 있습니다.

- **if-else 문**

  if-else 문의 구문은 다음과 같습니다.

```
if (<condition1>) {
  ... //code block 1
}
else if (<condition2>) {
  ... //code block 2
}
else {
  ... //code block 3
}
```

  if-else 문은 일반적으로 여러 블록으로 구성됩니다. 여러 조건 중 하나가 true인 상태에서 if-else 문이 실행될 경우에는 오직 하나의 블록만이 실행됩니다.

  else-if 및 else 블록은 옵션입니다. 또한 if-else 문은 세가지 블록으로 제한되지 않습니다. 필요한 만큼의 여러 else-if 블록을 포함할 수 있습니다.

  다음 예제는 if-else 문의 사용을 설명합니다.

```
if ( x % 2 == 0)
  System.out.println("x is even");
else
  System.out.println("x is odd");
if (x == y)
  System.out.println("x equals y");
else if (x < y)
  System.out.println("x is less than y");
else
  System.out.println("x is greater than y");
```

- **switch 문**

  switch 문은 if-else 문과 유사합니다. 다음은 switch 문의 일반적인 구문입니다.

```
switch (<expression>){
  case <value1>: <codeBlock1>;
    break;
  case <value2>: <codeBlock2>;
    break;
     default: <codeBlock3>;
}
```

다음 사항에 유의하십시오.

- 코드 블록에 하나의 문장만이 있는 경우 중괄호를 표시할 필요가 없습니다.
- default 코드 블록은 if-else 문의 else 블록에 해당합니다.
- 코드 블록은 조건이 아니라 변수 또는 표현식의 값을 기반으로 실행됩니다.
- <expression>의 값은 정수 타입이거나 char와 같이 int로 안전하게 변환할 수 있는 타입이어야 합니다.
- case 값은 원래 expression과 동일한 데이터 타입의 상수 표현식이어야 합니다.
- break 키워드는 옵션이며 코드 블록이 실행되면 switch 문의 실행을 종료하는 데 사용됩니다. codeBlock1 다음에 사용되지 않는 경우 codeBlock2는 codeBlock1이 실행을 종료한 뒤 바로 실행됩니다.
- expression이 값 숫자 중 하나일 때 코드 블록을 실행해야 하는 경우 각 값은 case <value>:와 같이 지정되어야 합니다.

예를 들면, c가 char 타입인 경우는 다음과 같습니다.

```
switch (c){
  case '1':case '3':case '5':case '7':case '9':
    System.out.println("c is an odd number");
    break;
  case '0':case '2':case '4':case '6':case '8':
    System.out.println("c is an even number");
    break;
  case ' ':
    System.out.println("c is a space");
    break;
     default:
    System.out.println("c is not a number or a space");
  }
```

switch는 c를 계산하고 값이 c와 동일한 case 문으로 점프합니다. case 값 어느 것도 c와 동일하지 않은 경우 default 섹션이 실행됩니다. 각 블럭당 얼마나 많은 값을 사용할 수 있는지 살펴보십시오.

예외 처리

예외 처리는 프로그램에서 런타임 오류를 잡아내고 이에 관한 상세 정보를 반환하도록 하는 구조적 방법을 제공합니다.또한 예외 핸들러를 설정하여 프로그램이 종료하기 전에 동작을 수행할 수 있습니다. 예외 처리는 키워드 try, catch, finally를 사용합니다. 메소드는 throws 및 throw 키워드를 사용하여 예외를 선언할 수 있습니다.

Java에서 예외는 클래스 java.lang.Exception 또는 java.lang.Error의 하위 클래스가 될 수 있습니다. 메소드가 예외 발생을 선언하면 예외를 *발생시킨다(throw)*고 말합니다. 예외 *캐치(catch)*는 예외 처리를 의미합니다.

메소드 선언에 명시적으로 선언된 예외는 *처리되어야 하며* 그렇지 않은 경우 코드가 컴파일되지 않습니다. 메소드 선언에 명시적으로 선언되어 있지 않은 예외는 실행 시 프로그램을 중단시킬 수 있는데 컴파일되기는 합니다. 예외를 잘 처리하면 코드가 더 효율적이 된다는 것을 유의하십시오.

예외를 캐치하려면 try 블록에서 예외를 일으킬 수 있는 코드에 괄호를 표시한 다음 예외 처리에 사용하려는 코드를 catch 블록에서 괄호로 표시합니다. 예외가 발생하고 프로그램이 종료되어도 실행되어야 하는 중요한 코드(예: 지우기 코드)가 있는 경우 해당 코드를 마지막의 finally 블록에서 괄호로 표시합니다. 예를 들면, 다음과 같이 작동됩니다.

```
try {
  ... // Some code that might throw an exception goes here.
}
catch( Exception e ) {
  ... // Exception handling code goes here.
  // This next line outputs a stack trace of the exception:
     e.printStackTrace();
}
finally {
  ... // Code in here is guaranteed to be executed,
     // whether or not an exception is thrown in the try block.
}
```

try 블록은 처리해야 하는 예외를 내는 코드에 괄호를 표시할 때 사용되어야 합니다. 예외가 발생하지 않을 경우에 try 블록의 모든 코드가 실행됩니다. 그러나 예외가 발생하는 경우 try 블록의 코드는 예외가 발생하는 지점에서 실행 중단되며 제어 기능은 예외가 처리되는 catch 블록으로 넘어갑니다.

하나 이상의 catch 블록의 예외를 처리하기 위해 필요한 모든 방법을 수행할 수 있습니다. 예외를 처리하는 가장 간단한 방법은 모든 예외를 한 catch 블록에서 처리하는 것입니다. 이를 수행하려면 catch 뒤의 괄호에 있는 인수가 클래스 Exception과 그 뒤의 예외를 지정하는 변수 이름을 나타내야 합니다. 이 인수는 java.lang.Exception의 인스턴스인 예외 또는 이의 하위 클래스, 다른 말로하면 모든 예외가 캐치되었음을 나타내야 합니다.

예외 종류에 따라 다른 예외 처리 코드를 작성해야 하는 경우 둘 이상의 catch 블록을 사용할 수 있습니다. 이런 경우 catch 인수에서 예외 종류로서 Exception을 전달하는 대신에 캐치하고자 하는 특정 타입의 예외에 해당하는 클래스 이름을 나타내야 합니다. 이 이름은 Exception의 모든 하위 클래스일 수 있습니다. catch 블록은 항상 지정된 예외 타입 및 하위 클래스의 타입을 캐치한다는 것을 유의하십시오.

finally 블록의 코드는 try 블록의 코드가 어떤 이유로 인해 완료되지 않는 경우에도 실행이 보장됩니다. 예를 들어, try 블록의 코드는 코드가 예외를 낼 경우 완료되지 않을 수 있지만 finally 블록의 코드는 실행됩니다. 따라서 finally 블록은 지우기 코드를 삽입하기 좋은 위치입니다.

작성 중인 메소드를 다른 코드에서 호출하는 경우 메소드를 호출 코드에 놓아 메소드가 발생할 수도 있는 예외를 처리하도록 합니다. 이런 경우 메

소드가 예외를 낼 수 있음을 선언하기만 하면 됩니다. 예외를 내는 코드는 throws 키워드를 사용하여 예외를 선언할 수 있습니다. 이는 예외를 캐치하는 대안이 될 수 있는데 메소드가 예외 스로우를 선언하는 경우 예외를 처리할 필요가 없기 때문입니다.

예를 들면, 다음과 같이 throws를 사용합니다.

```
public void myMethod() throws SomeException {
  ... // Code here might throw SomeException, or one of its subclasses.
      // SomeException is assumed to be a subclass of Exception.
}
```

또한 throw 키워드를 사용하여 무언가 잘못되었음을 나타낼 수 있습니다. 예를 들어, 사용자가 잘못된 정보를 입력하여 오류 메시지를 표시하고 싶을 때 이 키워드를 사용하여 예외를 낼 수 있습니다. 이를 수행하려면 다음과 같은 문장을 사용하십시오.

```
throw new SomeException("invalid input");
```

Chapter

16

Java 클래스 라이브러리

대부분의 프로그래밍 언어는 기본 제공된 클래스 라이브러리에 의존하여 특정 기능을 지원합니다.자바 언어에서는 패키지라고 하는 관련된 이러한 클래스 그룹은 Java 에디션에 따라 다양합니다.각 에디션은 일반 애플리케이션, 기업용 애플리케이션, 소비재 제품 등과 같은 특정 목적에 따라 사용됩니다.

Java 2 Platform 에디션

Java 2 Platform은 다양한 목적으로 사용되는 여러 에디션에서 사용할 수 있습니다. Java는 모든 플랫폼 상의 어디서든 실행할 수 있는 언어이므로 인터넷, 인트라넷, 소비재 전자 제품 및 컴퓨터 애플리케이션과 같은 다양한 환경에서 사용됩니다. Java는 다양한 응용력을 갖추고 있으므로Java 2 Standard Edition(J2SE), Java 2 Enterprise Edition(J2EE), Java 2 Micro Edition(J2ME)과 같은 몇 개의 에디션 패키지로 제공됩니다.기업용 애플리케이션 개발에서와 같은 경우에는 더 큰 패키지들이 사용됩니다.소비재 전자 제품과 같은 그 밖의 경우에는 언어의 극히 일부분만 사용됩니다. 각 에디션에는 애플리케이션 개발에 사용되는 Java 2 Software Development Kit(SDK)와 애플리케이션 실행에 사용되는 Java 2 Runtime Environment(JRE)가 들어 있습니다.

Table 16.1 Java 2 Platform 에디션

Java 2 Platform	약어	설명
Standard Edition	J2SE	자바 언어의 핵심인 클래스를 포함합니다.
Enterprise Edition	J2EE	기업용 애플리케이션 개발을 위한 J2SE 클래스와 추가 클래스를 포함합니다.
Micro Edition	J2ME	J2SE 클래스의 서브셋을 포함하며 소비재 전자 제품에서 사용됩니다.

Standard Edition

Java 2 Platform, Standard Edition (J2SE)은 개발자에게 풍부한 기능을 가진 안정되고 안전한 크로스 플랫폼 개발 환경을 제공합니다.이 Java 에디션은 데이터베이스 연결, 사용자 인터페이스 디자인, 입력/출력, 네트워크 프로그래밍과 같은 핵심 기능을 제공하고 자바 언어의 기본 패키지를 포함합니다.

참조 사항 Java 2 Platform Standard Edition Overview
(http://java.sun.com/j2se/1.3/)
"Introducing the Java™ Platform"
(http://developer.java.sun.com/developer/onlineTraining/new2java/programming/intro/)
16- 3페이지 "Java 2 Standard Edition 패키지"

Enterprise Edition

Java 2, Enterprise Edition(J2EE)은 개발자에게 다중층 기업용 애플리케이션을 구축 및 배포할 수 있는 툴을 제공합니다. J2EE는 Enterprise JavaBeans 개발, Java 서블릿, JavaServer Pages, XML 및 유연한 트랜잭션 제어를 지원하는 추가 패키지뿐만 아니라 J2SE 패키지를 제공합니다.

참조 사항 Java 2 Platform Enterprise Edition Overview
(http://java.sun.com/j2ee/overview.html)
Java 2 Enterprise Edition 기술 관련 기사
(http://developer.java.sun.com/developer/technicalArticles/J2EE/index.html)

Micro Edition

Java 2, Micro Edition(J2ME)은 호출기, 스마트 카드, 휴대폰, 휴대용 PDA 및 셋톱 박스와 같은 다양한 소비재 전자 제품에 사용됩니다. J2ME는 Java 보다 작은 패키지들을 사용하지만 플랫폼 상에서의 코드 이식성, 장소에 제약 없이 실행할 수 있는 기능 및 안전 네트워크 전달과 같은 J2SE 및 J2EE와 동일한 자바 언어의 장점을 제공합니다. J2ME는 Micro Edition javax.microedition.io에 고유한 특정 추가 패키지와 J2SE 패키지의 서브셋을 제공합니다. 또한 J2ME 애플리케이션은 J2SE 및 J2EE를 사용할 수 있도록 상향 확장할 수 있습니다.

참조 사항 Java 2 Platform Micro Edition Overview
(http://java.sun.com/j2me/)
소비재 & Embeded 제품 기술 관련 기사
(http://developer.java.sun.com/developer/technicalArticles/ConsumerProducts/index.html)

Java 2 Standard Edition 패키지

Java 2 Platform인 Standard Edition(J2SE)은 데이터베이스 연결, 사용자 인터페이스 디자인, 입력/출력(I/O) 및 네트워크 프로그래밍 지원을 포함하는 매우 훌륭한 라이브러리를 제공합니다. 이 라이브러리들은 패키지라고 하는 관련 클래스 그룹으로 구성됩니다. 다음 표는 이러한 패키지 중 몇 가지를 간략히 설명한 것입니다.

Table 16.2 J2SE 패키지

패키지	Package Name	설명
Language	java.lang	자바 언어의 주요 핵심을 포함하는 클래스.
유틸리티	java.util	유틸리티 데이터 구조 지원.
I/O	java.io	다양한 입력/출력 타입 지원.
텍스트	java.text	지역화는 텍스트, 날짜, 숫자, 메시지에 대한 처리를 지원합니다.
Math	java.math	임의 정밀도 정수 및 부동 소수점 산술을 위한 클래스
AWT	java.awt	사용자 인터페이스 디자인 및 이벤트 처리.
Swing	javax.swing	모든 플랫폼에서 유사하게 동작하는 모든 Java, lightweight 컴포넌트를 위한 클래스.
Javax	javax	자바 언어에 대한 확장.
애플릿	java.applet	애플릿 생성에 필요한 클래스.
Beans	java.beans	JavaBeans 개발을 위한 클래스.
리플렉션	java.lang.reflect	런타임 클래스 정보를 얻는 데 사용되는 클래스.
SQL	java.sql	데이터베이스의 데이터 액세스 및 처리 지원.
RMI	java.rmi	분산 프로그래밍 지원.
네트워킹	java.net	네트워크 애플리케이션의 개발을 지원하는 클래스.
보안	java.security	암호 보안 지원.

참고 Java 패키지는 Java 2 Platform 에디션에 따라 다양합니다. Java 2 Software Development Kit(SDK)는 다양한 용도로 사용되는 Standard Edition(J2SE), Enterprise Edition(J2EE), 및 Micro Edition(J2ME)과 같은 몇 개의 에디션에서 사용할 수 있습니다.

참조 사항
16- 1페이지 "Java 2 Platform 에디션"
JDK API 사양에서 "Java 2 Platform, Standard Edition 및 API Specification"
Sun의 자습서 "Creating and using packages" (http://www.java.sun.com/docs/books/tutorial/java/interpack/packages.html)
*JBuilder를 이용한 애플리케이션 구축*의 "Creating and managing projects"에서 "패키지"

언어 패키지는 다음과 같습니다: java.lang

Java 클래스 라이브러리에서 가장 중요한 패키지 중 하나는 java.lang 패키지입니다. 모든 java 프로그램에 자동으로 import되는 이 패키지에는 java 프로그래밍 언어 디자인에 기초가 되는 언어의 주요 지원 클래스가 들어 있습니다.

참조 사항 JDK API Documentation에서 java.lang 패키지 요약
16- 9페이지 "주요 java.lang 클래스"

유틸리티 패키지는 다음과 같습니다: java.util

java.util 패키지에는 Java 개발에 중요한 다양한 유틸리티 클래스 및 인터페이스가 들어 있습니다. 이 패키지의 클래스는 컬렉션 프레임워크와 날짜 및 시간 기능을 지원합니다.

참조 사항 JDK API Documentation에서 java.util 패키지 요약
16- 15페이지 "주요 java.util 클래스"

입력/출력(I/O) 패키지는 다음과 같습니다: java.io

java.io 패키지는 다양한 장치에서의 데이터의 읽기 및 쓰기를 지원합니다. Java는 문자 입력/출력 스트림도 지원합니다. 또한 java.io 패키지의 File 클래스는 Non-UNIX 플랫폼을 보다 잘 지원하기 위해 추상적이고 시스템과 독립적인 파일 및 디렉토리 경로 이름의 표현을 사용합니다. 이 패키지의 클래스는 다음과 같은 입력 스트림 클래스, 출력 스트림 클래스, 파일 클래스 및 StreamTokenizer 클래스 그룹으로 나뉩니다.

참조 사항 JDK API Documentation에서 java.io 패키지 요약
16- 17페이지 "주요 java.io 클래스"

텍스트 패키지는 다음과 같습니다: java.text

java.text 패키지는 텍스트, 날짜, 숫자 및 메시지를 지원하기 위해 지역화를 지원하는 클래스 및 인터페이스를 제공합니다. NumberFormat, DateFormat 및 Collator와 같은 이 패키지의 클래스는 로케일 특정 방식으로 숫자, 날짜, 시간 및 문자열의 형식을 지정할 수 있습니다. 그밖의 클래스는 구문 분석, 검색 및 문자열 정렬을 지원합니다.

참조 사항 JDK API Documentation에서 java.text 패키지 요약
*JBuilder를 이용한 애플리케이션 구축*의 "Internationalizing programs with JBuilder"

Math 패키지는 다음과 같습니다: java.math

java.lang.Math 클래스와 혼동하기 쉬운 java.math 패키지는 임의 정밀도 정수 산술 연산(BigInteger)과 임의 정밀도 부동 소수점 산술 연산(BigDecimal)을 수행하기 위한 클래스를 제공합니다.

BigInteger 클래스는 임의로 대형 정수를 표현하기 위한 지원을 제공합니다.

BigDecimal 클래스는 통화 계산과 같이 소수점 지원을 필요로 하는 계산에서 사용되며 기본 산술, 배율 처리, 비교, 형식 변환 및 해싱 연산 또한 제공합니다.

참조 사항 JDK API Documentation에서 java.math 패키지 요약
JDK API Documentation에서 java.lang.Math
JDK Guide to Features에서 "Arbitrary-Precision Math"

AWT 패키지는 다음과 같습니다: java.awt

AWT(Abstract Window Toolkit) 패키지는 JFC(Java Foundation Classes)의 일부로서 GUI(Graphical User Interface) 프로그래밍에 대한 지원을 제공하며 사용자 인터페이스 컴포넌트, 이벤트 처리 모델, 레이아웃 관리자, 그래픽 및 이미징 툴, 잘라내기와 붙여넣기용 데이터 전송 클래스와 같은 기능을 제공합니다.

참조 사항 JDK API Documentation에서 java.awt 패키지 요약
JDK Guide to Features에서 "Abstract Window Toolkit (AWT)"
"AWT Fundamentals"
(http://developer.java.sun.com/developer/onlineTraining/awt/)
Chapter 24, "자습서:애플릿 만들기"
*JBuilder를 이용한 애플케이션 구축*에서 "Designing a user interface" 및 "Using layout manages"

Swing 패키지는 다음과 같습니다: javax.swing

javax.swing 패키지는 자동으로 모든 OS 플랫폼의 룩앤필을 표현하는 "lightweight"(완전 Java 언어) 컴포넌트 집합을 제공합니다. Swing 컴포넌트는 버튼, 스크롤바 및 레이블과 같은 기존 AWT 컴포넌트와 트리 뷰, 테이블 및 탭모양의 창과 같은 추가 컴포넌트를 제공하는 100% 완전 Java 버전입니다.

참고 javax 패키지는 핵심 자바 언어에 대한 확장입니다.

참조 사항 JDK API Documentation에서 java.swing 패키지 요약
JDK Guide to Features에서 "Project Swing(Java™ Foundation Classes) Software"
"Java Foundation Classes(JFC)"
(http://java.sun.com/docs/books/tutorial/post1.0/preview/jfc.html)

Sun의 Swing 자습서 "Trail: Creating a GUI with JFC/Swing"
(http://www.java.sun.com/docs/books/tutorial/uiswing/index.html)
JBuilder *자습서*의 "Building a Java text editor"
다음은 *JBuilder를 이용한 애플리케이션 구축*에서의 관련 단원입니다.

- "Designing a user interface"
- "Laying out your UI"
- "Using layout managers"
- "Using nested panels and layouts"

Javax 패키지는 다음과 같습니다: javax

다양한 javax 패키지들은 핵심 자바 언어에 대한 확장입니다. 이러한 패키지에는 javax.swing, javax.sound, javax.rmi, javax.transactions 및 javax.naming이 있습니다. 또한 개발자가 고유의 사용자 지정 javax 패키지를 작성할 수도 있습니다.

참조 사항 JDK API Documentation에서 javax.accessibility, javax.naming, javax.rmi, javax.sound.midi, javax.sound.sampled,javax.swing 및 javax.transaction 패키지 요약

애플릿 패키지는 다음과 같습니다: java.applet

java.applet 패키지는 애플릿이 일반적으로 웹 브라우저인 애플릿 컨텍스트와 통신하기 위해 사용하는 클래스 뿐만 아니라 애플릿을 생성하는데 필요한 클래스도 제공합니다. 애플릿은 독자적으로 실행되지 않고 다른 애플리케이션 내부에 포함되도록 디자인된 Java 프로그램입니다. 일반적으로 애플릿은 HTML 페이지에서 호출되는 인터넷/인트라넷 서버에 저장되어 다수의 클라이언트 플랫폼으로 다운로드됩니다. 그런 다음 클라이언트 플랫폼(시스템)의 브라우저에서 제공하는 JVM(Java Virtual Machine)에서 실행됩니다. 이러한 전달 및 실행 작업을 Security Manager에서 관리하므로 애플릿이 하드 드라이브를 포맷하거나 "무단" 시스템으로의 연결 개방과 같은 작업을 수행하지 못하게 할 수 있습니다.

보안 문제 및 브라우저 JDK 호환성과 관련이 있으므로 개발 전에 애플릿에 대해 충분히 이해해야 합니다. 애플릿은 보안상의 이유로 Java 프로그램의 모든 기능을 제공하지는 않습니다. 또한 애플릿은 브라우저의 이전 JDK 버전에 의존할 수도 있습니다. 이 설명서를 제작하는 시점에는 최신 JDK 버전을 완전히 지원하지 않는 브라우저들이 많이 있었습니다. 예를 들면, 대부분의 브라우저에서 Swing을 지원하지 않는 이전 JDk 버전을 지원합니다. 그러므로 Swing 컴포넌트를 사용하는 애플릿은 이러한 브라우저에서 실행되지 않습니다.

참조 사항 JDK API Documentation에서 java.applet 패키지 요약
Sun의 자습서, "Trail: Writing applets"
(http://www.java.sun.com/docs/books/tutorial/applet/index.html)
Chapter 20, "Java Virtual Machine 소개"
JBuilder의 *웹 애플리케이션 개발*의 "Working with applets"

Chapter 24, "자습서:애플릿 만들기"

Beans 패키지는 다음과 같습니다: java.beans

java.beans 패키지에는 JavaBeans 개발 관련 클래스가 들어 있습니다. JavaBeans는 독립적인 재사용할 수 있는 컴포넌트로 사용되는 Java 클래스로서 Java 플랫폼의 "한 번 작성하고 나면 장소에 상관 없이 실행할 수 있는" 기능을 재사용할 수 있는 컴포넌트 개발로 확장합니다. 재사용할 수 있는 코드 부분들은 프로그램 테스트 시 최소한의 변경만으로 처리 및 업데이트할 수 있습니다.

참조 사항 JDK API Documentation에서 java.beans 패키지 요약
JavaBeans™ Technology 기술 관련 기사
(http://developer.java.sun.com/developer/technicalArticles/jbeans/index.html)
*JBuilder를 이용한 애플리케이션 구축*에서 "Creating JavaBeans with BeansExpress"

리플렉션 패키지는 다음과 같습니다: java.lang.reflect

java.lang.reflect 패키지는 런타임에 클래스를 검사 및 처리하기 위한 클래스와 인터페이스를 제공합니다. 리플렉션을 통해 필드 및 메소드와 로드된 클래스의 생성자에 대한 정보에 액세스할 수 있습니다. Java 코드는 이 반영된 정보를 사용하여 객체의 카운터파트 상에서 작동할 수 있습니다.

이 패키지의 클래스는 디버거, 인터프리터, 객체 inspector, 클래스 브라우저와 같은 애플리케이션을 제공하며 Object Serialization 및 클래스에서 선언되는 멤버 또는 public 멤버에 대한 JavaBeans 액세스와 같은 서비스를 제공합니다.

참조 사항 JDK API Documentation에서 java.lang.reflect 패키지 요약
JDK Guide to Features에서 "Reflection"

SQL 패키지는 다음과 같습니다: java.sql

java.sql 패키지에는 데이터 소스의 데이터에 액세스하고 이를 처리하기 위해 필요한 API를 제공하는 클래스가 들어 있습니다. java.sql 패키지는 JDBC 2.0(Java Database Connectivity) API라고도 합니다. 이 API는 다양한 타입의 데이터 소스에 액세스할 수 있도록 다양한 드라이버를 동적으로 설치하기 위한 프레임워크를 포함합니다. JDBC는 Java 플랫폼에서 SQL(Structured Query Language)과 같은 다양한 언어로 작성된 데이터베이스까지도 포함하여 거의 모든 데이터베이스와 연결할 수 있도록 하는 업계 표준입니다.

java.sql 패키지에는 데이터베이스 연결 설정, 데이터베이스로 SQL 문 전송, 쿼리 결과 검색 및 업데이트, SQL 값 맵핑, 데이터베이스에 대한 정보

제공, 예외 쓰로우 및 보안 제공을 위한 클래스, 인터페이스 및 메소드가 들어 있습니다.

참조 사항 JDK API Documentation에서 java.sql 패키지 요약
Sun의 자습서, "Trail: JDBC(TM) Database Access"
(http://web2.java.sun.com/docs/books/tutorial/jdbc/index.html)
*JDataStore 개발자 안내서*에서 JBuilder의 "SQL reference"

RMI 패키지는 다음과 같습니다: java.rmi

java.rmi패키지는 RMI(Java Remote Method Invocation)용 클래스를 제공합니다. RMI를 사용하면 Java 프로그래밍 언어로 작성된 프로그램 간에 원격으로 통신할 수 있습니다. RMI는 JVM(Java Virtual Machine) 상에 있는 객체가 다른 JVM 상에 있는 객체에 대해 메소드를 호출할 수 있도록 하는 메커니즘입니다.

참조 사항 JDK API Documentation에서 java.rmi 패키지 요약
JDK Guide to Features에서 "Java Remote Method Invocation(RMI)"
"Java Remote Method Invocation - Distributed Computing for Java (백서)"
(http://java.sun.com/marketing/collateral/javarmi.html)
"Tutorial: JBuilder의 *Distributed Application Developer뮐 Guide*에서 "Exploring Java RMI-based distributed applications in JBuilder"
JBuilder 예제 RMI 애플리케이션은 다음과 같습니다. JBuilder 설치 디렉토리에서 samples/RMI/ 및 DataExpress/StreamableDataSets/

네트워킹 패키지는 다음과 같습니다: java.net

java.net 패키지에는 네트워킹 애플리케이션 개발용 클래스가 들어 있습니다. 소켓 클래스를 사용하면 인터넷 상에서 모든 서버와 통신하거나 고유 인터넷 서버를 구현할 수 있습니다. 또한 인터넷에서 검색되는 데이터용 클래스도 제공합니다.

참조 사항 JDK API Documentation에서 java.net 패키지 요약
JDK Guide to Features에서 "Networking Features"

Security 패키지는 다음과 같습니다: java.security

보안 패키지 java.security는 보안 프레임워크용 클래스 및 인터페이스를 정의합니다. 보안용 클래스에는 다음과 같은 두 종류가 있습니다.

- 액세스 제어를 구현하고 허가되지 않은 코드로 인한 중요 작업의 수행을 막는 클래스.
- 메시지 요약 및 디지털 시그너처를 구현하며 클래스 및 기타 객체들을 인증하는 인증 클래스.

이러한 클래스를 사용하면 개발자는 권한 및 보안 정책을 생성하여 애플리케이션, 빈, 서블릿을 비롯한 애플릿 및 Java 코드에 대한 액세스를 보호할 수 있습니다. 코드가 로드되면 보안 정책에 따라 코드에 권한이 할당됩니다. 권한은 읽기/쓰기 또는 연결 액세스와 같은 액세스할 수 있는 리소스를 지정합니다. 어떤 권한을 사용할 수 있는지를 제어하는 정책은 일반적으로 코드의 보안 정책을 정의하는 별도로 구성할 수 있는 정책에서 초기화됩니다. 권한 및 정책을 사용하면 코드에 대해 유연하고 구성 가능하며 확장할 수 있는 액세스 제어를 수행할 수 있습니다.

참조 사항 JDK API Documentation에서 java.security 패키지 요약
JDK Guide to Features에서 "Security"

주요 java.lang 클래스

Object 클래스는 다음과 같습니다: java.lang.Object

java.lang 패키지에 들어 있는 Object 클래스는 모든 Java 클래스의 부모 클래스(수퍼클래스)입니다. 간단히 말하면 Object 클래스가 클래스 계층에서 루트이며 모든 Java 클래스가 이 클래스로부터 파생된다는 뜻입니다. Object 클래스 자체에는 생성자 하나와 clone(), equals() 및 toString()을 비롯한 몇 개의 중요 메소드가 들어 있습니다.

메소드	인수	설명
clone	()	객체의 사본을 생성 및 반환합니다.
equals	(Object obj)	하나의 객체가 지정된 객체와 "같은지" 여부를 나타냅니다.
toString	()	객체의 문자열 표현을 반환합니다.

clone() 메소드를 사용하는 객체는 단순히 자신의 사본을 만듭니다. 사본이 생성되면 먼저 clone에 대해 새 메모리가 할당된 다음 원래 객체의 내용이 clone 객체에 복사됩니다. Document 클래스에서 Cloneable 인터페이스를 구현하는 다음 예에서는 clone() 메소드를 사용하여 text 및 author 속성을 가진 Document 클래스의 사본이 생성됩니다. Cloneable 인터페이스를 구현하지 않는 경우 객체가 clone할 수 있는 객체로 표시되지 않습니다.

```
Document document1 = new Document("docText.txt", "Joe Smith");
Document document2 = document1.clone();
```

equals() 메소드는 두 개 객체의 속성을 비교하는 방식으로 동등 연산자에 대해 동일한 타입의 두 개 객체를 비교합니다. 이 메소드는 단순히 이 메소드를 호출하는 객체의 결과와 메소드에 전달되는 객체에 따라 좌우되는 부울 값을 반환합니다. 예를 들어, 객체 하나가 equals() 메소드를 호출하여 다른 동일한 객체에게 전달하는 경우 equals() 메소드는 True 값을 반환합니다.

toString() 메소드는 객체를 표현하는 문자열을 반환합니다. 이 메소드를 사용하여 다양한 타입의 객체에 대한 올바른 정보를 반환하려면 객체의 클래스에서 이 메소드를 오버라이드해야 합니다.

참조 사항 JDK API Documentation에서 java.lang.Object

타입 랩퍼 클래스

성능상의 이유로 기본(primitive) 데이터 타입은 Java에서 객체로 사용되지 않습니다. 기본 데이터 타입에는 숫자, 부울 및 문자가 있습니다.

그러나 객체로서 기본 데이터 타입을 필요로 하는 Java 클래스 및 메소드도 있습니다 다음 표에 표시된대로 Java에서는 랩퍼 클래스를 사용하여 기본 타입을 객체로 랩핑 또는 캡슐화합니다.

기본 타입	설명	랩퍼
boolean	True 또는 False(1비트)	java.lang.Boolean
byte	-128 ~ 127(부호있는 8비트 정수)	java.lang.Byte
char	유니코드 문자(16비트)	java.lang.Character
double	+1.79769313486231579E+308 ~ +4.9406545841246544E-324(64비트)	java.lang.Double
float	+3.40282347E+28 ~ +1.40239846E-45(32비트)	java.lang.Float
int	-2147483648 ~ 2147483647(부호있는 32비트 정수)	java.lang.Integer
long	-9223372036854775808 ~ 9223372036854775807(부호있는 64비트 정수)	java.lang.Long
short	-32768 ~ 32767(부호있는 16비트 정수)	java.lang.Short
void	기본 Java 타입 void를 표현하는 Class 객체에 대한 참조를 저장할 인스턴스를 생성할 수 없는 위치 표시자 클래스.	java.lang.Void

Character(char value)와 같은 랩퍼 클래스의 생성자는 단순히 랩핑하고 있는 클래스 타입의 인수를 받습니다. 예를 들면, 다음 코드는 Character 랩퍼 클래스의 생성 방법을 보여 줍니다.

```
Character charWrapper = new Character('T');
```

이러한 각각의 클래스에 고유 메소드가 들어 있다 하더라도 몇몇 메소드는 각 객체 전체에서 표준으로 사용됩니다. 기본 타입을 반환하는 메소드로는 toString() 및 equals()가 있습니다.

각각의 랩퍼 클래스는 charValue()와 같은 메소드 하나를 제공하는데 이 메소드는 랩퍼 클래스의 기본 타입을 반환합니다. 다음 코드는 charWrapper 객체를 사용한 기본 타입의 반환을 보여 줍니다. charPrimitive는 char로 선언되는 기본 데이터 타입입니다. 이 메소드를 사용하면 타입 랩퍼에 기본 데이터 타입을 할당할 수 있습니다.

```
char charPrimitive = charWrapper.charValue();
```

toString() 및 equals() 메소드는 Object 클래스에서 비슷하게 사용됩니다.

Math 클래스는 다음과 같습니다: java.lang.Math

java.math 패키지와 혼동하기 쉬운 java.lang 패키지의 Math 클래스는 일반 산술 함수를 구현하는 메소드를 제공합니다. 이 클래스는 인스턴스화되지 않고 final로 선언되므로 하위 클래스로 분류될 수 없습니다. 이 클래스에 포함된 일부 메소드에는 sin(), cos(), exp(), log(), max(), min(), random(), sqrt(), 및 tan()이 있습니다. 다음 예는 이러한 메소드 중 몇 가지를 보여 줍니다.

```
double d1 = Math.sin(45);
double d2 = 23.4;
double d3 = Math.exp(d2);
double d4 = Math.log(d3);
double d5 = Math.max(d2, Math.pow(d1, 10));
```

이러한 메소드 중 일부는 다양한 데이터 타입을 사용 및 반환하도록 오버라이드됩니다.

또한 Math 클래스는 PI 및 E 상수를 선언합니다.

참고 java.lang.Math와는 달리 java.math 패키지는 큰 수를 임의로 사용하기 위한 클래스 지원을 제공합니다.

참조 사항 JDK API Documentation에서 java.lang.Math
JDK API Documentation에서 java.math 패키지 요약

String 클래스는 다음과 같습니다: java.lang.String

java.lang 패키지의 String 클래스는 문자열을 표현하는 데 사용됩니다. C/C++와는 달리 Java는 문자 배열을 사용하여 문자열을 표현하지 않습니다. 문자열은 상수로서 생성되고 나면 값을 변경할 수 없습니다. String 클래스는 일반적으로 Java 컴파일러에서 따옴표 안에 들어 있는 문자열을 접할 때 생성됩니다. 그러나 몇 가지 방법으로 문자열은 생성할 수 있습니다. 다음 테이블은 String의 생성자 및 이 클래스에서 사용하는 인수 중 몇 개를 나열한 것입니다.

생성자	인수	설명
String	()	새 String 객체를 초기화합니다.
String	(String value)	새 String 객체를 String 인수를 사용하여 초기화합니다.
String	(char[] value)	동일한 순서의 배열을 포함하는 새 String을 생성합니다.
String	(char[] value, int offset, int count)	인수의 부분 배열을 포함하는 String을 생성합니다.
String	(StringBuffer buffer)	새 String 객체를 StringBuffer 인수의 내용을 사용하여 초기화합니다.

String 클래스에는 문자열을 처리할 때 필수적인 몇 가지 중요한 메소드가 들어 있습니다. 이러한 메서드는 문자열을 편집, 비교 및 분석하는 데 사용됩니다. 문자열은 변경될 수 없으므로 어떤 메소드도 문자열 순서를 변경할 수 없습니다. 다음 단원에서 설명할 StringBuffer 클래스는 문자열 변경 메소드를 제공합니다.

다음 테이블은 매우 중요한 몇 개의 메소드를 나열하고 이러한 메소드에서 사용 및 반환하는 인수와 값을 선언합니다.

메소드	인수	반환	설명
length	()	int	문자열에 포함된 문자의 수를 반환합니다.
charAt	(int index)	char	문자열의 지정된 인덱스에 있는 문자를 반환합니다.
compareTo	(String value)	int	문자열과 인수 문자열을 비교합니다.
indexOf	(int ch)	int	지정된 문자를 처음 접하게 되는 인덱스 위치를 반환합니다.
substring	(int beginIndex, int endIndex)	String	문자열의 부분 문자열인 새 문자열을 반환합니다.
concat	(String str)	String	이 문자열의 끝에 지정된 String을 연결합니다.
toLowerCase	()	String	문자열을 소문자로 반환합니다.
toUpperCase	()	String	문자열을 대문자로 반환합니다.
valueOf	(Object obj)	String	Object 인수의 문자열 표현을 반환합니다.

많은 메소드와 관련된 매우 효율적인 기능 중 하나는 보다 나은 유연성을 위해 이러한 메소드가 오버로드된다는 점입니다. 다음은 String 클래스 및 이 클래스의 일부 메소드의 사용법을 보여 줍니다.

```
String s1 = new String("Hello World.");

char cArray[] = {'J', 'B', 'u', 'i', 'l', 'd', 'e', 'r'};
String s2 = new String(cArray);          //s2 = "JBuilder"

int i = s1.length();                    //i = 12
char c = s1.charAt(6);                    //c = 'W'
i = s1.indexOf('e');                     //i = 1 (index of 'e' in "Hello World.")

String s3 = "abcdef".substring(2, 5);    //s3 = "cde"
String s4 = s3.concat("f");                //s4 = "cdef"
String s5 = String.valueOf(i);             //s5 = "1" (valueOf() is static)
```

참고 배열과 String 인덱스는 0에서 시작합니다.

참조 사항 JDK API Documentation에서 java.lang.String

StringBuffer 클래스는 다음과 같습니다: java.lang.StringBuffer

String 클래스와 마찬가지로 java.lang 패키지의 StringBuffer 클래스는 문자열 순서를 표현합니다. 문자열과는 달리 StringBuffer의 내용은 수정될 수 있습니다. 다양한 StringBuffer 메소드를 사용하면 문자열 버퍼의 길이 및 내용을 변경할 수 있습니다. 또한 필요할 경우 StringBuffer 객체의 길이를 증가시킬 수 있습니다. 마지막으로 StringBuffer를 수정한 후 StringBuffer의 내용을 표현하는 새 문자열을 생성할 수 있습니다.

StringBuffer 클래스에는 다음 표에 표시된대로 몇 개의 생성자가 있습니다.

생성자	인수	설명
StringBuffer	()	최대 16개의 문자를 저장할 수 있는 빈 문자열 버퍼를 생성합니다.
StringBuffer	(int length)	length를 통해 지정한 문자의 수를 저장할 수 있는 빈 문자열 버퍼를 생성합니다.
StringBuffer	(String str)	String str의 사본을 포함하는 문자열 버퍼를 생성합니다.

다음과 같은 몇 개의 중요한 메소드는 String 클래스에서 StringBuffer 클래스를 분리합니다. capacity(), setLength(), setCharAt(), append(), insert() 및 toString(). append() 및 insert() 메소드는 다양한 데이터 타입을 사용하도록 오버로드됩니다.

메소드	인수	설명
setLength	(int newLength)	Stringbuffer의 길이를 설정합니다.
capacity	()	StringBuffer에 할당된 메모리의 양을 반환합니다.
setCharAt	(int index, char ch)	StringBuffer의 지정된 인덱스에 있는 문자를 ch로 설정합니다.
append	(char c)	인수에 지정된 데이터 타입의 문자열 표현을 StringBuffer에 추가합니다.이 메소드는 다양한 데이터 타입을 사용하도록 오버로드됩니다.
insert	(int offset, char c)	인수에 지정된 데이터 타입의 문자열 표현을 StringBuffer에 삽입합니다.이 메소드는 다양한 데이터 타입을 사용하도록 오버로드됩니다.
toString	()	StringBuffer를 String으로 변환합니다.

capacity() 메소드는 StringBuffer에 할당된 메모리의 양을 반환하는 메소드로서 length() 메소드보다 더 큰 값을 반환할 수 있습니다.
StringBuffer(int length) 생성자를 사용하면 StringBuffer에 할당된 메모리를 설정할 수 있습니다.

다음 코드는 StringBuffer 클래스와 관련된 일부 메소드를 보여 줍니다.

```
StringBuffer s1 = new StringBuffer(10);
```

```
int c = s1.capacity();        //c = 10
int len = s1.length();        //len = 0

s1.append("Bor");             //s1 = "Bor"
s1.append("land");            //s1 = "Borland"

c = s1.capacity();            //c = 10
len = s1.length();            //len = 7

s1.setLength(2);              //s1 = "Bo"

StringBuffer s2 = new StringBuffer("Helo World");
s2.insert(3, "l");            //s2 = "Hello World"
```

참조 사항 JDK API Documentation에서 java.lang.StringBuffer

System 클래스는 다음과 같습니다: java.lang.System

java.lang 패키지의 System 클래스에는 플랫폼과 무관한 시스템 리소스와 정보에 액세스하고, 배열을 복사하며, 파일 및 라이브러리를 로드하고, 속성을 가져오고 설정하기 위한 몇 개의 유용한 클래스 필드 및 메소드가 들어 있습니다. 예를 들면, currentTimeMillis() 메소드는 현재 시스템 시간 액세스를 제공합니다. 또한 getProperty 및 setProperty 메소드를 사용하여 시스템 리소스를 검색 및 변경할 수도 있습니다. System 클래스에서 제공하는 또 하나의 편리한 기능은 gc() 메소드로서 이 메소드는 가비지 컬렉터에서 가비지 컬렉션을 통해 수행하는 요청입니다. 마지막으로 System 클래스를 통해서 개발자는 loadLibrary() 메소드를 사용하여 동적 링크 라이브러리를 로드할 수 있습니다.

System 클래스는 final 클래스로 선언되므로 하위 클래스로 분류될 수 없습니다. 또한 이 클래스는 메소드 및 변수를 static으로 선언합니다. 따라서 클래스의 인스턴스를 생성하지 않고도 메소드 및 변수를 사용할 수 있습니다.

또한 System 클래스는 시스템과 상호 작용하는 데 사용되는 몇 개의 변수를 선언합니다. 이러한 변수에는 in, out 및 err이 있습니다. in 변수는 시스템의 표준 입력 스트림을 표현하는 데 반해, out 변수는 표준 출력 스트림을 표현합니다. err 변수는 표준 오류 스트림입니다. 스트림은 인력/출력(I/O) 패키지 단원에서 자세히 설명합니다.

메소드	인수	설명
arrayCopy	arraycopy(Object src, int src_position, Object dst, int dst_position, int length)	지정된 위치에서 시작하는 지정된 소스 배열을 대상 배열의 지정된 위치로 복사합니다.
currentTimeMillis	()	현재 시간을 밀리초 단위로 반환합니다.
loadLibrary	(String libname)	인수를 통해 지정한 시스템 라이브러리를 로드합니다.

메소드	인수	설명
getProperty	(String key)	키로 표현되는 시스템 속성을 가져옵니다.
gc	()	더 이상 사용하지 않은 객체를 삭제하는 가비지 컬렉터를 실행합니다.
load	(String filename)	로컬 파일 시스템의 코드 파일을 동적 라이브러리로서 로드합니다.
exit	(int status)	현재 프로그램을 종료합니다.
setProperty	(String key, String value)	키에서 지정한 시스템 속성을 설정합니다.

참조 사항 JDK API Documentation에서 java.lang.System

주요 java.util 클래스

Enumeration 인터페이스는 다음과 같습니다: java.util.Enumeration

java.util 패키지의 Enumeration 인터페이스는 값을 열거할 수 있는 클래스를 구현하는 데 사용됩니다. Enumeration 인터페이스를 구현하는 클래스를 사용하면 쉽게 데이터 구조의 순회할 수 있습니다.

Enumeration 인터페이스에 정의된 메서드를 사용하면 Enumeration 객체는 값 집합에서 모든 요소를 하나씩 연속으로 검색할 수 있습니다. Enumeration 인터페이스에 선언되는 메소드는 hasMoreElements()와 nextElement()뿐입니다.

hasMoreElements() 메소드는 데이터 구조 안에 다른 요소들이 남아 있는 경우 True를 반환합니다. nextElement() 메소드는 열거되는 구조에서 다음 값을 반환합니다.

다음 예에서는 Enumeration 인터페이스를 구현하는 CanEnumerate 클래스를 생성합니다. 해당 클래스의 인스턴스는 Vector 객체 v의 모든 요소를 인쇄하는 데 사용됩니다.

```
CanEnumerate enum = v.elements();

while (enum.hasMoreElements()) {
  System.out.println(enum.nextElement());
}
```

Enumeration 객체에는 한 가지 제한이 있는 데 한 번 밖에 사용할 수 없다는 점입니다. Enumeration 객체에서 이전 요소들을 역추적할 수 있게 해주는 인터페이스에는 메소드가 정의되어 있지 않습니다. 그러므로 전체 값 집합을 열거하고 나면 소멸됩니다.

참조 사항 JDK API Documentation에서 java.util.Enumeration

Vector 클래스는 다음과 같습니다: java.util.Vector

Java는 모든 동적 데이터 구조를 지원하지 않으므로 Stack 클래스만을 정의합니다. 그러나 java.util 패키지의 Vector 클래스는 쉽게 동적 데이터 구조를 구현할 수 있는 방법을 제공합니다.

Vector 클래스는 새 요소를 추가할 때 필요한 것보다 더 많은 메모리를 할당하므로 효율적입니다 따라서 Vector의 용량은 대개 실제 크기보다 큽니다. 다음 테이블에서 표시하는대로 네 번째 생성자의 capacityIncrement 인수는 요소가 추가될 때마다 증가하는 Vector의 용량을 정의합니다.

생성자	인수	설명
Vector	()	배열 크기 10과 용량 증분이 0인 빈 벡터를 생성합니다.
Vector	(Collection c)	컬렉션의 반복자를 통해 반환되는 순서대로 컬렉션 요소를 포함하는 벡터를 생성합니다.
Vector	(int initialCapacity)	지정된 초기 용량을 갖고 용량 증분이 0인 빈 벡터를 생성합니다.
Vector	(int initialCapacity, int capacityIncrement)	지정된 초기 용량과 용량 증분을 갖는 빈 벡터를 생성합니다.

다음 테이블은 Vector 클래스의 중요 메소드와 이러한 메소드에서 사용하는 인수들 중 몇 개를 나열한 것입니다.

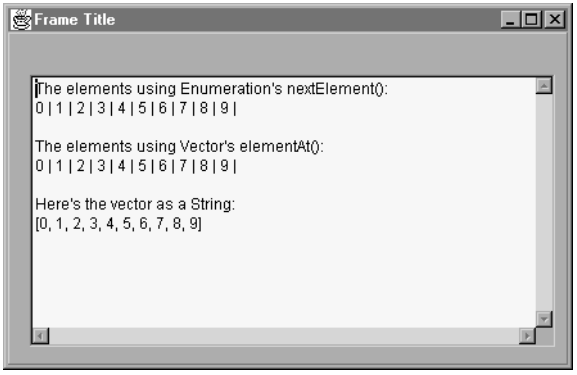
메소드	인수	설명
setSize	(int newSize)	벡터의 크기를 설정합니다.
capacity	()	벡터의 용량을 반환합니다.
size	()	벡터에 저장된 요소의 수를 반환합니다.
elements	()	벡터 요소의 열거를 반환합니다.
elementAt	(int)	지정된 인덱스에 있는 요소를 반환합니다.
firstElement	()	인덱스 0에 있는 벡터의 첫 번째 요소를 반환합니다.
lastElement	()	벡터의 마지막 요소를 반환합니다.
removeElementAt	(int index)	지정된 인덱스에서 요소를 제거합니다.
addElement	(Object obj)	크기를 1만큼 증가시키며 벡터 끝에 지정된 객체를 추가합니다.
toString	()	벡터에서 각 요소의 문자열 표현을 반환합니다.

다음 코드는 Vector 클래스의 사용을 보여 줍니다. vector1이라는 Vector 객체가 생성되고 나면 이 객체는 Enumeration의 nextElement() 메소드 사용, Vector의 elementAt() 메소드 사용 및 Vector의 toString() 메소드 사용이라는 세 가지 방법으로 요소를 열거합니다. 출력을 표시할 AWT 컴포넌트 textArea가 생성됩니다. 텍스트 속성은 setText() 메소드를 사용하여 설정됩니다.

```
Vector vector1 = new Vector();

  for (int i = 0; i <= 10; i++) {
  vector1.addElement(new Integer(i)); //addElement accepts object or
composite types
}                                   //but not primitive types

//enumerate vector1 using nextElement()
Enumeration e = vector1.elements();
textArea1.setText("The elements using Enumeration's nextElement():₩n");
        while (e.hasMoreElements()) {
  textArea1.append(e.nextElement()+ " | ");
}
textArea1.append("₩n₩n");

//enumerate using the elementAt() method
textArea1.append("The elements using Vector's elementAt():₩n");
for (int i = 0; i < vector1.size();i++) {
  textArea1.append(vector1.elementAt(i) + " | ");
}
textArea1.append("₩n₩n");

//enumerate using the toString() method
textArea1.append("Here's the vector as a String:₩n");
textArea1.append(vector1.toString());
```

다음 그림은 이 코드를 애플리케이션에서 사용할 경우 코드에서 수행하는 작업을 보여 줍니다.

Figure 16.1 Vector 및 Enumeration 예제



참조 사항 JDK API Documentation에서 java.util.Vector

주요 java.io 클래스

입력 스트림 클래스

입력 스트림은 파일, 문자열 또는 메모리와 같은 입력 소스에서 데이터를 읽는 데 사용됩니다. java.io 패키지에 들어 있는 입력 스트림 클래스의 예

로는 InputStream, BufferedInputStream, DataInputStream 및 FileInputStream이 있습니다.

입력 스트림 클래스를 사용하여 데이터를 읽는 기본적인 방법은 항상 다음과 같습니다.

1 입력 스트림 클래스의 인스턴스를 생성합니다.

2 데이터를 읽어올 소스를 클래스에 알려줍니다.

참고 입력 스트림 클래스는 데이터를 바이트의 연속 스트림으로 읽습니다. 현재 사용할 수 있는 데이터가 없는 경우 입력 스트림 클래스는 데이터를 사용할 수 있을 때까지 차단하거나 기다립니다.

입력 스트림 클래스 외에도 java.io 패키지는 DataInputStream의 경우를 제외하고는 리더 클래스를 제공합니다. 리더 클래스의 예로는 Reader, BufferedReader, FileReader 및 StringReader가 있습니다. 리더 클래스는 바이트 대신에 유니코드 문자를 읽는다는 점을 제외하고는 입력 스트림 클래스와 동일합니다.

InputStream 클래스는 다음과 같습니다 : java.io.InputStream

java.io 패키지에 들어 있는 InputStream 클래스는 다른 모든 입력 스트림 클래스의 추상 클래스이자 수퍼클래스입니다. 이 클래스는 바이트 스트림을 읽기 위한 기본 인터페이스를 제공합니다. 다음 표는 InputStream 클래스에 정의된 일부 메소드와 이들 메소드가 사용한 인수를 나열한 것입니다. close() 메소드를 제외한 나머지 각 메소드는 int 값을 반환합니다.

메소드	인수	설명
read	()	입력 스트림에서 다음 바이트를 읽고 정수로 반환합니다. 스트림의 끝에 도달하면 -1을 반환합니다.
read	(byte b[])	다수의 바이트를 읽고 배열 b에 저장합니다. 읽은 바이트의 수를 반환하거나 스트림의 끝에 도달하는 경우 -1을 반환합니다.
read	(byte b[], int off, int len)	오프셋 off에서 시작하는 최대 len 바이트의 데이터를 입력 스트림에서 배열로 읽어 들입니다.
available	()	입력 스트림용 메소드의 다음 호출자로부터 방해를 받지 않는 상태로 입력 스트림에서 읽을 수 있는 바이트의 수를 반환합니다.
skip	(long n)	입력 스트림에서 데이터의 n바이트를 건너뛰거나 무시합니다.
close	()	입력 스트림을 닫고 스트림에서 사용한 시스템 리소스를 해제합니다.

참조 사항 JDK API Documentation에서 java.io.InputStream

FileInputStream 클래스는 다음과 같습니다 : java.io.FileInputStream

java.io 패키지의 FileInputStream 클래스는 InputStream 클래스와 매우 유사하나 특별히 파일 읽기용으로 디자인되었습니다. 이 클래스에는 FileInputStream(String filename), FileInputStream(File fileobject) 및 FileInputStream(FileDescriptor fdObj)이라는 세 개의 생성자가 있습니다. 첫 번째 생성자는 파일 이름을 인수로 받지만 두 번째 생성자는 단순히 파일 객체를 받습니다. 세 번째 생성자는 파일 디스크립터 객체를 받습니다. "파일 클래스"는 16- 23페이지에서 설명합니다.

생성자	인수	설명
FileInputStream	(String filename)	파일 시스템에서 경로 이름 filename 이 이름 지정한 파일을 연결하여 FileInputStream을 생성합니다.
FileInputStream	(File fileobject)	파일 시스템에서 fileobject 파일이 이름 지정한 파일을 연결하여 FileInputStream을 생성합니다.
FileInputStream	(FileDescriptor fdObj)	파일 시스템에서 실제 파일과의 기존 연결을 표현하는 파일 디스크립터 fdObj를 사용하여 FileInputStream을 생성합니다.

다음 예는 FileInputStream 클래스의 사용을 보여 줍니다.

```
import java.io.*;

class FileReader {
  static void main(String args[]) {
  byte buff[] = new byte[80];
  try {
   InputStream fileIn = new FileInputStream("Readme.txt");
   int i = fileIn.read(buff);
   String s = new String(buff);
  System.out.println("Looping");
  }
  catch(FileNotFoundException e) {
  }
    catch(Exception e) {
  }
 }
}
```

이 예에서는 입력 데이터를 저장하는 문자 배열이 생성됩니다. 그리고 나서 FileInputStream 객체는 인스턴스화되고 입력 파일의 이름이 생성자에게 전달됩니다. 그런 다음 FileInputStream read() 메소드를 사용하여 문자 스트림을 읽은 후 buff 배열에 저장합니다. 처음 80바이트를 Readme.txt 파일에서 읽어온 후 buff 배열에 저장합니다.

참고 또한 FileReader 클래스를 FileInputStream() 메소드 대신에 사용할 수도 있습니다. 이 경우 byte 배열을 char 배열로 변경만 하면 됩니다. 그러면 다음과 같이 reader 객체가 인스턴스화됩니다.

```
Reader fileIn = new FileReader("Readme.txt");
```

마지막으로 read 호출의 결과를 확인하기 위해 buff 배열을 사용하여 String 객체를 생성한 다음 System.out.println() 메소드에 전달합니다.

앞에서 설명한대로 java.lang에 정의된 System 클래스는 시스템 리소스 액세스를 제공합니다. System.out은 System의 static 멤버로서 표준 출력 장치를 나타냅니다. println() 메소드는 출력을 표준 출력 장치에 전송할 때 호출됩니다. System.out 객체는 PrintStream 타입의 객체이며 이 객체에 관해서는 16- 20페이지 "출력 스트림 클래스"에서 설명합니다.

System.in 객체는 System 클래스의 또 다른 static 멤버로서 InputStream 타입이고 표준 입력 장치를 나타냅니다.

참조 사항 JDK API Documentation에서 java.io.FileInputStream

출력 스트림 클래스

출력 스트림 클래스는 입력 스트림 클래스와 상반되는 클래스입니다. 출력 스트림 클래스는 다양한 출력 소스로 데이터의 스트림을 출력하는 데 사용됩니다. java.io 패키지에 있는 Java 주요 출력 스트림 클래스로는 OutputStream, PrintStream, BufferedOutputStream, DataOutputStream 및 FileOutputStream가 있습니다.

OutputStream 클래스는 다음과 같습니다 : java.io.OutputStream

데이터의 스트림을 출력하기 위해 OutputStream 객체가 생성되고 나면 그 객체는 특정 출력 소스로 데이터를 출력하라는 지시를 받습니다. 예상대로 DataOutputStream 클래스를 제외하고 각 클래스에 해당하는 라이터 클래스가 있습니다. OutputStream 클래스는 다음과 같은 메소드를 정의합니다.

메소드	인수	설명
write	(int b)	b를 출력 스트림에 씁니다.
write	(byte b[])	배열 b를 출력 스트림에 씁니다.
write	(byte b[], int off, int len)	오프셋 off에서 시작하는 바이트 배열로부터 len 바이트를 출력 스트림에 씁니다.
flush	()	출력 스트림을 플러시하고 버퍼에 저장된 데이터 출력을 강요합니다.
close	()	출력 스트림을 닫고 출력 스트림과 관련된 시스템 리소스를 모두 해제합니다.

참조 사항 JDK API Documentation에서 java.io.OutputStream

PrintStream 클래스는 다음과 같습니다 : java.io.PrintStream

java.io 패키지의 PrintStream 클래스는 기본적으로 데이터를 텍스트로 출력하도록 디자인되었으며 두 개의 생성자를 가지고 있습니다. 첫 번째 생성자는 지정된 조건을 기초로 하여 버퍼에 저장된 데이터를 플러시하는 반면, 두 번째 생성자는 autoflush가 true로 설정되는 경우 새 줄 문자를 접할 때 데이터를 플러시합니다.

생성자	인수	설명
PrintStream	(OutputStream out)	새로운 인쇄 스트림을 만듭니다.
PrintStream	(OutputStream out, boolean autoflush)	새로운 인쇄 스트림을 만듭니다.

다음 표는 PrintStream 클래스에 정의된 메소드 중 몇 개를 보여 줍니다.

메소드	인수	설명
checkError	()	스트림을 플러시한 후 오류가 발견되면 False 값을 반환합니다.
print	(Object obj)	객체를 인쇄합니다.
print	(String s)	문자열을 인쇄합니다.
println	()	시스템 속성 line.separator에서 정의한 줄 구분 문자열을 사용하여 줄을 인쇄하고 종료합니다. 이 때 줄 구분 문자열이 반드시 새줄 문자(□ n□ 일 필요는 없습니다.
println	(Object obj)	객체를 인쇄하고 줄을 종료합니다. 이 메소드는 print(Object) 이후에 println()을 호출하는 것처럼 동작합니다.

print() 및 println() 메소드는 다양한 데이터 타입을 받을 수 있도록 오버로드됩니다.

참조 사항 JDK API Documentation에서 java.io.PrintStream

BufferedOutputStream 클래스는 다음과 같습니다 : java.io.BufferedOutputStream

java.io 패키지의 BufferedOutputStream 클래스는 버퍼에 값을 저장한 후 버퍼가 다 차거나 flush() 메소드가 호출될 때만 값을 씀으로써 출력 효율성을 증가시키는 버퍼 출력 스트림을 구현합니다.

생성자	인수	설명
BufferedOutputStream	(OutputStream out)	데이터를 출력 스트림에 쓸 새로운 512바이트 버퍼 출력 스트림을 생성합니다.
BufferedOutputStream	(OutputStream out, int size)	지정된 버퍼 크기를 가진 출력 스트림에 데이터를 쓸 새 버퍼 출력 스트림을 생성합니다.

BufferedOutputStream은 데이터를 플러시하고 출력 스트림에 쓰기 위한 세 개의 메소드를 제공합니다.

메소드	인수	설명
flush	()	버퍼 출력 스트림을 플러시합니다.
write	(byte[] b, int off, int len)	오프셋 off에서 시작하는 바이트 배열로부터 len 바이트를 버퍼 출력 스트림에 씁니다.
write	(int b)	바이트를 버퍼 출력 스트림에 씁니다.

참조 사항 JDK API Documentation에서 java.io.BufferedOutputStream

DataOutputStream 클래스는 다음과 같습니다 : java.io.DataOutputStream

데이터 출력 스트림을 사용하면 애플리케이션에서 이식 가능한 바이너리 형식으로 출력 스트림에 기본(primitive) Java 데이터 타입을 쓸 수 있습니다. 그리고 나면 애플리케이션은 데이터 입력 스트림을 사용하여 데이터를 다시 읽을 수 있습니다.

java.io 패키지의 DataOutputStream 클래스에는 DataOutputStream(OutputStream out)이라는 하나의 생성자가 있습니다. 이 생성자는 데이터를 출력 스트림에 쓰는 데 사용되는 새 데이터 출력 스트림을 생성합니다.

DataOutputStream 클래스는 다양한 write() 메소드와 flush() 및 size() 메소드를 사용하여 기본(primitive) 데이터 타입을 출력합니다.

메소드	인수	설명
flush	()	데이터 출력 스트림을 플러시합니다.
size	()	데이터 출력 스트림의 바이트 수를 반환합니다.
write	(int b)	출력 스트림에 바이트를 씁니다.
writeType	(type v)	출력 스트림에 지정된 기본(primitive) 타입을 바이트로 씁니다.

참조 사항 JDK API Documentation에서 java.io.DataOutputStream

FileOutputStream 클래스는 다음과 같습니다 : java.io.FileOutputStream

파일 출력 스트림은 파일 또는 FileDescriptor에 데이터를 쓰기 위한 출력 스트림입니다. 파일의 사용 또는 생성 가능 여부는 기본 플랫폼에 따라 다릅니다. 일부 플랫폼에서는 파일을 열여서 한 번에 FileOutputStream을 한 개씩만 쓸 수 있습니다. 이러한 상황에서는 호출되는 파일이 이미 열려 있는 경우 클래스의 생성자가 수행되지 못합니다. java.io 패키지의

FileOutputStream은 OutputStream의 하위 클래스로서 몇 개의 생성자를 사용합니다.

생성자	인수	설명
FileOutputStream	(File file)	지정된 파일에 쓸 파일 출력 스트림을 생성합니다.
FileOutputStream	(FileDescriptor fdObj)	파일 디스크립터에 쓸 출력 파일 스트림을 생성합니다. 출력 파일 스트림은 파일 시스템에서 실제 파일에 대한 기존 연결을 표현합니다.
FileOutputStream	(String name)	지정된 이름을 가진 파일에 쓰려면 출력 파일 스트림을 만듭니다.
FileOutputStream	(String name, boolean append)	지정된 이름을 가진 파일에 쓰려면 출력 파일 스트림을 만듭니다.

FileOutputStream은 close(), finalize() 및 몇 개의 write() 메소드를 비롯하여 몇 개의 메소드를 제공합니다.

메소드	인수	설명
close	()	파일 출력 스트림을 닫고 관련 시스템 리소스를 해제합니다.
finalize	()	파일과의 연결을 정리하고 스트림에 대한 참조가 없을 때 close() 메소드를 호출합니다.
getFD	()	스트림과 관련된 파일 디스크립터를 반환합니다.
write	(byte[] b, int off, int len)	오프셋 off에서 시작하는 바이트 배열로부터 len 바이트를 파일 출력 스트림에 씁니다.

참조 사항 JDK API Documentation에서 java.io.FileOutputStream

파일 클래스

java.io 패키지의 FileInputStream 및 FileOutputStream 클래스만이 파일 입/출력을 처리하는 기본 함수를 제공합니다. java.io 패키지는 향상된 파일 지원을 위해 File 클래스 및 RandomAccessFile 클래스를 제공합니다. File 클래스는 파일 속성 및 함수에 쉽게 액세스 할 수 있도록 해주며 RandomAccessFile 클래스는 파일에서 읽고 파일에 쓰기 위한 다양한 메소드를 제공합니다.

File 클래스는 다음과 같습니다 : java.io.File

Java의 File 클래스는 파일 및 디렉토리 경로 이름에 대해 플랫폼과 무관한 추상적 표현을 사용합니다. 다음 표는 File 클래스의 세 가지 생성자를 나열한 것입니다.

생성자	인수	설명
File	(String path)	지정된 경로 이름 문자열을 추상 경로 이름으로 변환하여 새 File 인스턴스를 생성합니다.
File	(String parent, String child)	부모 경로 이름 문자열과 자식 경로 이름 문자열에서 새 File 인스턴스를 생성합니다.
File	(File parent, String child)	부모 추상 경로 이름 및 자식 경로 이름 문자열에서 새 File 인스턴스를 생성합니다.

또한 File 클래스는 존재 여부, 판독성, 기록성, 타입, 크기, 파일 및 디렉토리 수정 시간을 검사하며, 새 디렉토리 생성, 파일과 디렉토리의 이름 바꾸기 및 삭제 작업을 수행하는 많은 중요한 메소드를 구현합니다.

메소드	인수	반환	설명
delete	()	boolean	파일 또는 디렉토리를 삭제합니다.
canRead	()	boolean	애플리케이션에서 추상 경로 이름으로 표시되는 파일을 읽을 수 있는지 여부를 테스트합니다.
canWrite	()	boolean	애플리케이션에서 해당 파일에 쓸 수 있는지 여부를 테스트합니다.
renameTo	(File dest)	boolean	파일의 이름을 바꿉니다.
getName	()	String	파일 또는 디렉토리의 이름 문자열을 반환합니다.
getParent	()	String	파일 또는 디렉토리의 부모 디렉토리에 대한 경로 이름 문자열을 반환합니다.
getPath	()	String	추상 경로 이름을 경로 이름 문자열로 변환합니다.

참조 사항 JDK API Documentation에서 java.io.File

RandomAccessFile 클래스은 다음과 같습니다 : java.io.RandomAccessFile

java.io 패키지의 RandomAccessFile 클래스는 파일에 대한 순차 액세스만을 제공하는 FileInputStream 및 FileOutputStream 클래스보다 훨씬 뛰어납니다. RandomAccessFile 클래스를 통해서 파일의 지정된 위치에서 임의의 바이트, 텍스트 및 Java 데이터 타입을 읽고 지정된 위치에 쓸 수 있습니다. 다음과 같이 RandomAccessFile(String name, String mode)

와 RandomAccessFile(File file, String mode)라는 두 개의 생성자가 있습니다 mode 인수는 RandomAccessFile 객체가 읽기("r")용으로 사용되는지 읽기/쓰기("rw")용으로 사용되는 지를 나타냅니다.

생성자	인수	설명
RandomAccessFile	(String name, String mode)	읽거나, 옵션으로 지정된 이름의 파일에 쓸 랜덤 액세스 파일을 생성합니다.
RandomAccessFile	(File file, String mode)	읽거나, 옵션으로 File 인수에서 지정한 파일에 쓸 랜덤 액세스를 생성합니다.

다음 표에서 표시한대로 RandomAccessFile에서 구현한 훌륭한 메소드들이 많이 있습니다.

메소드	인수	설명
seek	(long pos)	이 파일의 시작 지점부터 측정되며 읽기 또는 쓰기 작업이 발생하는 파일 포인터 오프셋을 설정합니다.
read	()	입력 스트림에서 다음 데이터 바이트를 읽습니다.
read	(byte b[], int off, int len)	오프셋 off에서 시작하는 최대 len 바이트의 데이터를 입력 스트림에서 배열로 읽어 들입니다.
readType	()	파일에서 readChar, readByte, readLong과 같은 지정된 데이터 타입을 읽습니다.
write	(int b)	지정된 바이트를 파일에 씁니다.
write	(byte b[], int off, int len)	오프셋 off에서 시작하는 바이트 배열로부터 len 바이트를 출력 스트림에 씁니다.
length	()	파일 길이를 반환합니다.
close	()	파일을 닫고 관련 시스템 리소스를 해제합니다.

참조 사항 JDK API Documentation에서 java.io.RandomAccessFile

StreamTokenizer 클래스는 다음과 같습니다: java.io.StreamTokenizer

java.io 패키지의 StreamTokenizer 클래스는 입력 스트림을 읽고 분할하거나 그리고 나서 하나씩 처리할 수 있는 개별 토큰으로 구문을 분석하는데 사용됩니다. 토큰은 숫자 또는 단어를 나타내는 문자 그룹입니다. 스트림 토큰 생성자는 문자열, 숫자, 식별자 및 주석을 인식합니다. 스트림을 토큰으로 처리하는 이 기술은 아마도 문자 입력을 처리하는 파서, 컴파일러 또는 프로그램에서 가장 일반적으로 사용될 것입니다.

이 클래스는 StreamTokenizer(Reader r)이라는 생성자 하나만 사용하며 다음과 같은 상수를 정의합니다.

상수	설명
TT_EOF	파일의 끝을 읽었음을 나타냅니다.
TT_EOL	줄의 끝을 읽었음을 나타냅니다.
TT_NUMBER	숫자 토큰을 읽었음을 나타냅니다.
TT_WORD	단어 토큰을 읽었음을 나타냅니다.

StreamTokenizer 클래스는 인스턴스 변수 nval, sval 및 ttype를 사용하여 각각 숫자 값, 문자열 값 및 토큰 타입을 저장합니다.

StreamTokenizer 클래스는 토큰의 사전적 구문을 정의하는 데 사용되는 몇 개의 메소드를 구현합니다.

메소드	인수	설명
nextToken	()	입력 스트림에서 다음 토큰의 구문을 분석합니다. 다음 토큰이 숫자인 경우 TT_NUMBER를 반환하고 다음 토큰이 단어 또는 문자인 경우 TT_WORD를 반환합니다.
parseNumbers	()	숫자의 구문을 분석합니다.
lineno	()	현재의 라인 번호를 반환합니다.
pushBack	()	nextToken() 메소드의 다음 호출 시 ttype 필드의 현재 값을 반환합니다.
toString	()	현재 토큰에 상응하는 문자열을 반환합니다.

스트림 토큰 생성자를 사용하는 경우 다음 절차를 실행합니다.

1. Reader의 StreamTokenizer 객체를 생성합니다.
2. 문자 처리 방법을 정의합니다.
3. nextToken() 메소드를 사용하여 다음 토큰을 가져옵니다.
4. ttype 인스턴스 변수를 읽어 토큰 타입을 찾습니다.
5. 인스턴스 변수에서 토큰 값을 읽습니다.
6. 토큰을 처리합니다.
7. nextToken()에서 StreamTokenizer.TT_EOF를 반환할 때까지 3~6단계를 반복합니다.

참조 사항 JDK API Documentation에서 java.io.StreamTokenizer

Chapter 17

Java의 객체 지향 프로그래밍

객체 지향 프로그래밍은 1967년 Simula' 67언어가 도입되면서 사용되었습니다. 그러나 실질적으로는 1980년 중반에서야 프로그래밍 패러다임의 전면에 떠오르게 되었습니다.

기존의 구조적 프로그래밍과는 달리 객체 지향 프로그래밍은 데이터 및 데이터에 속하는 연산을 단일 데이터 구조에 배치합니다. 구조적 프로그래밍에서 데이터 및 데이터의 연산은 분리되어 있으며 데이터 구조는 연산을 수행할 프로시저 및 함수에 전송됩니다. 속성 및 연산이 동일한 엔티티의 부분이므로 객체 지향 프로그래밍은 이 설계에서 발생하는 여러 문제점을 해결합니다. 객체 지향 프로그래밍은 모든 객체가 객체와 관련된 속성 및 활동을 모두 가지는 실제 세계를 더욱 세밀하게 모델링합니다.

Java는 *순수한* 객체 지향 언어인데 이는 Java의 가장 바깥쪽 레벨의 데이터 구조가 *객체*임을 의미합니다. Java에는 독립형 상수, 변수 및 함수가 없습니다. 클래스 및 객체에서 모두 액세스할 수 있다는 점이 Java의 가장 좋은 기능 중 하나입니다. 다른 혼합 객체 지향 언어는 객체 확장자 외에도 구조적 언어 특성을 가지고 있습니다. 예를 들어,C++ 및 Object Pascal은 객체 지향 언어이지만 객체 지향 확장자의 효과를 약화시키는 구조적 프로그래밍 구조를 작성할 수 있습니다. Java에서는 이를 수행할 수 없습니다!

이 장에서는 사용자가 다른 객체 지향 언어에 대해 어느 정도의 지식을 가지고 있다고 가정합니다. 그렇지 않은 경우 객체 지향 프로그래밍에 대한 보다 상세 설명이 들어 있는 다른 자료를 참조해야 합니다. 이 장에서는 Java의 객체 지향 기능을 강조하고 요약합니다.

클래스

클래스와 객체는 같은 것이 아닙니다. 클래스는 타입 정의인 반면 객체는 클래스 타입의 인스턴스 선언입니다. 클래스를 생성하면 해당 클래스를 토대로 원하는 만큼 여러 객체를 생성할 수 있습니다. 클래스와 객체 간의

관계는 체리 파이 조리법과 체리 파이의 관계와 같습니다. 즉 한가지 조리법을 사용하여 원하는 만큼의 체리 파이를 만들 수 있습니다.

클래스에서 객체를 생성하는 프로세스를 객체 *인스턴스화* 또는 클래스 *인스턴스* 생성이라고 부릅니다.

클래스 선언 및 인스턴스화

Java의 클래스는 매우 단순합니다. 다음은 빈 클래스에 대한 클래스 정의입니다.

```
class MyClass {
}
```

이 클래스는 아직 유용하지 않으나 Java에는 적합합니다. 보다 유용한 클래스는 몇 개의 데이터 멤버와 메소드를 포함하는데 이들은 곧 추가 될 것입니다. 먼저 클래스 인스턴스화를 위한 구문을 살펴봅니다. 이 클래스의 인스턴스를 생성하려면 클래스 이름과 관련있는 new 연결자를 사용하며 객체에 인스턴스 변수를 선언해야 합니다.

```
MyClass myObject;
```

그러나 인스턴스 변수를 선언한다고 해서 객체에 대해 메모리 및 다른 리소스가 할당되는 것은 아닙니다. myObject라는 참조가 생성되는 것일 뿐 객체는 인스턴스화되지 않습니다. new 연산자가 이 작업을 수행합니다.

```
myObject = new MyClass();
```

메소드인것 처럼 클래스 이름을 사용합니다. 다음 단원에서 살펴보겠지만 우연히 위와 같이 사용되는 것이 아닙니다. 이 코드 줄이 실행되면 아직 존재하지 않는 클래스의 멤버 변수 및 메소드에 "." 연산자를 사용하여 액세스할 수 있습니다.

일단 객체를 생성하면 이의 소멸에 대해서는 걱정할 필요가 없습니다. Java의 객체는 자동으로 가비지 컬렉트되는데 이것은 객체 참조가 더 이상 사용되지 않을 경우 가상 머신이 new 연산자가 할당한 모든 자원을 자동으로 해제함을 의미합니다.

데이터 멤버

위에서 설명한 것처럼 Java의 클래스는 데이터 멤버와 메소드 모두를 포함할 수 있습니다. 데이터 멤버 또는 멤버 변수는 클래스에 선언되는 변수입니다. 메소드는 몇 가지 작업을 수행하는 함수 또는 루틴입니다. 다음은 데이터 멤버만 포함하는 클래스입니다.

```
public class DogClass {
  String name, eyeColor;
  int age;
  boolean hasTail;
}
```

이 예제는 데이터 멤버를 포함하는 DogClass라는 클래스를 생성합니다. name, eyeColor, age와 hasTail이라는 플래그가 데이터 멤버입니다. 클래스의 멤버 변수로서 어떠한 데이터 타입이라도 포함할 수 있습니다. 데이터 멤버에 액세스하려면 먼저 클래스 인스턴스를 생성한 다음 "." 연산자를 사용하여 데이터에 액세스해야 합니다.

클래스 메소드

클래스에 메소드도 포함할 수도 있습니다. 사실상 Java에는 독립 함수 또는 프로시저가 없습니다. 모든 하위 루틴은 클래스의 메소드로 정의됩니다. 다음은 speak() 메소드가 추가된 DogClass 예제입니다.

```
public class DogClass {
  String name,eyeColor;
  int age;
  boolean hasTail;

  public void speak() {
    JOptionPane.showMessageDialog(null, "Woof! Woof!");
  }
}
```

메소드를 정의할 때 메소드 구현은 선언 바로 아래 나타납니다. 이것이 클래스가 한 위치에 정의되고 구현 코드가 그밖의 다른 위치에 나타나는 다른 객체 지향 언어와 틀린 점입니다. 메소드는 반환 타입 및 메소드가 수신한 매개변수를 지정해야 합니다. speak() 메소드에는 매개변수를 사용할 수 없습니다. 이 메소드는 또한 값을 반환하지 않기 때문에 반환 타입은 void입니다.

메소드를 호출하려면 멤버 변수에 액세스할 때처럼 즉, "." 연산자를 사용하여 메소드에 액세스합니다.예를 들면, 다음과 같습니다.

```
DogClass dog = new DogClass();
dog.age = 4;
dog.speak();
```

생성자 및 종결자

모든 Java 클래스는 *constructor*라는 특수 목적의 메소드를 가지고 있습니다. 생성자는 항상 클래스와 동일한 이름을 가지고 있고 반환 값을 지정할 수 없습니다. 생성자는 객체가 필요로 하는 모든 자원을 할당하며 객체의 인스턴스를 반환합니다. new 연산자를 사용하는 것은 사실상 생성자를 호출하는 것입니다. 객체의 인스턴스는 항상 반환 타입이기 때문에 생성자에 반환 타입을 지정할 필요가 없습니다.

대부분의 객체 지향 언어는 *소멸자*라고 하는 해당 메소드를 가지고 있는데 이 소멸자는 생성자가 할당한 모든 자원을 해제하기 위해 호출됩니다. 그러나 Java가 사용자에 대한 모든 리소스를 자동으로 해제하기 때문에 Java에는 소멸자 메커니즘이 없습니다.

그러나 클래스가 삭제될 때 가비지 콜렉터가 처리할 수 없는 몇 가지 특수 지우기를 사용자가 수행해야 하는 상황이 있습니다. 예를 들어, 객체의 수명에 영향받는 일부 파일을 여는 경우 사용자는 객체가 소멸될 때 파일이 제대로 종료되기를 원할 것입니다. *종결자*라는 클래스에 대해 정의할 수 있는 특수 목적을 갖는 또다른 메소드가 있습니다. 이 메소드(있는 경우)를 객체 소멸 직전에 가비지 콜렉터가 호출합니다. 그러므로 수행해야 할 특수 지우기 작업이 있는 경우 종결자에서 이를 처리할 수 있습니다. 그러나 가비지 콜렉터는 가상 머신에서 우선순위가 낮은 스레드로 실행되므로 사용자는 가비지 콜렉터가 실제로 객체를 언제 소멸하는지 예상할 수 없습니다. 이런 코드가 언제 호출될지 예상할 수 없으므로 타이밍에 맞는 코드를 종결자에 삽입해서는 안됩니다.

사례 연구: 간단한 OOP 예제

이 단원에서는 클래스 정의 및 객체 인스턴스화에 관한 간단한 예제를 살펴 보기로 합니다. 두 객체(개와 사람)를 생성하는 애플리케이션을 개발하고 객체의 속성을 폼에 나타낼 것입니다.

JBuilder를 처음 접하는 경우 이 장은 나중에 보고 이 예제를 시작하기 전에 JBuilder의 통합 개발 환경에 대해 배워야 합니다. Chapter 5, "사용자 인터페이스 구축"부터 시작하십시오. Chapter 3, "에디터 사용" 및 Chapter 4, "애플리케이션 개발 자동화"도 읽어보아야 합니다. 이 모두를 한 번에 배우려면 Chapter 23, "자습서: 애플리케이션 구축" 에서 예제 애플리케이션을 생성해 보십시오.

다음과 같은 작업 수행이 수월해지면 이 장을 다시 살펴볼 준비가 된 것입니다.

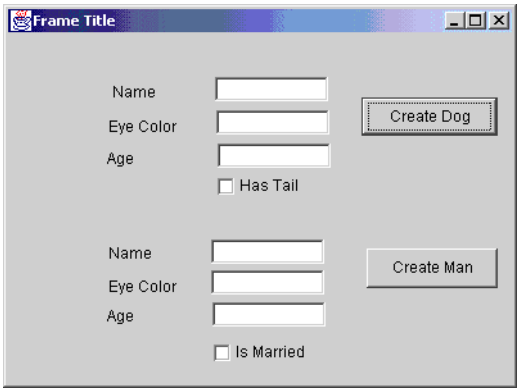
- JBuilder의 Application 마법사를 사용한 애플리케이션 시작
- 컴포넌트 팔레트로부터의 컴포넌트 선택 및 UI 디자이너로의 컴포넌트 배치
- Inspector를 사용하여 컴포넌트 속성 설정
- JBuilder의 컨텐트 창에서 에디터와 UI 디자이너 전환

- 에디터 사용

JBuilder 개발 환경에 추가 질문이 있는 경우 *Building Applications with JBuilder* 책의 관련 장에서 추가 정보를 찾아 볼 수 있습니다.

구축하게 될 실행 예제 애플리케이션은 다음과 같이 나타납니다.

Figure 17.1인스턴스화된 두 객체를 보여 주는 예제 애플리케이션



이 단원에 나열된 다음 단계를 따라 이 예제 애플리케이션을 위한 간단한 UI를 생성합니다.

1 다음과 같은 방법으로 애플리케이션 생성 및 UI 디자인을 시작합니다.

  1 새 프로젝트를 시작합니다. Project 마법사를 시작하려면 File|New Project를 선택합니다.

  2 oop1을 Project Name 필드에 입력하고 Finish를 클릭합니다.

  새 프로젝트가 열립니다 .

  3 File|New를 선택하고 Application 아이콘을 클릭하여 Application 마법사를 시작합니다.

  4 기본 클래스 이름을 적용합니다. 패키지 이름은 oop1입니다.

  5 Next를 클릭한 다음 Finish를 클릭하여 Frame1.java 및 Application1.java 파일을 생성합니다.

  6 컨텐트 창의 Design 탭을 클릭하여 Frame1.java의 UI 디자이너를 표시합니다.

  7 구조 창에서 contentPane을 선택합니다. Inspector에서 contentPane의 layout 속성을 XYLayout에 설정합니다.(Foundation 사용자의 경우 layout을 null에 설정하십시오.) XYLayout 및 null은 좀처럼 애플리케이션에 적합하지 않으나 레이아웃 사용 방법을 배우기 전까지 이를 사용하여 깔끔하진 않지만 빠르게 UI를 생성할 수 있습니다.

2 위의 스크린샷을 참조하여 필요한 컴포넌트를 UI 디자이너에 놓습니다.

1. 컴포넌트 팔레트의 Swing 탭을 클릭하고 JTextField 컴포넌트를 클릭합니다. (컴포넌트에 커서를 위치시키면 컴포넌트가 레이블된 툴팁이 나타납니다.) javax.swing.JTextField가 레이블된 컴포넌트를 클릭합니다.} UI 디자이너를 클릭하고 컴포넌트를 화면에 그릴 때 마우스 버튼을 누른 상태로 있습니다. 폼에 세 개의 JTextField 컴포넌트로 이루어진 그룹 두 개가 생성될 때까지 이 단계를 다섯 번 더 반복합니다.
2. UI 디자이너의 각 JTextField를 클릭할 때 Shift키를 눌러 모든 JTextField를 선택합니다. Inspector에서 text 속성을 선택하고 여기에 나타나는 텍스트를 삭제합니다. 이렇게 하면 각 JTextField 컴포넌트의 모든 텍스트가 삭제됩니다.
3. 각 JTextField의 name 속성 값을 변경합니다. 첫번째는 txtfldDogName, 두번째는 txtfldDogEyeColor, 세번째는 txtfldDogAge, 네번째는 txtfldManName, 다섯번째는 txtfldManEyeColor 그리고 여섯번째는 txtfldManAge로 이름을 지정합니다.
4. 폼에 6개의 JLabel 컴포넌트를 그립니다. 각 컴포넌트는 JTextField 컴포넌트와 인접해 있어야 합니다.
5. 이러한 컴포넌트의 text 속성 값을 변경하여 적절하게 각 JTextField 컴포넌트에 레이블을 붙입니다. 예를 들어, 폼의 상단에 있는 JLabel의 text 속성은 Name이고 두번째 속성은 Eye Color이어야 합니다.
6. 폼에 JCheckBox 컴포넌트 두 개를 놓습니다. 세 개의 JTextField 컴포넌트로 이루어진 첫번째 그룹 아래에 첫번째 체크박스를 놓고 JTextField 컴포넌트으로 이루어진 두번째 그룹 아래에 두번째 체크박스를 놓습니다.
7. 차례로 폼의 각 JCheckBox 컴포넌트를 선택하여 첫번째 컴포넌트의 text 속성을 Has Tail로 변경하고 두번째 컴포넌트의 text 속성을 Is Married로 변경합니다.
8. 첫번째 체크박스의 name 속성 값을 chkboxDog로 변경하고 두번째 체크박스의 이름을 chkboxMan으로 변경합니다.
9. 폼에 JButton 컴포넌트 두 개를 놓는데 첫번째는 컴포넌트의 상단 그룹 오른쪽에 두번째는 컴포넌트의 하단 그룹 오른쪽에 놓습니다.
10. 첫번째 버튼의 text 속성을 Create Dog로, 두번째 버튼의 text 속성을 Create Man으로 변경합니다.

마지막 단계에서 File|Save All을 선택하여 프로젝트를 저장합니다.

이제 프로그래밍을 시작할 준비가 되었습니다. 먼저 새 클래스를 생성합니다.

1 File|New Class를 선택하여 Class 마법사를 시작합니다.

2 패키지 이름은 oop1으로 유지하고 Class Name은 DogClass로 하되 Base Class는 변경하지 않습니다.

3 Public 및 Generate Default Constructor 옵션만 선택표시하고 다른 모든 옵션은 선택 해제합니다.

4 OK를 클릭합니다.

Class 마법사는 DogClass.java 파일을 생성합니다. 생성된 코드가 다음과 같이 나타나도록 코드를 수정합니다.

```
package oop1;

public class DogClass {
  String name, eyeColor;
  int age;
  boolean hasTail;

  public DogClass() {
    name = "Snoopy";
    eyeColor = "Brown";
    age = 2;
    hasTail = true;
  }
}
```

DogClass를 기타의 멤버 변수와 함께 정의합니다. 또한 DogClass 객체를 인스턴스화하는 생성자도 있습니다.

Class 마법사를 사용하여 Class Name을 ManClass로 지정하는 경우를 제외하고 동일한 단계를 반복하여 ManClass.java 파일을 생성합니다. 다음과 같이 결과 코드를 수정합니다.

```
package oop1;

public class ManClass {
  String name, eyeColor;
  int age;
  boolean isMarried;

  public ManClass() {
    name = "Steven";
    eyeColor = "Blue";
    age = 35;
    isMarried = true;
  }
}
```

이 두 클래스는 매우 유사합니다. 다음 단원에서 이 유사성을 이용하게 됩니다.

컨텐트 창 상단의 Frame1 탭을 클릭하여 Frame1 클래스로 되돌아갑니다. 하단의 Source 탭을 클릭하고 되돌아가 에디터를 엽니다. 객체 참조로서 두 인스턴스 변수를 선언합니다. 다음은 굵게 표시된 부분이 Frame1 변수 선언의 소스 목록입니다. 굵게 표시된 줄을 클래스에 추가하십시오.

```
public class Frame1 extends JFrame {
// Create a reference for the dog and man objects
  DogClass dog;
  ManClass man;

  JPanel contentPane;
  JPanel jPanel1 = new JPanel();
  . . .
```

컨텐트 창 하단의 Design 탭을 클릭하여 자신이 디자인한 UI로 되돌아갑니다. Create Dog 버튼을 더블 클릭합니다. JBuilder는 해당 버튼에 대한 이벤트 핸들러의 시작점을 생성하며 커서를 이벤트 핸들러 코드 안에 놓습니다. 이벤트 핸들러 코드를 입력하여 dog 객체를 인스턴스화하고 dog 텍스트 필드를 입력할 수 있습니다.코드는 다음과 같아야 합니다.

```
void jButton1_actionPerformed(ActionEvent e) {
  dog = new DogClass();
  txtfldDogName.setText(dog.name);
  txtfldDogEyeColor.setText(dog.eyeColor);
  txtfldDogAge.setText(Integer.toString(dog.getAge()));
  chkboxDog.setSelected(true);
}
```

코드에서 볼 수 있는 것처럼 dog 객체 생성자를 호출한 다음 dog 객체 생성자의 멤버 변수에 액세스합니다.

Design 탭을 클릭하여 UI 디자이너로 되돌아갑니다. Create Man 버튼을 더블 클릭합니다. JBuilder에서 Create Man 버튼의 이벤트 핸들러가 생성됩니다. 다음과 같이 이벤트 핸들러에 입력합니다.

```
void jButton2_actionPerformed(ActionEvent e) {
  man = new ManClass();
  txtfldManName.setText(man.name);
  txtfldManEyeColor.setText(man.eyeColor);
  txtfldManAge.setText(Integer.toString(man.getAge()));
  chkboxMan.setSelected(true);
}
```

이제 애플리케이션을 컴파일하고 실행할 수 있습니다. Project|Make Project "oop1.jpx"를 선택하여 컴파일합니다. 오류가 없는 경우 Run|Run Project를 선택합니다.

모든 상황이 정상인 경우 폼이 화면에 나타납니다. Create Dog 버튼을 클릭하면 dog 객체가 생성되고 dog 값이 dog 필드에 나타납니다. Create Man 버튼을 클릭하면 man 객체가 생성되고 man 값이 해당 필드에 나타납니다.

클래스 상속

생성한 dog과 man 객체에는 유사한 점이 많이 있습니다. 객체 지향 프로그래밍의 이점 중 하나는 계층에서 이와 같은 유사성을 처리하는 기능입니다. 이 기능을 *상속*이라고 부릅니다. 클래스가 다른 클래스에서 상속되면 자식 클래스는 자동으로 모든 특성(멤버 변수) 및 동작(메소드)을 부모 클래스에서 상속합니다. 상속은 항상 추가적이기 때문에 클래스에서 상속받거나 부모 클래스가 가지고 있는 것보다 적게 상속 받는 방법은 없습니다.

Java의 상속은 extends 키워드를 통해 처리됩니다. 한 클래스가 다른 클래스에서 상속되면 자식 클래스는 부모 클래스를 확장합니다.예를 들면, 다음과 같습니다.

```
public class DogClass extends MammalClass {
. . .
}
```

사람과 개가 공통으로 가지고 있는 항목은 모든 포유류에서 동일하다고 할 수 있으므로 MammalClass를 생성하여 이러한 유사성을 처리할 수 있습니다. DogClass 및 ManClass에서 공통 항목 선언을 제거하고 그 대신 MammalClass에 공통 항목을 선언한 다음 MammalClass에서 하위 클래스인 DogClass와 ManClass를 제거할 수 있습니다.

Class 마법사를 사용하여 MammalClass를 생성합니다. 다음과 같이 결과 코드를 입력해야 합니다.

```
package oop1;

public class MammalClass {
  String name, eyeColor;
  int age;

  public MammalClass() {
    name = "The Name";
    eyeColor = "Brown";
    age = 0;
  }
}
```

MammalClass는 DogClass와 ManClass에 공통되는 특성을 가지고 있습니다. 이제 DogClass와 ManClass를 다시 작성하여 상속을 이용합니다.

다음과 같이 DogClass 코드를 수정합니다.

```
package oop1;

 public class DogClass extends MammalClass {

  boolean hasTail;

  public DogClass() {
    // implied super()
    name = "Snoopy";
    age = 2;
    hasTail = true;
  }
}
```

다음과 같이 ManClass 코드를 수정합니다.

```
package oop1;

public class ManClass extends MammalClass{

  boolean isMarried;

  public ManClass() {
    name = "Steven";
    eyeColor = "Blue";
    age = 35;
    isMarried = true;
  }
}
```

DogClass는 특별히 eyeColor 값을 할당하지 않는나 ManClass는 할당한다는 점에 유의하십시오. DogClass는 개 Snoopy가 갈색 눈을 가지고 있고 DogClass가 eyeColor 변수를 선언하고 이 변수를 "Brown"값에 할당하는 MammalClass에서 갈색 눈을 상속하기 때문에 값을 eyeColor에 할당할 필요가 없습니다. 그러나 사람 Steven은 파란 눈을 가지고 있기 때문에 "Blue" 값을 MammalClass에서 상속되는 eyeColor 변수에 할당해야 합니다.

다시 프로젝트를 컴파일하여 실행하여 보십시오. (Run|Run Project를 선택하면 애플리케이션은 컴파일된 다음 실행됩니다.) 프로그램의 UI는 방금 전과 동일해 보이지만 dog 및 man 객체는 이제 MammalClass에서 모든 공통 멤버 변수를 상속합니다.

DogClass가 MammalClass를 확장하는 대로 DogClass는 MammalClass가 가지고 있는 모든 멤버 변수와 메소드를 갖게 됩니다. 실제로 MammalClass조차도 다른 클래스에서 상속됩니다. Java의 모든 클래스는 궁극적으로는 Object 클래스를 확장합니다. 따라서 다른 클래스를 확장하지 않는 클래스가 선언된 경우 이 클래스는 암시적으로 Object 클래스를 확장합니다.

Java의 클래스는 한 번에 한 클래스에서만 상속할 수 있습니다(*단일 상속*). Java와 달리 C++와 같은 몇몇 언어를 통해 클래스는 한 번에 여러 클래스

로부터 상속하기도 합니다(*다중 상속*). 클래스는 한 번에 한 클래스만 확장할 수 있습니다. 상속을 사용하여 계층을 확장할 수 있는 횟수에 제한이 없다고 하더라도 한 번에 한 확장에서만 사용해야 합니다. 다중 상속은 좋은 기능이지만 다중 상속을 통해 객체 계층이 매우 복잡해 질 수 있습니다. Java는 후에 살펴 보게 될 복잡하지 않으면서도 동일한 여러 가지 이점을 제공하는 메커니즘을 가지고 있습니다.

MammalClass는 매우 실용적이면서도 편리한 기본값을 설정하는 생성자를 가지고 있습니다. 하위 클래스가 이 생성자에 액세스할 수 있다면 더욱 좋을 것입니다.

실제로 하위 클래스는 이 생성자에 액세스할 수 있는데 Java에서는 이 작업을 두 가지 다른 방법으로 수행할 수 있습니다. 부모 생성자를 명시적으로 호출하지 않는 경우 자식 생성자의 첫번째 줄처럼 Java는 부모의 기본 생성자를 자동으로 호출합니다. 이러한 동작이 일어나지 않도록 하는 유이한 방법은 자식 클래스의 첫번째 줄처럼 부모의 생성자 중 하나를 직접 호출하는 것입니다. 생성자 호출은 항상 이와 같이 체인화되어 있으며 이 메커니즘을 깰 수 없습니다. 이는 Java 언어의 매우 우수한 기능인데 다른 객체 지향 언어에서는 부모 생성자 호출 실패가 흔히 발생하는 버그이기 때문입니다. Java는 사용자가 호출하지 않는 경우 사용자를 위해 항상 이 작업을 수행합니다. DogClass 생성자(// implied super())의 첫번째 줄에 있는 주석이 의미하는 것이 바로 이것입니다. MammalClass 생성자는 이 지점에서 자동으로 호출됩니다. 이 메커니즘은 매개변수를 사용하지 않는 수퍼클래스(부모 클래스) 생성자의 존재에 따라 결정됩니다. 생성자가 존재하지 않고 사용자가 자식 생성자의 첫번째 줄처럼 다른 생성자 중 하나를 호출하지 않는 경우 클래스는 컴파일하지 않습니다.

부모 생성자 호출

사용자는 명시적으로 수퍼클래스 생성자를 자주 호출하려 하기 때문에 Java에는 이 작업을 수월하게 하는 키워드가 있습니다. super()는 적절히 제공되는 매개변수를 갖는 부모의 생성자를 호출합니다.

하나의 클래스에 둘 이상의 생성자가 있을 수도 있습니다. 한 클래스에 동일한 이름의 메소드가 둘 이상 있는 경우에는 이 메소드를 overloaded라고 합니다. 하나의 클래스에 여러 개의 생성자가 있기도 합니다.

예제 애플리케이션의 경우 계층에 있는 변경이 처음 두 예제 버전 간의 유일한 차이점입니다. 객체 인스턴스화 및 기본 폼은 전혀 변경되지 않습니다. 현재 포유류의 모든 특성을 수정해야 하는 경우 이 작업은 MammalClass에서 수행할 수 있으며 자식 클래스만 다시 컴파일하면 되기 때문에 애플리케이션의 디자인은 더욱 효율적입니다. 작성한 변경 사항은 자식 클래스에 적용됩니다.

액세스 변경자

클래스 멤버(변수 및 메소드)에 액세스할 수 있는 시기를 이해하는 것은 매우 중요합니다. Java에는 멤버에 액세스하는 방법을 자세히 지정할 수 있는 몇 가지 옵션이 있습니다.

일반적으로 사용자는 클래스 멤버를 포함한 프로그램 요소의 범위를 가능한 한 제한하고 싶어합니다. 액세스할 수 있는 공간이 적을 수록 부적절하게 액세스할 수 있는 공간이 적어집니다.

Java에는 클래스 멤버에 대해 private, protected, public 및 default와 같은 네 가지 다른 액세스 변경자가 있습니다(또는 변경자가 없을 수도 있습니다). 동일한 패키지의 클래스는 패키지 외부의 클래스와 다르게 액세스한다는 사실 때문에 조금 복잡합니다. 다음의 두 가지 표는 동일한 패키지와 패키지 외부에 있는 클래스 및 멤버 변수의 액세스 가능성 및 상속 가능성을 보여줍니다(패키지는 다음 단원에서 설명하기로 합니다).

클래스 패키지에서 액세스

Access Modifier	Inherited	Accessible
기본값(변경자 없음)	예	예
Public	예	예
Protected	예	예
Private	아니오	아니오

이 표는 클래스 멤버가 동일한 패키지의 다른 멤버와 관련하여 액세스 및 상속되는 방법을 보여줍니다. 예를 들어, private으로 선언된 멤버는 동일한 패키지의 다른 멤버에서 액세스하거나 상속할 수 없습니다. 반면에 다른 변경자를 사용하여 선언된 멤버는 해당 패키지의 모든 다른 맴버에 의해 액세스되거나 상속될 수 있습니다. 동일한 애플리케이션의 모든 부분은 oop1 패키지의 부분이므로 다른 패키지에서의 클래스 액세스에 관해 염려할 필요가 없습니다.

패키지 외부로의 액세스

클래스 패키지의 외부에서 코드에 액세스하는 경우 규칙이 변경됩니다.

Access Modifier	Inherited	Accessible
기본값(변경자 없음)	아니오	아니오
Public	예	예
Protected	예	아니오
Private	아니오	아니오

예를 들면, 이 표는 protected 멤버는 패키지 외부의 클래스에서 상속 받을 수 있지만 액세스할 수 없음을 보여줍니다.

이 두 가지 액세스 표에서 보면 public 멤버를 액세스하려는 모든 사용자가 사용할 수 있는 반면(생성자는 항상 public임), private 멤버는 클래스 외부에서 액세스하거나 상속 받을 수 없습니다. 그러므로 클래스 내부에서 사용하려는 멤버 변수 또는 메소드는 private으로 선언해야 합니다.

객체 지향 프로그래밍에서는 클래스의 모든 멤버 변수를 private으로 하여 accessor 메소드라고 하는 특별 형식의 메소드를 통해 액세스함으로써 클래스 내의 정보를 숨기도록 권장하고 있습니다.

accessor 메소드

accessor 메소드(getter 또는 setter라고도 함)는 실제 데이터 저장소를 클래스에 대해 private으로 유지하면서 외부 public 인터페이스를 클래스에 제공하는 메소드입니다. 이것은 향후 어느 때에나 실제로 내부 값을 설정하는 메소드를 수정하지 않고 클래스 안의 데이터 내부 표시를 변경할 수 있기 때문에 효과적인 아이디어입니다. 클래스에 대한 public 인터페이스를 변경하지 않는 한 해당 클래스 및 public 메소드에 의존하는 코드는 깨지지 않습니다.

Java의 accessor 메소드는 대개 쌍으로 표시합니다. 즉, 하나는 내부 값을 *얻기* 위해, 다른 하나는 내부 값을 *설정*하기 위해서 입니다. 규칙에 따라 Get 메소드는 접두사로 "get"과 함께 내부 private 변수 이름을 사용합니다. Set 메소드도 마찬가지로 "set"과 함께 사용합니다. 읽기 전용 속성은 Get 메소드만 가집니다. 일반적으로 Boolean Get 메소드는 접두사로 "get" 대신에 "is" 또는 "has"를 사용합니다. accessor 메소드는 또한 특정 멤버 변수에 지정되는 데이터를 쉽게 검증 할 수 있도록 해줍니다.

예제는 다음과 같습니다. DogClass에서 모든 내부 멤버 변수를 private으로 지정하고 accessor 메소드를 추가하여 내부 값에 액세스합니다. DogClass는 단 하나의 새로운 멤버 변수 tail을 생성합니다.

```
package oop1;

public class DogClass extends MammalClass{

  // accessor methods for properties
  // Tail
  public boolean hasTail() {
    return tail;
  }

  public void setTail( boolean value ) {
    tail = value;
  }

  public DogClass() {
    setName("Snoopy");
    setAge(2);
    setTail(true);
  }

  private boolean tail;
}
```

변수 tail은 클래스 아래로 이동되며 이제 private으로 선언됩니다. 정의 위치는 중요하지 않지만 Java에서는 클래스의 private 멤버를 클래스 정의

아래에 배치하는 것이 일반적입니다(결국 사용자는 클래스 외부에서 액세스할 수 없기 때문에 코드를 읽는 경우 먼저 public에 관심을 갖게 됩니다). DogClass는 이제 public 메소드를 가지고 tail 값을 가져와 설정합니다. getter 메소드는 hasTail()이고 setter 메소드는 setTail()입니다.

동일한 방법을 따라 다음과 같이 ManClass를 변경합니다.

```
package oop1;

 public class ManClass extends MammalClass {

  public boolean isMarried() {
    return married;
  }

  public void setMarried(boolean value) {
    married = value;
  }

  public ManClass() {
    setName("Steven");
    setAge(35);
    setEyeColor("Blue");
    setMarried(true);
  }

  private boolean married;
}
```

이러한 두 클래스에 대한 생성자는 이제 accessor 메소드를 사용하여 MammalClass의 변수값을 설정합니다. 그러나 MammalClass는 이러한 값 설정을 위한 accessor 메소드를 가지고 있지 않기 때문에 accessor 메소드를 MammalClass에 추가해야 합니다.

코드가 다음과 같이 나타나도록 MammalClass를 변경합니다.

```
public class MammalClass {

  // accessor methods for properties
  // name
  public String getName() {
    return name;
  }

  public void setName(String value) {
    name = value;
  }
```

```
    // eyecolor
    public String getEyeColor() {
      return eyeColor;
    }

    public void setEyeColor(String value) {
      eyeColor = value;
    }

    // sound
    public String getSound() {
      return sound;
    }

    public void setSound(String value) {
      sound = value;
    }

    // age
    public int getAge() {
      return age;
    }

    public void setAge(int value)  {
      if (value > 0) {
        age = value;
      }
  else
      age = 0;
    }

    public MammalClass() {
      setName("The Name");
      setEyeColor("Brown");
      setAge(0);
    }

    private String name, eyeColor, sound;
    private int age;
  }
```

또한 새로운 sound 멤버 변수가 MammalClass에 추가되었음을 유의하십시오. 이 클래스 역시 accessor 메소드를 가지고 있습니다. DogClass와 ManClass는 MammalClass를 확장하므로 이들 역시 sound 속성을 갖게 됩니다.

Frame1.java의 이벤트 핸들러 역시 accessor를 사용해야 합니다. 다음과 같이 이벤트 핸들러를 수정합니다.

```
  void jButton1_actionPerformed(ActionEvent e) {
    dog = new DogClass();
    txtfldDogName.setText(dog.getName());
    txtfldDogEyeColor.setText(dog.getEyeColor());
    txtfldDogAge.setText(Integer.toString(dog.getAge()));
```

```
  chkboxDog.setSelected(true);
}

void jButton2_actionPerformed(ActionEvent e) {
  man = new ManClass();
  txtfldManName.setText(man.getName());
  txtfldManEyeColor.setText(man.getEyeColor());
  txtfldManAge.setText(Integer.toString(man.getAge()));
  chkboxMan.setSelected(true);
}
```

추상 클래스

클래스의 메소드를 *abstract*으로 선언할 수 있는데 이것은 이 클래스의 메소드에 대한 구현은 없지만 이 클래스를 확장하는 모든 클래스가 구현을 제공한다는 것을 의미합니다.

예를 들어, 사용자는 모든 포유류가 자신들의 달리기 최고 속도를 보고하기를 원하되 각 포유류 마다 다른 속도를 보고하기를 원한다고 가정합니다. 포유류 클래스에서 *speed()*라고 하는 추상 메소드를 생성해야 합니다. 소스 코드 아래의 private 멤버 변수 선언 바로 위의 MammalClass에 speed() 메소드를 추가합니다.

```
abstract public void speed();
```

클래스에 추상 메소드가 있으면 전체 클래스 또한 추상으로 선언되어야 합니다. 이것은 최소한 하나의 추상 메소드(따라서 추상 클래스도 포함)를 포함하는 클래스는 인스턴스화할 수 없음을 나타냅니다. 그러므로 abstract 키워드를 다음과 같이 MammalClass 선언 시작에 추가합니다.

```
abstract public class MammalClass {

  public String getName() {
  ...
```

MammalClass를 확장하는 각각의 클래스는 speed() 메소드를 구현해야 합니다. 그러므로 이 메소드를 DogClass() 생성자 아래의 DogClass 코드에 추가합니다.

```
public void speed() {
  JOptionPane.showMessageDialog(null, "30 mph", "Dog Speed", 1);
}
```

이 speed() 메소드를 ManClass 코드에 추가합니다.

```
public void speed() {
  JOptionPane.showMessageDialog(null, "17 mph", "Man Speed", 1);
}
```

각 speed() 메소드가 Swing 컴포넌트인 JOptionPane 컴포넌트를 생성하므로 패키지 문장 바로 뒤에 있는 이 문장을 다음과 같이 DogClass와 ManClass의 상단에 추가합니다.

```
import javax.swing.*;
```

이 문장을 통해 전체 Swing 라이브러리를 이러한 클래스에서 사용할 수 있습니다. 곧 import 문에 대해 살펴 보게 될 것입니다.

다형성

다형성은 분리되어 있으나 아직까지 연결되어 있는 두개의 클래스가 동일한 메시지를 수신하지만 각기 자신의 방식 대로 이를 처리하는 기능을 말합니다. 다시 말해서 상이하지만 연결되어 있는 두 클래스가 동일한 메소드 이름을 가지고 있으나 다른 방식으로 메소드를 구현한다는 것입니다.

따라서 차일드 클래스에도 구현되는 클래스를 가질 수 있으며 부모 클래스에서 코드에 액세스할 수 있습니다(앞서 설명한 생성자 자동 체인화와 유사). 생성자 예제에서와 같이 super 키워드를 사용하여 수퍼클래스의 모든 메소드와 멤버 변수에 액세스할 수 있습니다.

여기에서 간단한 예제로 Parent와 Child 두 클래스를 가지고 있다고 합시다.

```
class Parent {
  int aValue = 1;
  int someMethod(){
    return aValue;
  }
}

class Child extends Parent {
  int aValue;                          // this aValue is part of this class
  int someMethod() {                       // this overrides Parent's method
    aValue = super.aValue + 1;           // access Parent's aValue with super
    return super.someMethod() + aValue;
  }
}
```

Child의 someMethod()는 Parent의 someMethod()를 오버라이드합니다. 따라서 부모 클래스의 메소드와 이름은 동일하지만 다르게 구현되어 다른 동작을 취하는 자식 클래스 메소드는 overridden 메소드입니다.

Child 클래스의 someMethod()가 값 3을 반환하는 방법이 보입니까? 이 메소드는 super 키워드를 사용하여 Parent의 aValue 변수에 액세스하며 값 1을 이 변수에 추가하고 결과 값 2를 자신의 aValue 변수에 할당합니다. 메소드의 마지막 줄은 값 1과 함께 Parent.aValue를 반환하는 Parent의 someMethod()를 호출합니다. 여기에 앞 줄에서 값 2를 할당한 Child.aValue의 값을 추가합니다. 그러므로 1 + 2 = 3이 됩니다.

인터페이스 사용

인터페이스와 추상 클래스와 매우 유사하지만 한 가지 중요한 차이점이 있는데 인터페이스는 어떠한 코드를 포함할 수 없다는 점입니다. Java의 인터페이스 메커니즘은 다중 상속을 대체하기 위해 고안되었습니다.

인터페이스는 특수한 클래스 선언으로 상수 및 메소드 선언을 선언할 수 있지만 메소드를 구현할 수는 없습니다. 인터페이스에 코드를 삽입할 수 없습니다.

다음은 예제 애플리케이션에 대한 인터페이스 선언입니다.

JBuilder Interface 마법사를 사용하여 다음과 같이 인터페이스를 시작할 수 있습니다.

1. 객체 갤러리를 열려면 File|New를 선택합니다. Interface 아이콘을 더블 클릭하여 Interface 마법사를 표시합니다.
2. 인터페이스 이름을 SoundInterface로 지정하여 기타 모든 값이 변경되지 않게 합니다. (Generate Header Comments를 선택 해제하여 헤더를 생략할 수 있습니다.)
3. OK를 선택하여 새 인터페이스를 생성합니다.

새 SoundInterface에서 다음과 같이 speak() 메소드 선언을 추가합니다.

```
package oop1;

public interface SoundInterface {

  public void speak()
}
```

interface 키워드가 class 대신 사용됨을 유의하십시오. 인터페이스에 선언된 모든 메소드는 기본적으로 public이기 때문에 액세스 기능을 지정할 필요가 없습니다. 클래스는 implements 키워드를 사용하여 인터페이스를 구현할 수 있습니다. 또한 클래스는 단 하나의 다른 클래스를 확장할 수 있지만 클래스는 필요한 만큼 인터페이스를 구현할 수 있습니다. 이는 Java의 인터페이스가 다른 언어의 다중 상속에 의해 처리하는 상황을 처리하는 방법입니다. 여러 경우에서 클래스처럼 인터페이스를 사용할 수 있습니다. 다시 말해서 편의상 인터페이스를 구현하는 객체를 인터페이스의 하위 클래스처럼 처리할 수 있다는 것입니다. 그러나 인터페이스를 구현하는 객체를 변환하는 경우 인터페이스가 정의하는 메소드에만 액세스할 수 있음을 유의하십시오.

다음은 다형성 및 인터페이스에 대한 예제입니다. MammalClass 정의에서 새 SoundInterface를 구현하려 할 경우 단어 implements SoundInterface를 클래스 정의에 추가하여 구현합니다. 그런 다음 MammalClass에 대해 speak() 메소드를 정의하고 구현해야 합니다. MammalClass를 수정하여 SoundInterface 및 speak() 메소드를 구현합니다. 다음은 MammalClass에 대한 전체 코드입니다.

```
package oop1;

import javax.swing.*;

abstract public class MammalClass implements SoundInterface {

  // accessor methods for properties
  // name
```

```
  public String getName() {
    return name;
  }

  public void setName( String value ) {
    name = value;
  }

  // eyecolor
  public String getEyeColor() {
    return eyeColor;
  }

  public void setEyeColor( String value ) {
    eyeColor = value;
  }

  // sound
  public String getSound() {
    return sound;
  }

  public void setSound( String value ) {
    sound = value;
  }

  // age
  public int getAge() {
    return age;
  }

  public void setAge( int value ) {
    if ( value > 0 )
    {
      age = value;
    }
else
    age = 0;
  }

  public MammalClass() {
    setName("The Name");
    setEyeColor("Brown");
    setAge(0);
  }

  public void speak() {
    JOptionPane.showMessageDialog(null, this.getSound(),
    this.getName() + " Says", 1);
  }

  abstract public void speed();

  private String name, eyeColor, sound;
```

```
  private int age;
}
```

이제 MammalClass 정의는 SoundInterface를 완벽하게 구현합니다. speak() 메소드 구현이 Swing 라이브러리의 일부인 JOptionPane 컴포넌트를 사용하므로 파일 상단 근처에 import 문장을 추가해야 합니다.

```
import javax.swing.*;
```

이 import 문을 통해 MammalClass에서 전체 Swing 라이브러리를 사용할 수 있습니다. import 문에 대한 자세한 내용은 17- 23페이지 "import 문"에서 볼 수 있습니다.

DogClass 및 ManClass가 MammalClass를 확장하기 때문에 이들 클래스는 이제 자동으로 MammalClass에 정의된 speak() 메소드에 액세스할 수 있습니다. 직접 특별히 speak()를 구현할 필요가 없습니다. speak() 메소드에 전달된 sound 변수 값은 DogClass 및 ManClass 생성자에 설정됩니다. DogClass 클래스는 다음과 같이 나타납니다.

```
package oop1;
import javax.swing.*;

public class DogClass extends MammalClass{

  public boolean hasTail() {
    return tail;
  }

  public void setTail(boolean value) {
    tail = value;
  }

  public DogClass() {
    setName("Snoopy");
    setSound("Woof, Woof!");
    setAge(2);
    setTail(true);
  }

  public void speed() {
  JOptionPane.showMessageDialog(null, "30 mph", "Dog Speed", 1);
  }

  private boolean tail;
}
```

ManClass는 다음과 같이 나타납니다.

```
package oop1;
import javax.swing.*;

 public class ManClass extends MammalClass {

  public boolean isMarried() {
    return married;
  }
```

```
  public void setMarried(boolean value) {
    married = value;
  }

  public ManClass() {
    setName("Steven");
    setEyeColor("Blue");
    setSound("Hello there! I'm " + this.getName() + ".");
    setAge(35);
    setMarried(true);
  }

  public void speed() {
  JOptionPane.showMessageDialog(null, "17 mph", "Man Speed", 1);
  }

  private boolean married;
}
```

새로운 두 개의 버튼 추가

예제 애플리케이션에 speak()와 speed() 메소드를 추가했다고 해도 애플리케이션에서는 아직까지 이를 호출하지 않습니다. 이를 변경하려면 다음과 같이 버튼 두 개를 Frame1.java 클래스에 추가해야 합니다.

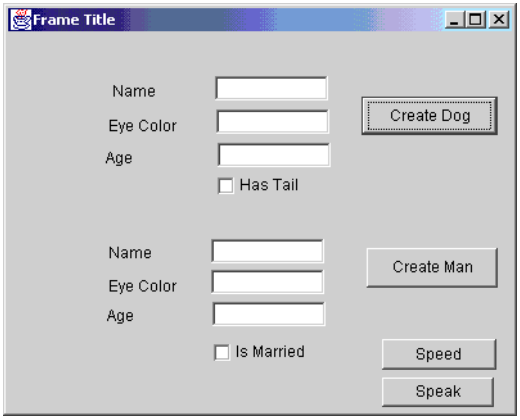
1 컨텐트 창의 Frame1 탭을 클릭합니다.

2 UI 디자이너를 표시하려면 Design 탭을 클릭합니다.

3 폼에 추가 버튼 두 개를 놓습니다.

4 Inspector에서 첫번째 버튼의 text 속성값을 Speed로 변경하고 두번째 버튼의 text 속성값을 Speak로 변경합니다.

Figure 17.2새 버전의 예제 애플리케이션에 Speed와 Speak 단추가 추가됩니다.



Source 탭을 클릭하여 Frame1.java 코드로 되돌아간 다음 다음에서 굵게 표시되는 코드를 다음과 같이 클래스 정의에 추가합니다.

```
// Create a reference for the objects
  DogClass dog;
  ManClass man;

//Create an Array of SoundInterface
SoundInterface soundList[] = new SoundInterface[2];

//Create an Array of Mammal
MammalClass mammalList[] = new MammalClass[2];
```

두 배열을 생성하는 코드를 추가하였는데 하나는 Mammal, 다른 하나는 SoundInterface용 배열입니다.

또한 dog과 man 객체에 대한 참조를 배열에 추가하는 Create Dog 및 Create Man 이벤트 핸들러에 코드를 추가합니다.

```
void button1_actionPerformed(ActionEvent e) {
  dog = new DogClass();
  txtfldDogName.setText(dog.getName());
  txtfldDogEyeColor.setText(dog.getEyeColor());
  txtfldDogAge.setText(Integer.toString(dog.getAge()));
  chkboxDog.setSelected(true);
  mammalList[0] = dog;
  soundList[0]  = dog;
}

void button1_actionPerformed(ActionEvent e) {
  man = new ManClass();
  txtfldManName.setText(man.getName());
  txtfldManEyeColor.setText(man.getEyeColor());
  txtfldManAge.setText(Integer.toString(man.getAge()));
  chkboxMan.setSelected(true);
  mammalList[1] = man;
  soundList[1]  = man;
  }
```

UI 디자이너로 되돌아가 Speed 버튼을 더블 클릭하고 JBuilder가 시작하는 이벤트 핸들러를 입력합니다. 코드는 다음과 같이 나타납니다.

```
void button1_actionPerformed(ActionEvent e) {
  for (int i = 0; i <= 1; i++) {
    mammalList[i].speed();
  }
}
```

코드는 배열(두 개 모두)에 있는 포유류 목록 전체를 순환하며 각 객체에 포유류의 속도를 표시할 것을 알립니다. 처음으로 목록을 돌 때 개가 속도를 표시하고 두번째로 목록을 돌 때 사람이 속도를 표시합니다. 이것이 바로 동일한 메시지를 수신하고 서로 다른 방식으로 이 메시지에 대응하는 분리되어 있으나 관련되어 있어 두개 객체의 행동에 있어서의 다형성입니다.

Speak 버튼의 코드는 매우 유사합니다.

```
void button1_actionPerformed(ActionEvent e) {
  for (int i = 0; i <= 1; i++) {
    soundList[i].speak();
  }
}
```

File|Save All을 선택하여 모든 변경 사항을 저장합니다.

편리한 경우 SoundInterface를 클래스로 처리할 수 있습니다. 인터페이스는 복잡성을 증가시키지 않고 다중 상속의 여러 이점을 제공한다는 점에 유의합니다.

애플리케이션 실행

이제 수정한 애플리케이션을 실행할 준비가 되었습니다. Run|Run Project를 선택하여 프로젝트를 다시 컴파일한 다음 실행합니다.

애플리케이션이 실행하기 시작하면 Create Dog 및 Create Man 버튼을 클릭하여 Speed 및 Speak 버튼을 누르거나 NullPointerException이 표시되기 전에 dog 및 man 객체를 생성합니다.

객체가 존재하고 Speed 버튼을 클릭할 경우 mammalList배열에 있는 첫 번째 포유류인 개의 속도를 보고하는 메시지 상자가 표시됩니다. OK를 클릭하여 메시지 상자를 없애면 두번째 메시지 상자가 표시되어 두번째 포유류인 사람의 속도를 보고합니다. Speak 버튼을 클릭하면 유사한 동작이 발생하지만 각 포유류가 내는 소리가 메시지로서 표시됩니다.

Java 패키지

코드를 재사용하려 할 경우 Java를 통해 사용자는 *패키지*라고 하는 논리 그룹에 여러 개의 클래스 정의를 그룹화할 수 있습니다. 예를 들어, 조직의 작업 프로세스를 모델링하는 사업 규칙 그룹을 만드는 경우 사용자는 클래스 정의를 하나의 패키지에 넣으려 할 것입니다. 패키지는 이전에 생성한 코드를 좀 더 쉽게 재사용 할 수 있도록 해줍니다.

import 문

Java 언어에는 미리 정의된 여러 패키지가 있습니다. 예를 들어, java.applet 패키지에는 Java 애플릿으로 작업하기 위한 클래스가 있습니다.

```
public class Hello extends java.applet.Applet {
```

이 코드는 Java 패키지 java.applet의 Applet이라는 클래스를 참조합니다. 이 클래스를 참조할 때마다 전체 클래스 이름 java.applet.Applet을 반복해야 한다면 꽤 비효율적임을 상상할 수 있습니다. 이 방법을 대신하는 대

안을 Java에서 제공합니다. 다음을 선택하여 자주 사용하게 될 패키지를 import할 수 있습니다.

```
import java.applet.*;
```

위의 문장은 컴파일러에 "인식할 수 없는 클래스 이름이 보이는 경우 java.applet 패키지를 살펴보라"고 말해줍니다. 이제 새 클래스를 선언할 경우 다음과 같이 할 수 있습니다.

```
public class Hello extends Applet {
```

이 방법이 더욱 간단합니다. 그러나 두 개의 클래스가 import된 두개의 패키지에서 정의하는 같은 이름을 갖는 경우 문제가 발생합니다. 이런 경우 전체 이름을 사용해야 합니다.

패키지 선언

패키지 생성은 패키지를 사용하는 것만큼이나 쉽습니다. 예를 들어 ,mypackage라는 패키지를 생성하려 할 경우 파일의 시작 부분에서 package 문장을 사용하면 됩니다.

```
package mypackage;

public class Hello extends java.applet.Applet {

 public void init() {
   add(new java.awt.Label("Hello World Wide Web!"));
  }

} // end class
```

이제 다음 문장과 함께 mypackage에 선언된 클래스에 액세스할 수 있습니다.

```
import mypackage.*;
```

이 파일은 mypackage라고 하는 하위 디렉토리에 있어야 한다는 것을 명심하십시오. 이렇게 함으로써 Java 컴파일러가 쉽게 패키지를 찾을 수 있습니다. JBuilder의 Project 마법사는 프로젝트 이름과 일치하도록 자동으로 디렉토리를 설정합니다. 또한 import한 패키지의 기본 디렉토리가 JBuilder IDE의 Source Path 또는 프로젝트의 Souce Path에 나열되어 있어야 합니다. 그러면 패키지를 다른 기본 디렉토리에 다시 놓으려 할 경우 이를 기억하기 쉽습니다.

JBuilder에서의 패키지 작업에 관한 자세한 내용은 *Building Applications with JBuilder*의 "Packages"를 참조하십시오.

Chapter

18

스레드 기법

스레드는 전체 Java 프로그램의 한 부분입니다. 스레드는 프로그램 안에서 하나의 순차적인 제어 흐름입니다. 스레드는 시작, 시퀀스 그리고 끝을 가집니다. 스레드는 자체로 실행되지 못 하고 프로그램 안에서 실행됩니다. 프로그램이 단일 시퀀스의 실행인 경우 스레드를 명시적으로 설정할 필요가 없고 VM(Java Virtual Machine)이 설정을 대신합니다.

자바 언어의 강력한 면 중의 하나는 동일한 프로그램 안에서 동시에 실행하는 다중 스레드를 간단하게 프로그램할 수 있다는 것입니다. 예를 들어, 웹 브라우저는 한 사이트에서 파일을 다운로드하면서 동시에 다른 사이트에 액세스할 수 있습니다. 브라우저가 두 작업을 동시에 할 수 없는 경우 다른 사이트를 탐색하기 전에 파일 다운로드를 마칠 때까지 기다려야 합니다.

Java VM는 항상 *데몬 스레드*라는 다중 스레드를 실행합니다. 예를 들어, 연속 실행되는 데몬 스레드는 가비지 컬렉션 작업을 수행합니다. 다른 데몬 스레드는 마우스와 키보드 이벤트를 처리합니다. 프로그램으로 Java VM 스레드 중의 하나를 잠그는 것이 가능합니다. 프로그램이 이벤트를 받지 않은 채 죽은 듯이 보이는 경우 스레드를 사용해 보십시오.

다음은 Java에서의 스레드 사용에 대한 내용을 상세히 다룬 책을 나열한 것입니다.

- *Java Threads, Second Edition*, Scott Oaks, Henry Wong 등 공저.
- *Taming Java Threads*, Allen Holub 저
- *Mastering Java Threads*, DDC Publishing

스레드의 수명 주기

모든 스레드는 분명한 수명 주기를 가집니다. 즉, 스레드는 시작, 중지, 실행 정지 및 이벤트 대기 등을 할 수 있으며 실행 중에 다른 스레드에게 공

지할 수 있습니다. 이 단원에서는 스레드 수명 주기의 좀더 일반적인 면에 대해 설명합니다.

run() 메소드 사용자 지정

스레드의 실행 동작을 구현하려면 run() 메소드를 사용합니다. 계산, 분류, 애니메이션 등과 같이 Java 문이 얻을 수 있는 것이라면 어떤 것이든지 이러한 동작이 될 수 있습니다.

다음 두 가지 기법 중 하나를 사용하여 스레드에 대해 run() 메소드를 구현할 수 있습니다.

- java.lang.Thread 클래스를 하위 분류합니다.
- java.lang.Runnable 인터페이스를 구현합니다.

스레드 클래스 하위 분류하기

새로운 클래스를 다른 스레드에서 생성하고 해당 클래스의 객체를 실행하고자 할 경우에는 java.lang.Thread 클래스를 하위 분류해야 합니다. Thread 클래스의 기본 run() 메소드는 아무 작업도 하지 않으므로 클래스는 run() 메소드를 오버라이드 해야합니다. run() 메소드는 스레드가 시작할 때 실행되는 첫 번째 메소드입니다.

예를 들어, CountThread같은 클래스는 Thread를 하위 분류하고 그 run() 메소드를 오버라이드입니다. 이 예제에서 run() 메소드는 스레드를 식별하고 그 이름을 화면에 표시합니다. for 루프는 start 값에서 finish 값까지 정수를 카운트하고 각 카운트를 화면에 표시합니다. 그 다음 루프가 실행을 종료하기 전에 메소드는 스레드가 실행을 종료했음을 가리키는 문자열을 화면에 표시합니다.

```
public class CountThread extends Thread {
  private int start;
  private int finish;

  public CountThread(int from, int to) {
    this.start = from;
    this.finish = to;
  }

  public void run() {
     System.out.println(this.getName()+ " started executing...");
     for (int i = start; i <= finish; i++) {
        System.out.print (i + " ");
     }
  System.out.println(this.getName() + " finished executing.");
  }
}
```

CountThread 클래스를 테스트하려면 다음과 같이 테스트 클래스를 만들 수 있습니다.

```
public class ThreadTester {
  static public void main(String[] args) {
```

```
        CountThread thread1 = new CountThread(1, 10);
        CountThread thread2 = new CountThread(20, 30);
        thread1.start();
        thread2.start();
    }
}
```

테스트 애플리케이션의 main() 메소드는 1에서 10까지 카운트하는 thread1과 20에서 30까지 카운트하는 thread2인 두 개의 CountThread 객체를 만듭니다. 두 스레드 모두 start() 메소드를 호출하여 시작됩니다. 이 테스트 애플리케이션의 출력은 다음과 같이 나타납니다.

```
Thread-0 started executing...
1 2 3 4 5 6 7 8 9 10 Thread-0 finished executing.
Thread-1 started executing...
20 21 22 23 24 25 26 27 28 29 30 Thread-1 finished executing.
```

출력에서 thread1과 thread2와 같은 스레드 이름는 표시되지 않는다는 것에 유의하십시오. 스레드에 이름을 지정하지 않는 한 Java는 Thread-n 형식의 이름을 스레드에게 자동적으로 부여하는데 여기서 n은 0에서 시작하는 고유 숫자입니다. 클래스 생성자 또는 setName(String) 메소드를 사용하여 스레드에 이름을 지정할 수 있습니다.

이 예제에서 Thread-0는 먼저 실행하고 먼저 종료합니다. 그러나 이 스레드는 먼저 시작하고 제일 나중에 종료하거나 부분적으로 시작하고 Thread-1에 의해 중단될 수도 있습니다. 이것은 Java의 스레드는 모든 시퀀스에서 실행이 보장되지 않기 때문입니다. 사실상 ThreadTester를 실행할 때마다 다른 출력을 얻을 수도 있습니다. 기본적으로 스레드 스케줄링의 과정은 프로그래머가 아니라 Java 스레드 스케줄러에 의해 제어됩니다.자세한 내용은 18- 7페이지 "스레드 우선 순위"를 참조하십시오.

Runnable 인터페이스 구현

기존 클래스의 객체가 자체 스레드에서 실행하도록 하려면 java.lang.Runnable 인터페이스를 구현할 수 있습니다. 이 인터페이스는 Thread 클래스로부터 상속하지 않는 클래스에 스레드 지원을 추가합니다. 이 인터페이스는 run()이라는 단 하나의 메소드만 제공하는데 이 메소드를 사용자의 클래스에 대해 구현해야 합니다.

참고 클래스가 Applet과 같은 Thread 이외의 클래스를 하위 분류하는 경우 Runnable 인터페이스를 사용하여 스레드를 만들어야 합니다.

Runnable 인터페이스를 구현하는 새로운 CountThread 클래스를 만들려면 CountThread 클래스의 정의를 변경해야 합니다. 다음과 같이 변경 사항이 굵게 강조된 채로 클래스 정의 코드가 나타납니다.

```
public class CountThread implements Runnable {
```

또한 스레드 이름을 알아내는 방법을 변경해야 합니다. 여기서 클래스 CountThread를 인스턴스화하지 않기 때문에 위의 경우에서 java.lang.Object인 CountThread의 수퍼클래스의 getName() 메소드를 호출할 수 없습니다. 이 메소드는 사용할 수 없습니다. 대신 getName() 메

소드와 조금 다른 형식으로 스레드 이름을 반환하는 Thread.currentThread() 메소드를 특별히 사용해야 합니다.

다음과 같이 변경 사항이 굵게 강조된 채로 전체 클래스가 나타납니다.

```
public class CountThread implements Runnable {
  private int start;
  private int finish;

  public CountThread(int from, int to) {
    this.start = from;
    this.finish = to;
  }

  public void run() {
    System.out.println(Thread.currentThread() + " started executing...");
    for (int i=start; i <= finish; i++) {
      System.out.print (i + " ");
    }
  System.out.println(Thread.currentThread() + " finished executing.");
  }
}
```

테스트 애플리케이션은 객체를 만드는 방법을 변경해야 합니다. CountThread를 인스턴스화하는 대신 애플리케이션은 새로운 클래스에서 Runnable 객체를 만들어 스레드 생성자 중의 하나로 이 객체를 전달합니다. 다음과 같이 변경 사항이 굵게 강조된 채로 코드가 나타납니다.

```
public class ThreadTester {
  static public void main(String[] args) {
    CountThreadRun thread1 = new CountThreadRun(1, 10);
    new Thread(thread1).start();
    CountThreadRun thread2 = new CountThreadRun(20, 30);
    new Thread(thread2).start();
  }
}
```

다음과 같이 이 프로그램의 출력이 나타납니다.

```
Thread[Thread-0,5,main] started executing...
1 2 3 4 5 6 7 8 9 10 Thread[Thread-0,5,main] finished executing.
Thread[Thread-1,5,main] started executing...
20 21 22 23 24 25 26 27 28 29 30 Thread[Thread-1,5,main] finished
executing.
```

여기서 Thread-0은 스레드 이름이고 5는 스레드가 만들어 질 때 부여 받은 우선 순위이며 main은 스레드가 할당되는 기본 ThreadGroup입니다. (지정된 것이 아무 것도 없을 경우 Java VM는 우선 순위와 그룹을 할당합니다.)

참조 사항 18- 7페이지 "스레드 우선 순위"
18- 8페이지 "스레드 그룹"

스레드 정의

Thread 클래스는 7개의 생성자를 제공합니다. 이러한 생성자는 다양한 방법으로 다음 세 개의 매개변수를 결합합니다.

- run() 메소드가 스레드 안에서 실행되는 Runnable 객체
- 스레드를 식별하는 String 객체
- 스레드를 지정할 ThreadGroup 객체 ThreadGroup 클래스는 관련 스레드 그룹을 구성합니다.

생성자	설명
Thread()	새로운 Thread 객체를 할당합니다.
Thread(Runnable target)	target을 run 객체로 갖도록 새로운 Thread 객체를 할당합니다.
Thread(Runnable target, String name)	target을 run 객체로 갖고 지정된 name을 자신의 이름으로 갖도록 새로운 Thread 객체를 할당합니다.
Thread(String name)	지정된 name을 자신의 이름으로 갖도록 새로운 Thread 객체를 할당합니다.
Thread(ThreadGroup group, Runnable target)	group에 의해 참조되는 스레드 그룹에 속하고 target를 자신의 실행 객체로 갖도록 새 Thread 객체를 할당합니다.
Thread(ThreadGroup group, Runnable target, String name)	target을 자신의 실행 객체로 갖고 지정된 name을 자신의 이름으로 가지며 group에 의해 참조되는 스레드 그룹에 속하도록 새로운 Thread 객체를 할당합니다.
Thread(ThreadGroup group, String name)	group에 의해 참조되는 스레드 그룹에 속하고 지정된 name을 자신의 이름으로 갖도록 새로운 Thread 객체를 할당합니다.

상태를 스레드에 연결하려면 스레드를 만들 때 ThreadLocal 객체를 사용합니다. 이 클래스를 통해 각 스레드는 사용자 또는 트랜잭션 ID와 같은 private 정적 변수를 자체에서 독립적으로 초기화한 복사본을 가질 수 있습니다.

스레드 시작

스레드를 시작하려면 start() 메소드를 호출합니다. 이 메소드는 스레드를 실행하는데 필요한 시스템 리소스를 만들고 스레드를 스케줄링하며 스레드의 run() 메소드를 호출합니다.

start() 메소드가 돌아온 후에 스레드는 실행하고 실행 가능한 상태에 있습니다. 대부분의 컴퓨터에는 하나의 CPU만이 있기 때문에 Java VM은 스레

드를 스케줄링해야 합니다.자세한 내용은 18- 7페이지 "스레드 우선 순위"을 참조하십시오.

스레드를 실행할 수 없게 만들기

스레드를 실행할 수 없는 상태로 만들려면 다음 기법 중의 하나를 사용합니다.

- sleep() 메소드: 이 메소드를 사용하면 실행하지 않은 시간을 초 및 10억분의 1초 단위로 지정할 수 있습니다.
- wait() 메소드: 이 메소드는 지정된 조건이 충족될 때까지 현재 스레드를 기다리도록 합니다.
- 입력 또는 출력에서 스레드를 막습니다.

스레드를 실행할 수 없는 경우 프로세서가 가용하더라도 스레드는 실행되지 않습니다. 실행할 수 없는 상태를 종료하려면 실행할 수 없는 상태로 들어가는 조건의 대부분과 맞아야 합니다. 예를 들어, sleep() 메소드를 사용했다면 초 단위의 지정된 시간은 경과했을 것입니다. wait() 메소드를 사용하였을 경우에는 다른 객체는 대기 스레드(notify() 또는 notifyAll()을 사용)에게 조건의 변경 사항을 알려야 합니다. 스레드가 입력 또는 출력에 의해 막힐 경우 입력 또는 출력을 종료해야 합니다.

또한 join() 메소드를 사용하여 스레드가 실행 중인 스레드를 종료하기를 대기하도록 합니다. 현재 대기 중인 스레드에 대해 이 메소드를 호출합니다. 매개변수를 밀리 초 단위로 메소드에 전달하여 스레드에 대한 타임아웃을 지정할 수 있습니다. join() 메소드는 타임아웃이 만료되거나 스레드가 종료할 때까지 스레드를 기다립니다. 이 메소드는 isAlive() 메소드와 연계하여 작동하는데, 스레드가 시작하고 중단하지 않으면 isAlive()는 true를 반환합니다.

suspend() 메소드와 resume() 메소드는 deprecated이었다는 것에 유의하십시오. suspend() 메소드는 데드록되기 쉽습니다. 대상 스레드가 일시중지되었을 때 중요한 시스템 리소스를 보호하는 모니터를 잠그고 있는 경우 대상 스레드가 재개될 때까지 어떤 스레드도 이 리소스에 액세스할 수 없습니다. 모니터는 한 번에 하나의 스레드가 객체에 대해 동기화된 메소드를 실행하고 있다는 것을 확인하기위해 사용되는 Java 객체입니다.자세한 내용은 18- 7페이지 "스레드 동기화"을 참조하십시오.

스레드 중지

더 이상 stop() 메소드를 사용하여 스레드를 중지할 수 없습니다. 이 메소드는 불안전할 때와 같이 deprecated 상태입니다. 스레드를 중지시키면 스레드가 잠근 모든 모니터의 잠금을 해제합니다. 이러한 모니터 중의 하나에 의해 이전에 보호받았던 객체가 일관성 없는 상태에 있는 경우 다른 스레드는 그 객체를 일관성 없는 것으로 여기게 됩니다. 그렇게 되면 프로그램이 손상될 수 있습니다.

스레드를 중단하려면 한정된 루프를 사용하여 run() 메소드를 종료합니다.

자세한 내용은 JDK 설명서에 있는 "Why are Thread.stop, Thread.suspend, Thread.resume and runtime.runFinalizersOnExit Deprecated?"라는 주제를 참조하십시오.

스레드 우선 순위

Java 스레드가 만들어지면 Java 스레드는 자신을 만든 스레드의 우선 순위를 상속합니다. setPriority() 메소드를 사용하여 스레드의 우선 순위를 설정할 수 있습니다. 스레드 우선 순위는 MIN_PRIORITY에서 MAX_PRIORITY(Thread 클래스에서 상수)의 범위 내의 정수 값으로 나타납니다. 최고 우선 순위를 가진 스레드가 실행됩니다.

스레드가 중단 또는 포기하거나 실행할 수 없게 되면 더 낮은 우선 순위의 스레드가 실행됩니다. 동일한 우선 순위의 스레드가 대기하고 있는 경우 Java 스캐줄러는 그 중에 하나를 골라 라운드로빈 방식으로 실행합니다. 스레드는 다음과 같은 상태가 될 때까지 실행합니다.

- 더 높은 우선 순위의 스레드가 실행할 수 있는 상태
- yield() 메소드를 사용하여 스레드가 포기하거나 그것의 run() 메소드가 종료되는 상태
- 스레드의 시간 할당이 만료된 상태 이것은 타임슬라이싱을 지원하는 시스템에만 적용됩니다.

이러한 타입의 스케줄링은 *고정 우선 순위 스케줄링*이라는 스케줄링 알고리즘에 기반합니다. 스레드는 다른 스레드와 비교했을 때 그 우선 순위에 기반하여 실행됩니다. 최고 우선 순위를 가진 스레드는 항상 실행 중입니다.

타임슬라이싱

일부 운영 체제는 *타임슬라이싱*이라는 스케줄링 메커니즘을 사용합니다. 타임슬라이싱은 CPU를 타임 슬롯으로 나눕니다. 하나 이상의 스레드가 종료하거나 최고 우선 순위 스레드가 실행할 수 있는 상태에 있을 때까지 시스템은 동등한 우선 순위를 갖고 있는 최고 우선 순위 스레드에게 실행 시간을 부여합니다. 타임슬라이싱을 모든 운영 체제에서 지원하지 않기 때문에 프로그램은 타임슬라이싱 스케줄링 메커니즘에 의존해선 안됩니다.

스레드 동기화

다중 스레드 계산의 핵심적인 문제 중의 하나는 하나 이상의 스레드가 동일한 데이터 구조에 액세스하는 상황을 처리하는 것입니다. 예를 들어, 한 스레드가 목록의 요소를 업데이트하는 동안 다른 스레드가 이들 요소를 동시에 분류하려 할 경우 프로그램은 데드록되거나 부적절한 결과를 발생시

킬 수 있습니다. 이러한 문제를 방지하려면 *스레드 동기화*를 사용해야 합니다.

두 객체가 동시에 동일한 메소드를 액세스하는 것을 방지하는 가장 간단한 방법은 스레드를 잠그는 것입니다. 스레드가 잠겨 있는 동안 잠금이 필요한 다른 스레드는 첫 번째 스레드가 잠금을 해제할 때까지 대기해야 합니다. 메소드를 *thread-safe*한 상태로 유지 하려면 한 번에 단 하나의 스레드에 의해 실행될 수 있는 메소드를 선언할 때 synchronized 키워드를 사용합니다. 또한 객체를 동기화할 수 있다는 것에 유의하십시오.

예를 들어, 지역 변수를 사용하는 값을 스와핑하는 swap() 메소드를 만들고 그 메소드를 실행하기 위해 다른 두 개의 스레드를 만드는 경우 프로그램은 부적절한 결과를 발생시킬 수 있습니다. 첫 번째 스레드는 Java 스케줄러 때문에 메소드의 처음 반만 실행할 수 있습니다. 그런 다음 첫 번째 스레드가 작업을 완료하지 않았기 때문에 틀린 값을 사용하기는 하지만 두 번째 스레드는 전체 메소드를 실행할 수 있습니다. 그러면 첫 번째 스레드는 메소드를 완료하기 위해 되돌아옵니다. 이 경우에 값의 스와핑은 발생하지 않는 듯이 보입니다. 이러한 문제의 발생을 방지하려면 메소드 선언에서 synchronized 키워드를 사용합니다.

기본적인 규칙으로 객체의 속성을 수정하는 모든 메소드는 synchronized로 선언되어야 합니다.

스레드 그룹

모든 Java 스레드는 *스레드 그룹*의 멤버입니다. 스레드 그룹은 여러 스레드를 단일 객체로 모아 모든 스레드를 한꺼번에 처리합니다. 스레드 그룹은 java.lang.ThreadGroup 클래스에 의해 구현됩니다.

런타임 시스템은 스레드 생성 동안 스레드를 스레드 그룹에 넣습니다. 스레드는 기본 그룹에 들어가거나 스레드를 만들 때 사용자가 지정하는 스레드 그룹에 들어갑니다. 한 번 스레드가 만들어지면 스레드를 새로운 그룹으로 옮길 수 없습니다.

생성자에서 그룹 이름을 지정하지 않고 스레드를 만드는 경우 런타임 시스템은 그 스레드를 만든 스레드와 동일한 그룹에 새 스레드를 놓습니다. 보통 지정되지 않은 스레드는 main 스레드 그룹에 들어갑니다. 그러나 애플릿에서 스레드를 만드는 경우 새 스레드는 애플릿이 실행 중인 브라우저 또는 뷰어에 따라 main이 아닌 스레드 그룹에 들어갑니다.

ThreadGroup을 사용하여 스레드를 생성하는 경우 그룹은 다음과 같습니다.

- 자체 생성 이름
- Java 런타임에 의해 생성된 그룹
- 애플릿이 실행 중인 애플리케이션에 의해 생성된 그룹

스레드가 그 일부인 그룹의 이름을 알려면 getThreadGroup() 메소드를 사용합니다. 일단 스레드의 그룹을 알면 어떤 다른 스레드가 그룹에 있는지 결정할 수 있고 그 스레드들을 한꺼번에 처리할 수 있습니다.

Chapter

19

직렬화

*객체 직렬화*는 언제라도 복구될 준비가 되어 있는 디스크 또는 기타 저장 장치에 전체 객체를 저장하는 과정을 말합니다. 객체를 복구하는 과정은 *비직렬화*라 합니다. 이 단원에서는 직렬화가 유용한 이유와 Java에서 직렬화와 비직렬화를 구현하는 방법에 대해 배웁게 됩니다.

직렬화된 객체는 *영구적*이라고 합니다. 메모리에 있는 대부분의 객체는 *일시적* 상태에 있는데 이것은 객체의 참조가 범위에서 벗어 나거나 컴퓨터 전원이 끊어지는 경우에 객체가 사라진다는 것을 의미합니다. 영구 객체는 디스크나 테이프 또는 ROM의 어딘가에 저장된 객체의 사본이 있는 한 존재합니다.

직렬화하는 이유는 무엇일까요?

일반적으로 디스크 또는 기타 저장 장치에 데이터를 저장하려면 특별한 데이터 형식을 정의하고 그 형식을 읽고 쓰는 함수 집합을 작성하며 파일 형식과 데이터 형식 간의 매핑을 만들어야 합니다. 데이터를 읽고 쓰는데 사용되는 함수는 간단하면서 확장성에 문제가 있거나 만들고 유지하기가 복잡하고 어렵습니다.

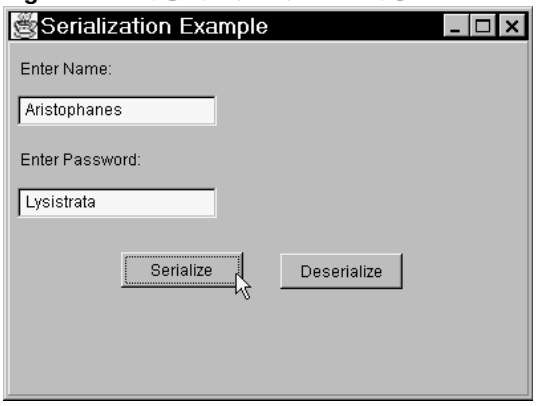
Java 전체는 객체와 객체 지향 프로그래밍에 기반하고 직렬화 형태로 객체에 대한 저장 메커니즘을 제공합니다. Java를 작업 방식대로 사용하면 파일 형식과 I/O(입력/출력)의 세부 사항에 대하여 더 이상 걱정할 필요가 없습니다. 대신 객체를 디자인하고 구현하여 실질적인 업무를 해결하는데 집중할 수 있습니다. 예를 들어, 클래스를 영구적으로 만들고 나중에 새로운 필드를 추가하는 경우 데이터를 읽고 쓰는 루틴을 수정하는 것에 대하여 걱정할 필요가 없습니다. 직렬화된 객체에서 모든 필드는 자동적으로 기록되고 복구됩니다.

Java 직렬화

직렬화는 JDK 1.1의 기능으로서 제일 먼저 시작되었습니다. Java의 직렬화 지원은 몇 개의 지원 클래스 및 인터페이스와 더불어 Serializable 인터페이스, ObjectOutputStream 클래스와 ObjectInputStream 클래스로 구성됩니다. 여기서 사용자 정보를 디스크에 저장하고 다시 읽을 수 있는 애플리케이션을 설명하면서 이러한 세 가지 항목을 모두 조사할 것입니다.

예를 들어, 아래에 표시된대로 특정 사용자에 대한 정보를 저장하려 한다고 가정해 봅시다.

Figure 19.1사용자 이름과 암호 저장



사용자가 본인의 이름과 암호를 적절한 필드에 입력한 다음에는 애플리케이션은 이 사용자에 대한 정보를 디스크에 저장해야 합니다. 물론 이것은 매우 간단한 예제이지만 사용자는 사용자 애플리케이션 환경 설정과 마지막으로 연 문서 등에 대한 데이터 저장을 쉽게 상상할 수 있습니다.

JBuilder를 사용하면 위에서 제시된 것과 같은 사용자 인터페이스를 디자인할 수 있습니다. 이 작업에 대한 도움말이 필요할 경우 *Building Applications with JBuilder*에서 "Designing a user interface"를 참조하십시오. Name 텍스트 필드 textFieldName와 암호 필드 passwordFieldName의 이름을 지정합니다. 여기서 볼 수 있는 두 가지 레이블 이외에 프레임 아래 근처에 세 번째 레이블을 추가하고 labelOutput이라는 이름을 지정합니다.

직렬화할 수 있는 인터페이스 사용

현재 사용자를 나타내는 새로운 클래스를 만듭니다. 새 클래스는 사용자 이름과 사용자 암호를 나타나는 속성을 가지고 있어야 합니다.

다음과 같은 방법으로 새로운 클래스를 만듭니다.

1 File|New Class를 선택하여 Class 마법사를 표시합니다.

2 Class Information 섹션에서 UserInfo로 새로운 클래스 이름을 지정합니다. 다른 필드는 변경하지 않은 상태로 남겨둡니다.

3 Options 섹션에서 Public 옵션과 Generate Default Constructor 옵션만 선택하고 다른 옵션은 모두 선택 해제합니다.

4 OK를 선택합니다.

Class 마법사는 새로운 클래스 파일을 만들어 프로젝트에 추가합니다. 생성된 코드를 다음과 같이 나타나도록 수정합니다.

패키지 직렬화는 다음과 같습니다.

```
public class UserInfo implements java.io.Serializable {
    private String userName = "";
    private String userPassword = "";

  public UserInfo() {
  }

  public String getUserName() {
    return userName;
  }

  public void setUserName(String s) {
    userName = s;
  }

  public String getUserPassword() {
    return userPassword;
  }

  public void setUserPassword(String s) {
    userPassword = s;
  }
}
```

위에서 사용자 이름을 유지하는 변수와 사용자 암호를 유지하는 다른 변수를 추가했습니다. 또한 두 필드 모드에 accessor 메소드를 추가했습니다.

여기서 UserInfo 클래스가 java.io.Serializable 인터페이스를 구현하는 것에 유의하십시오. Serializable은 구현할 메소드를 지정하기 때문에 *인터페이스 태그하기*로 알려져 있으나 단지 그 객체를 특정한 타입으로 "태그"합니다.

직렬화하려는 객체는 이러한 Serializable 인터페이스를 구현해야 합니다. 그렇지 않으면 이 장에서 나중에 사용되는 기술이 작동하지 않으므로 이 점은 중요합니다. 예를 들어, 이 인터페이스를 구현하지 않은 객체를 직렬화하려는 경우 NotSerializableException이 발생합니다.

여기서 java.io 패키지를 import하여 애플리케이션이 객체를 읽고 쓰는데 필요한 입력 및 출력 클래스와 인터페이스에 대한 액세스 권한을 가지도록 해야합니다. 다음 import 문을 프레임 클래스 상단에 추가합니다.

```
import java.io.*;
```

출력 스트림 사용

UserInfo 객체를 직렬화하기 전에 먼저 이 객체를 인스턴스화하고 사용자가 텍스트 필드에 입력한 값으로 설정해야 합니다. 사용자가 정보를 필드에 입력하고 Serialize 버튼을 클릭하면 사용자가 입력한 값은 다음과 같이 UserInfo 객체 인스턴스에 저장됩니다.

```
void jButton1_actionPerformed(ActionEvent e) {
    UserInfo user = new UserInfo();                  // instantiate a user object
    user.setUserName(textFieldName.getText());
    user.setUserPassword(textFieldPassword.getText());
}
```

JBuilder의 UI 디자이너를 사용하는 경우 Serialize 버튼을 더블 클릭하면 JBuilder는 jButton1_actionPerformed() 이벤트 코드를 시작합니다. 사용자 객체를 인스턴스화한 다음 user.setUserName() 메소드를 추가하면 user.setUserPassword() 메소드는 이벤트 핸들러를 호출합니다.

그런 다음 직렬화된 데이터를 포함하는 파일에 FileOutputStream을 엽니다. 이 예제에서 파일은 C:₩userInfo.ser이라고 합니다. 다음 코드를 Serialize 버튼 이벤트 핸들러에 추가합니다.

```
try {
  FileOutputStream file = new FileOutputStream("c:₩userInfo.ser");
```

객체를 직렬화하는 ObjectOutputStream을 만든 후에 다음 코드를 이벤트 핸들러에 추가하여 다음과 같이 ObjectOutputStream을 FileOutputStream에 보냅니다.

```
ObjectOutputStream out = new ObjectOutputStream(file);
```

이제 UserInfo 객체를 파일로 보낼 준비가 되었습니다. ObjectOutputStream's writeObject() 메소드를 호출하여 이 작업을 수행합니다. 객체가 실제로 파일에 기록되었음을 확인하려면 flush() 메소드를 호출하여 출력 버퍼를 플러시합니다.

```
out.writeObject(u);
out.flush();
```

출력 스트립을 닫아 파일 정보와 같이 스트림이 사용하는 리소스를 해제합니다.

```
    out.close();
}
```

파일에 기록하는 데 문제가 있거나 객체가 *Serializable* 인터페이스를 지원하지 않을 경우 IOException을 잡아내는 핸들러에 코드를 추가합니다.

```
catch (java.io.IOException IOE) {
     labelOutput.setText("IOException");
}
```

Serialize 버튼용 이벤트 핸들러의 전체 모습은 다음과 같이 나타납니다.

```
void jButton1_actionPerformed(ActionEvent e) {
    UserInfo user = new UserInfo();
    user.setUserName(textFieldName.getText());
    user.setUserPassword(textFieldPassword.getText());
  try {
  FileOutputStream file = new FileOutputStream("c:₩userInfo.ser");
       ObjectOutputStream out = new ObjectOutputStream(file);
       out.writeObject(user);
out.flush();
    }
catch (java.io.IOException IOE) {
      labelOutput.setText("IOException");
    }
finally {
    out.close();
    }
}
```

이제 프로젝트를 컴파일하고 실행합니다. Name 필드와 Password 필드에 값을 입력하고 Serialize 버튼을 클릭합니다. 객체를 텍스트 에디터에서 열어 그 객체가 기록되어 있다는 것을 확인할 수 있습니다. (객체를 편집하지 마십시오. 파일이 손상될 수도 있습니다!) 직렬화된 객체에는 다음과 같이 ASCII 텍스트와 바이너리 데이터가 혼합되어 들어 있음을 유의하십시오.

Figure 19.2직렬화된 객체

ObjectOutputStream 메소드

ObjectOutputStream 클래스에는 데이터를 스트림에 기록하는데 유용한 여러 가지 메소드가 들어 있습니다. 여기서 객체 쓰기에 한정되지 않습니다. writeInt(), writeFloat(), writeDouble() 등을 호출하면 모든 기본적인 타입이 스트림에 기록됩니다. 하나 이상의 객체 또는 기본 타입을 동일한 스트림에 기록하려면 이러한 메소드를 동일한 ObjectOutputStream 객체로 반복하여 호출하면 됩니다. 그러나 이러한 작업을 수행할 때 이 객체를 *동일한 순서로* 다시 읽어야 합니다.

입력 스트림 사용

이제 객체를 디스크에 기록했지만 어떤 방법으로 이 객체를 다시 얻을 수 있을까요? 사용자는 한 번 Deserialize 버튼을 클릭하여 데이터를 디스크에서 새로운 객체로 다시 읽어 들이기를 바랍니다.

새로운 FileInputStream 객체를 만들어 이 과정을 시작하여 방금 기록한 파일에서 읽을 수 있습니다. JBuilder를 사용하는 경우 UI 디자이너에서 Deserialize 버튼을 더블 클릭하고 JBuilder가 만든 이벤트 핸들러에서 다음과 같이 굵게 표시된 코드를 추가합니다.

```
void jButton4_actionPerformed(ActionEvent e) {
  try {
    FileInputStream file = new FileInputStream("c:₩userInfo.ser");
```

그 다음 해당 파일에서 객체를 읽는 기능을 제공하는 ObjectInputStream을 만듭니다.

```
ObjectInputStream input = new ObjectInputStream(file);
```

그 다음, ObjectInputStream.readObject() 메소드를 호출하여 그 파일에서 첫 번째 객체를 읽습니다. readObject()은 타입 Object를 반환하므로 이것을 적절한 타입 (UserInfo)로 변환할 수 있습니다.

```
UserInfo user = (UserInfo)input.readObject();
```

읽기를 마치면 ObjectInputStream을 닫아서 파일 디스크립터와 같은 관련 리소스를 모두 해제하는 것을 잊지 마십시오.

```
input.close();
```

마지막으로 UserInfo 클래스의 다른 객체에서와 같이 user 객체를 사용할 수 있습니다. 이 경우 대화 상자에 추가한 세 번째 레이블 필드에 다음과 같이 이름과 암호를 표시합니다.

```
labelOutput.setText("Name is " + user.getUserName() +
  ", password is: " +
  user.getUserPassword()};
```

파일에서 읽기으면 IOException을 발생시킬 수 있으므로 이 예외를 처리해야 합니다. 파일이 어떤 방식으로든지 손상되면 다음과 같이 StreamCorruptedException(IOException의 하위 클래스)이 또한 발생할 수 있습니다.

```
catch (java.io.IOException IOE) {
    labelOutput.setText("IOException");
}
```

사용자가 처리해야 하는 다른 예외가 있습니다. readObject() 메소드는 ClassNotFoundException을 발생시킬 수 있습니다. 구현하지 않은 객체를 읽으려고 하면 이러한 예외가 발생할 수 있습니다. 예를 들어, 이 객체가 다른 프로그램에 의해 작성되거나 파일이 작성된 이후에 UserInfo 클래스의 이름을 재지정한 경우 ClassNotFoundException이 발생합니다.

```
catch (ClassNotFoundException cnfe) {
  labelOutput.setText("ClassNotFoundException");
}
}
```

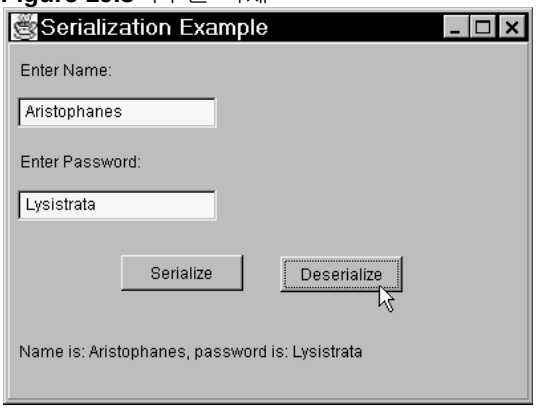
다음은 Deserialize 버튼 이벤트 핸들러의 전체 모습입니다.

```
void jButton2_actionPerformed(ActionEvent e) {
  try {
    FileInputStream file = new FileInputStream("c:₩userInfo.ser");
    ObjectInputStream input = new ObjectInputStream(file);
    UserInfo user = (UserInfo)input.readObject();
    input.close();
    labelOutput.setText("Name is " + user.getUserName() +
     ", password is: " +
```

```
        user.getUserPassword());
      }
    catch (java.io.IOException IOE) {
         labelOutput.setText("IOException");
     }
     catch (ClassNotFoundException cnfe) {
       labelOutput.setText("ClassNotFoundException");
     }
  }
```

이제 프로젝트를 컴파일하고 실행할 때 Name 값과 Password 값을 입력하고 Serialize 버튼을 클릭하여 정보를 디스크에 저장합니다. 그런 다음 Deserialize 버튼을 클릭하여 직렬화된 UserInfo 객체를 다시 읽습니다.

Figure 19.3복구된 객체



ObjectInputStream 메소드

ObjectInputStream은 writeDouble()과 writeFloat() 등과 같은 메소드의 상대 메소드인 readDouble()과 readFloat() 등과 같은 메소드를 또한 가집니다. 객체가 스트림에 기록된 방식과 같은 방식으로 각 메소드를 순차적으로 호출해야 합니다.

객체 스트림 쓰기 및 읽기

직렬화하고 있는 객체가 기본 타입이 아닌 다른 객체를 참조하는 필드를 포함하고 있을 때 사용자는 어떤 일이 발생하는지 궁금할 것입니다. 이 경우에 기본 객체와 보조 객체는 모두 스트림에 기록됩니다. 그러나 스트림에 기록된 두 객체는 모두 Serializable 인터페이스를 구현해야 한다는 것을 알아야 합니다. 두 객체가 위의 인터페이스를 구현하지 않는 경우 writeObject() 메소드를 호출하면 NotSerializableException이 발생합니다.

객체 직렬화로 잠재적인 보안 문제가 발생할 수 있음을 명심하십시오. 위의 예제에서 암호를 직렬화된 객체에 기록했습니다. 일부 환경에서 이 기술이 사용 가능할 수 있으나 객체를 직렬화하기로 선택할 때 보안 문제를 염두에 두십시오.

마지막으로 영구 객체를 만들되 기본 직렬화 메커니즘을 사용하지 않으려 할 경우 Serializable 인터페이스는 사용자 정의 직렬화 수행을 구현할 수 있는 writeObject()와 readObject()이란 두 가지 메소드를 제공합니다. 또한 Externalizable 인터페이스는 유사한 메커니즘을 제공합니다. 이러한 기술에 대한 내용은 JDK 설명서를 참조하십시오.

Chapter

20

Java Virtual Machine 소개

이 장에서는 JVM(Java Virtual Machine)을 소개합니다. JVM에 대한 기본 내용에 익숙해지는 것이 중요하며 최고급 Java 프로그래밍을 시작하지 않는 이상 JVM에 대하여 걱정할 필요는 없습니다. 이 장에서는 사용자를 위한 정보만을 제공합니다.

Java Virtual Machine에 대하여 알아보기 전에 이 장에서 사용하는 일부 전문 용어에 대하여 설명합니다. 먼저 JVM(Java Virtual Machine)은 Java 프로그램이 실행되는 환경을 말합니다. Java Virtual Machine 사양은 기본적으로 추상적인 컴퓨터를 정의하고 컴퓨터가 실행할 수 있는 명령을 지정합니다. 이러한 명령을 *바이트코드*라고 합니다. 일반적으로 Java 바이트코드와 JVM의 관계는 명령 집합과 CPU의 관계와 같습니다. 바이트코드는 Java 컴파일러가 생성하고 Java 인터프리터가 실행하는 바이트 길이의 명령입니다. 컴파일러가 .java 파일을 컴파일하면 일련의 바이트코드를 만들어내어 .class 파일에 저장합니다. 그러면 Java 인터프리터는 .class 파일에 저장된 바이트코드를 실행할 수 있습니다.

이 장에서 사용하는 다른 용어로는 Java 애플리케이션과 애플릿이 있습니다. 때로는 Java 애플리케이션과 Java 애플릿을 구별해야 합니다. 그러나 이 장의 일부 단원에서는 Java 애플리케이션과 Java 애플릿을 구별할 필요가 없습니다. 이러한 경우에는 Java 애플리케이션과 Java 애플릿을 모두 지칭하는 *app*를 사용하게 됩니다.

여기서는 Java란 실제로 무엇인가를 밝히는 것이 중요합니다. Java는 단순한 컴퓨터 언어가 아닌 컴퓨터 *환경*입니다. Java는 디자인 타임 Java(Java 언어 자체)와 런타임 Java(JVM)라는 두 개의 다른 주요 요소로 구성되는데 그 각각의 두 요소가 Java에 있어서 필수적인 부분이기 때문입니다. *Java*라는 단어에 대한 위와 같은 해석은 좀더 기술적인 것입니다.

재미있는 것은 *Java*의 실제적인 의미는 언어가 아니라 런타임 환경을 나타낸다는 것입니다. "이 시스템은 Java를 실행할 수 있다"라는 말이 실제로 의미하는 것은 그 시스템이 JRE(Java Runtime Environment)를 지원한

다는 것이고, 더 정확히 말하면 Java Virtual Machine을 구현한다는 의미입니다.

*Java Virtual Machine Specification*과 Java Virtual Machine의 *구현*은 분명히 구분되어야 합니다. JVM 사양은 JVM을 구현하는 방법을 정의한 문서(Sun의 웹 사이트 http://java.sun.com/docs/books/vmspec/index.html에서 참조)입니다. Java 구현이 정확하게 이 사양을 따를 경우 이 Java 구현에서 실행하는 Java app는 다른 JVM 구현에서 실행할 때와 동일한 결과를 산출하게 됩니다 이러한 JVM 사양은 Java 프로그램이 어떠한 플랫폼에서든지 실행할 수 있도록 해줍니다.

JVM 사양은 모든 플랫폼에서 구현될 수 있기 때문에 플랫폼 독립적입니다. JVM의 특정한 구현은 플랫폼 의존적임에 유의하십시오. 그 이유는 JVM 구현이 컴퓨터의 운영 체제(OS)와 직접적으로 작용하는 Java 부분이기 때문입니다. OS는 각각 다르므로 특정 JVM 구현은 특정 대상의 OS와 상호작용하는 방법에 대해 인지하고 있어야 합니다.

모든 JVM의 구현이 JVM 사양을 따르기 때문에 JVM의 구현 하에서 Java 프로그램을 실행하면 예상 가능한 런타임 환경을 만들 수 있습니다 다른 JVM 구현이 있더라도 이식성을 보장하려면 몇 가지 요구 사항이 충족되어야 합니다. 즉, 여러 구현 간에 다른 점이 있어도 이식성에는 영향을 미치지 않아야 합니다.

JVM은 다음과 같은 기능을 수행합니다.

- 생성된 객체에 메모리 할당
- 가비지 컬렉션 수행
- 리지스터 및 스택 작업 처리
- 장치 액세스와 같은 특정 기능을 위해 호스트 시스템을 호출
- Java app의 보안 모니터링

나머지 장 전체에서는 마지막 기능인 보안 기능에 초점을 맞춥니다.

Java VM 보안

JVM의 가장 중요한 역할 중의 하나는 Java app의 보안을 모니터링하는 것입니다. JVM은 특정한 메커니즘을 사용하여 보안 제한 사항을 Java app에 부과합니다. 이 메커니즘(또는 보안 모델)은 다음과 같은 역할을 합니다.

- 실행 중인 코드를 어느 정도까지 "신뢰"하는지 결정하고 코드에 적절한 액세스 수준을 지정합니다.
- 바이트코드가 부적합한 동작을 수행하지 않는지 확인합니다.
- 모든 바이트코드가 제대로 생성되는지 확인합니다.

다음 단원에서 이러한 보안 역할이 Java에서 처리되는 방법에 대하여 알아봅니다.

보안 모델

이 단원에서 Java 보안 모델에 있는 몇몇 다른 요소를 살펴봅니다. 특히 Java Verifier, Security Manager와 java.security 패키지, Class Loader의 역할에 대하여 조사합니다. 다음은 Java app를 안전하게 만드는 일부 컴포넌트입니다.

Java 인증자

클래스를 로드할 때마다 먼저 인증 과정을 거쳐야 합니다. 이러한 인증 과정의 주된 역할은 클래스의 각 바이트코드는 Java VM의 사양을 위반하지 않는다는 것을 확인하는 것입니다. 바이트코드 위반에 대한 예로 타입 오류와 오버플로(overflowed) 또는 언더플로(underflowed)된 산술 연산을 들 수 있습니다. 인증 과정은 Java 인증자(verifier)에 의해 처리되며 다음과 같은 네 단계로 구성됩니다.

1 클래스 파일 구조 인증

2 시스템 차원의 인증 작업 수행

3 바이트코드 확인

4 런타임 타입 및 액세스 확인 작업 수행

인증자의 첫 단계는 클래스 파일의 구조를 확인하는 것입니다. 모든 클래스 파일은 공통 구조를 공유하는데 예를 들어, 이들 파일들은 항상 값이 0xCAFEBABE인 *매직* 넘버로 시작해야 합니다. 이 단계에서 인증자는 또한 상수 풀(상수 풀은 클래스 파일의 문자열과 숫자가 저장되어 있는 장소)이 손상되었는지 확인합니다. 그 외에 인증자는 클래스 파일의 끝에 추가된 바이트가 없는지 확인합니다.

두 번째 단계는 시스템 차원의 인증 작업을 수행합니다. 이 단계에서 상수 풀에 대한 모든 참조의 유효성을 확인하고 클래스가 적절하게 하위 분류되었는지 확인합니다.

세 번째 단계에서는 바이트코드를 확인합니다. 이 단계는 전체 인증 과정에서 가장 중요하고 복잡한 단계입니다. 바이트코드를 확인한다는 것은 바이코드 타입이 유효하고 바이코트 인수가 적당한 숫자와 타입인지 확인하는 것을 의미합니다. 또한 인증자는 메소드 호출에서 올바른 인수 타입과 숫자를 전달했는지, 각 외부 함수가 적절한 타입을 반환했는지 확인합니다.

마지막 단계에서는 런타임을 확인합니다. 이 단계에서 외부적으로 참조된 클래스가 로드되고 그 메소드가 확인됩니다. 메소드 확인에서는 메소드 호출이 외부 클래스에 있는 메소드의 시그너처와 일치하는지 확인합니다. 또한 인증자는 클래스가 액세스 제한 사항을 위반하지 않았는지 확인하기 위해 현재 로드된 클래스에 의한 액세스 시도를 모니터합니다. 또 다른 액

세스 확인이 private 변수와 protected변수가 적합하게 액세스되지 않았다는 것을 확인하기 위해 변수에 대하여 수행됩니다.

이러한 지루한 확인 과정에서 Java 인증자가 보안 모델에서 얼마나 중요한지 알 수 있습니다. 또한 모든 컴파일러는 Java 바이트코드를 생성하도록 프로그램될 수 있기 때문에 확인 과정은 컴파일러 차원에서가 아니라 인증자 차원에서 이루어져야 한다는 것에 유의하십시오. 이 때, 컴파일러는 확인 과정을 통과하도록 프로그램될 수 있기 때문에 컴파일러에 의존하여 인증 과정을 수행하는 것은 매우 위험합니다. 이러한 점 때문에 JVM이 필요합니다.

Java 인증자에 대한 자세한 내용은 *Java Virtual Machine Specification* (http://java.sun.com/docs/books/vmspec/index.html)을 참조하십시오.

Security Manager 및 java.security 패키지

java.lang 패키지에 정의된 클래스 중의 하나로 SecurityManager 클래스가 있습니다. 이 클래스는 실행 중인 app가 어떤 위험한 작업을 수행할 권한을 갖고 있는지를 결정하기 위해 Java app에 대한 보안 정책을 확인합니다. 보안 정책의 주요 역할은 액세스 권한을 결정하는 것입니다. Java 1.1에서 SecurityManager 클래스는 오로지 보안 정책 설정에 대한 책임을 담당하지만 Java 2 이상에서는 새로운 java.security 패키지를 사용하여 훨씬 자세하고 강력한 보안 모델을 만들어졌습니다 . SecurityManager 클래스에는 "check"으로 시작되는 여러 가지 메소드가 있습니다. Java 1.1에서 이러한 "check" 메소드의 기본 구현은 SecurityException을 발생시키는 것입니다. Java 2 에서 "check" 메소드 대부분의 기본 구현은 SecurityManager.checkPermission() 메소드를 호출하고, 뒤이어 이 메소드의 기본 구현은 java.security.AccessController.checkPermission()을 호출합니다. 사용 권한을 확인하는 실제 알고리즘은 AccessController입니다.

SecurityManager 클래스에는 특정 작업의 허용 여부를 확인하는데 사용되는 많은 메소드가 있습니다. 예를 들어, checkRead() 메소드와 checkWrite() 메소드는 메소드 호출자가 특정한 파일에 대한 읽기 및 쓰기 작업을 수행할 권한을 가졌는지 여부를 확인합니다. 두 메소드는 checkPermission() 메소드를 호출하여 이 작업을 수행하고 뒤이어 AccessController.checkPermission()을 호출합니다. JDK에서 많은 메소드는 위험한 작업을 수행하기 전에 SecurityManager를 사용합니다. JDK는 레거시 때문에 이 작업을 수행하고 훨씬 제한된 보안 모델이 있었던 시기의 JDK의 초기 버전에는 SecurityManager가 있었습니다. app에서 SecurityManager 클래스(간접적으로 동일한 메소드를 호출)를 사용하는 대신 직접 AccessController.checkPermission()을 호출할 수 있습니다.

정적 System.setSecurityManager() 메소드를 사용하여 기본 보안 관리자를 환경에 로드할 수 있습니다. 이제 Java app가 위험한 작업을 수행해야 할 때마다 Java app는 환경에 로드된 SecurityManager 객체에 문의할 수 있습니다.

Java app가 SecurityManager 클래스를 사용하는 방법은 일반적으로 동일합니다. app가 시작될 때("-Djava.security.manager") 특별한 명령 줄

인수를 사용하거나 아니면 다음과 유사한 코드에서 SecurityManager의 인스턴스가 제일 먼저 만들어집니다.

```
SecurityManager security = System.getSecurityManager();
```

System.getSecurityManager() 메소드는 현재 로드된 SecurityManager의 인스턴스를 반환합니다. System.setSecurityManager() 메소드를 사용하는 어떤 SecurityManager도 설정되지 않은 경우 System.getSecurityManager()는 **null**을 반환하고, 그렇지 않으면 환경에 로드된 SecurityManager의 인스턴스를 반환합니다. 이제 app이 파일을 읽을 수 있는지 확인하려 한다고 가정합시다. 확인 작업은 다음과 같습니다.

```
if (security != null) {
  security.checkRead (fileName);
}
```

먼저 if 문은 SecurityManager 객체가 존재하는지 확인한 다음 checkRead() 메소드를 호출합니다. checkRead() 메소드가 작업을 허용하지 않는 경우 SecurityException이 발생하여 작업은 결코 수행되지 않지만 이 메소드가 작업을 허용할 경우에는 모두 잘 수행됩니다.

대부분의 Java 실행 브라우저는 자동으로 보안 관리자를 사용하기 때문에 애플릿이 실행 중일 경우 일반적으로 보안 관리자는 로드됩니다. 반면, 애플리케이션에서는 System.setSecurityManager() 메소드를 사용하는 환경에 보안 관리자가 로드되거나 애플리케이션을 시작할 때 명령 줄에서 로드되지 않으면 보안 관리자가 자동으로 사용되지 않습니다. 애플릿에 대한 동일한 보안 정책을 애플리케이션에 사용하려면 보안 관리자가 로드되었음을 확인해야 합니다.

자체 보안 정책을 지정하려면 java.security 패키지에서 클래스를 처리해야 합니다. 이 패키지에서 중요한 클래스는 Policy, Permission, AccessController 등 입니다. 마지막 수단일 경우를 제외하고 SecurityManager를 하류 분류하지 말아야 하고 이 점에 매우 주의를 기울여야 합니다. security 패키지에 대해서는 이 설명서에서 자세히 다루지 않습니다. 기본 보안 정책은 대부분의 초보 Java 개발자에게 충분합니다. 좀 더 고급 보안 문제에 관심이 있거나 java.security 패키지에 대한 자세한 내용은 JDK 설명서의 "Security Architecture"를 참조하십시오.

클래스 로더

클래스 로더는 Java app의 보안을 모니터하기 위해 보안 관리자와 함께 작동합니다. 다음은 클래스 로더의 주요 역할을 요약한 것입니다.

- 로드하려는 클래스가 이미 로드되었는지 결정합니다.
- 클래스 파일을 Virtual Machine에 로드합니다.
- 보안 정책에 따라 로드된 클래스에 지정된 권한을 결정합니다.
- 로드된 클래스에 대한 정보를 보안 관리자에게 제공합니다
- 클래스가 로드되는 경로를 결정합니다(System 클래스는 항상 BOOTCLASSPATH에서 로드).

클래스의 각 인스턴스는 추상 클래스 java.lang.ClassLoader의 하위 클래스의 인스턴스인 클래스 로더 객체와 관련있습니다. 클래스 로딩은 클래스가 인스턴스화될 때 자동적으로 발생합니다. ClassLoader 또는 기존 하위 클래스 중의 하나를 하위 분류하여 사용자 지정 클래스 로드를 만들 수 있으나 대부분의 경우에는 필요하지 않습니다. 클래스 로더 메커니즘에 대한 자세한 내용은 JDK 설명서의 java.lang.ClassLoader와 "Security Architecture"를 참조하십시오.

지금까지 Java app의 보안을 보장하기 위해 Java 인증자 SecurityManager와 클래스 로더가 어떻게 작동하는지에 대해 알아 보았습니다. 그 외에 Java app의 보안에 추가된 java.security 패키지의 메커니즘과 같은 이 장에서 설명하지 않은 다른 메커니즘이 있습니다. 또한 자바 언어 자체로 작성된 보안 기준이 있지만 이 장에서는 그 내용에 대해서는 다루지 않습니다.

Just-In-Time 컴파일러란 무엇입니까?

이 장에서 JIT 컴파일러(Just-In-Time)에 대한 간단한 설명을 포함하는 것이 적절합니다. JIT 컴파일러는 Java 바이트코드를 CPU가 직접 실행할 원시 시스템 명령으로 번역합니다. 이렇게 함으로써 Java app 성능이 확실히 향상됩니다. 원시 명령이 바이트코드 대신에 실행된다면 이전에 설명한 인증 과정에 어떤 일이 발생할까요? Java 인증자는 바이트코드가 번역되기 전에 바이트코드를 계속 확인하기 때문에 실제로 이 확인 과정은 바뀌지 않습니다.

Chapter

21

Java Native Interface(JNI)의 사용

이 장에서는 Java Native Method Interface(JNI)를 사용하여 Java 애플리케이션에서 원시 메소드를 호출하는 방법에 대해 설명합니다. JNI 작동 방법에 대해 먼저 설명한 다음 Java 메소드를 원시 메소드로 만드는 방법과 native 키워드에 대해 설명합니다. 마지막으로 Java 클래스용 C 헤더 파일을 생성하는데 사용되는 JDK의 **javah** 툴을 검사합니다.

Java 코드가 여러 플랫폼에서 실행되도록 디자인되었지만, 그 자체로 완벽하지 않은 상황이 있습니다.예를 들면, 다음과 같습니다.

- 표준 Java 클래스 라이브러리는 애플리케이션에 필요한 플랫폼 의존 기능을 지원하지 않습니다.
- 다른 언어에서 기존 라이브러리에 액세스하여 Java 코드에 액세스하기를 원할 경우
- 어셈블리처럼 더 낮은 레벨의 프로그램에서 구현하려는 코드를 갖춘 다음 Java 애플리케이션이 그 코드를 호출하도록 합니다.

Java Native Interface는 JDK에 포함된 표준 크로스 플랫폼 프로그래밍 인터페이스입니다. 이 인터페이스를 통해 C, C++, 어셈블리 등과 같은 다른 프로그래밍 언어로 작성된 애플리케이션 및 라이브러리와 함께 작동할 수 있는 Java 프로그램을 작성할 수 있습니다.

JNL을 사용하여 *Java 원시 메소드*를 작성하여 Java 객체(배열과 문자열 포함)를 생성, 조사, 업데이트하고, Java 메소드 호출, 예외 검색 및 쓰로우, 클래스 로드 및 클래스 정보 검색, 런타임 타입 확인등을 할 수 있습니다.

그 외에 Invocation API를 사용하여 Java Virtual Machine을 원시 애플리케이션에 포함시킨 다음, JNI 인터페이스 포인터를 사용하여 VM 기능에 액세스할 수 있습니다. 이를 통해 VM 소스 코드와 연결하지 않고 기존 애플리케이션에서 Java를 사용할 수 있습니다.

JNI 작동 방법

플랫폼 독립성이라는 Java의 주요 목적을 성취하기 위해 Sun사는 Java Virtual Machine의 구현을 표준화하지 않았습니다. 즉, Sun사는 JVM의 내부 아키텍처를 엄격하게 지정하지 않고 공급 업체가 JVM을 자체적으로 구현하도록 하였습니다. 모든 JVM 구현이 플랫폼 독립성을 가지는데 필요한 표준(예: .class 파일의 표준 구조)을 준수해야 하므로 위와 같이 함으로써 Java가 플랫폼 독립형이 되도록 했습니다.

이러한 시나리오가 갖는 유일한 문제는 런타임 시스템이 다양한 JVM 구현에 따라 다르기 때문에 Java app에서 원시 라이브러리에 액세스하는 것이 어렵게 된다는 점입니다. 그러한 이유에서 Sun사는 Java 애플리케이션에서 원시 라이브러리에 액세스하는 표준 방식으서 JNI를 내놓았습니다.

원시 메소드가 Java 애플리케이션에서 액세스되는 방법은 JDK 1.1에서 변경되었습니다. 이전 방법을 사용하면 Java 클래스가 원시 라이브러리에서 메소드를 직접 액세스할 수 있습니다. 새로운 구현에서는 JNL을 Java 클래스와 원시 라이브러리 간의 중간 층으로 사용합니다. JVM이 원시 메소드를 직접 호출하도록 하는 대신 JVM은 JNI에 대한 포인터를 사용하여 메소드를 호출합니다. JNI에 대한 포인터를 사용하여 메소드를 호출하기 때문에 JVM 구현이 다르더라도 원시 메소드(JNI)에 액세스하는데 사용하는 레이어는 항상 동일하게 됩니다.

원시 키워드 사용

매우 쉽게 Java 메소드를 원시 메소드로 만들 수 있습니다. 다음은 필요한 단계를 요약한 것입니다.

1 메소드의 몸체를 삭제합니다.

2 메소드 시그너처의 끝에 세미콜론을 추가합니다.

3 메소드 시그너처의 시작 부분에 native 키워드를 추가합니다.

4 메소드의 몸체를 런타임에 로드될 원시 라이브러리에 포함합니다.

예를 들어, 다음과 같은 메소드가 Java 클래스에 있다고 가정합니다.

```
public void nativeMethod () {
  //the method's body
}
```

다음은 원시 메소드를 만드는 방법입니다.

```
public native void nativeMethod ();
```

이제 원시 메소드를 선언하였으므로 실제 구현이 원시 라이브러리에 포함될 것입니다. 라이브러리를 호출하여 구현이 전역적으로 사용 가능하도록 하는 것이 이 메소드를 멤버로 하는 클래스의 임무입니다. 클래스가 라이브러리를 호출하도록 만드는 가장 쉬운 방법은 다음을 클래스에 추가하는 것입니다.

```
정적
{
  System.loadLibrary (nameOfLibrary);
}
```

일단 클래스가 먼저 로드되면 static 코드 블록은 항상 실행됩니다. 가상으로 static 코드 블록에 어떤 것이라도 포함할 수 있습니다. 그러나 라이브러리 로딩이 가장 일반적인 용도입니다. 어떤 이유에서 라이브러리를 로드하지 못할 경우 라이브러리에서 메소드가 한 번 호출되면 UnsatisfiedLineError 예외가 발생합니다. JVM은 그 이름에 올바른 확장명(Windows에서는 .dll이고 UNIX에서는 .so)을 추가하므로 라이브러리 이름에서 확장명을 지정할 필요가 없습니다.

javah 툴 사용

JDK는 Java 클래스용 C 헤더 파일을 생성하는데 사용되는 javah라는 툴을 제공합니다. 다음은 javah를 사용하는 일반적인 구문입니다.

javah [options] *className*

여기서 className은 C 헤더 파일을 생성하려는 클래스 파일의 이름(.class 확장명이 빠진)을 나타냅니다. 명령 줄에서 하나 이상의 클래스를 지정할 수 있습니다. 각 클래스에 대하여 javah는 기본적으로 .h 파일을 클래스의 디렉토리에 추가합니다. .h 파일을 다른 디렉토리에 넣으려면 -o 옵션을 사용합니다. 클래스가 패키지 안에 있으면 패키지를 클래스 이름과 함께 지정해야 합니다.

예를 들어, 패키지 myPackage에서 클래스 myClass에 대한 헤더 파일을 생성하려면 다음을 수행합니다.

javah myPackage.myClass

생성된 헤더 파일에는 패키지 이름(myPackage_myClass.h)이 포함됩니다.

다음은 javah 옵션의 일부 목록입니다.

옵션	설명
-jni	JNI 헤더 파일을 만듭니다
-verbose	진행 정보를 표시합니다
-version	javah 버전을 표시합니다
-o directoryName	지정된 디렉토리의 .h 파일을 출력합니다
-classpath 경로	기본 클래스 경로를 오버라이드합니다

javah에 의해 생성된 .h 파일의 내용에는 클래스의 native 메소드에 대한 모든 함수 프로토타입이 포함됩니다. 프로토타입을 수정하여 Java 런타임 시스템이 원시 메소드를 찾아 호출할 수 있습니다. 이러한 수정에는 원시 메소드 호출을 위해 만들어진 이름 지정 규칙에 따른 메소드 이름 변경이 기본적으로 포함됩니다. 수정된 이름은 클래스 및 메소드 이름에 대한 접

두사 Java_를 포함합니다. 따라서, myClass라는 클래스 안에 nativeMethod라는 원시 메소드를 갖고 있는 경우 myClass.h 파일에 나타나는 이름은 Java_myClass_nativeMethod입니다.

JNI에 대한 자세한 내용은 다음을 참조하십시오.

- Java Native Interface (http://java.sun.com/j2se/1.3/docs/guide/jni/)
- Java Native Interface Specification (http://java.sun.com/j2se/1.3/docs/guide/jni/spec/jniTOC.doc.html)
- *Java Tutorial*, "Trail: Java Native Interface" (http://java.sun.com/docs/books/tutorial/native1.1/index.html)

Java Native Interface에 대해 다음과 같은 책들을 참조하십시오.

- *Java Native Interface: Programmers Guide and Specification (Java Series)*, Sheng Liang 저
  - amazon.com (http://www.amazon.com/exec/obidos/ASIN/0201325772/inprisecorporati/107-5674331-0009332)
  - fatbrain.com (http://www1.fatbrain.com/asp/bookinfo/bookinfo.asp?theisbn=0201325772&from=SWP279)
- *Essential Jni: Java Native Interface(Essential Java)*, Rob Gordon 저
  - amazon.com (http://www.amazon.com/exec/obidos/ASIN/0136798950/inprisecorporati/107-5674331-0009332)
  - fatbrain.com (http://www1.fatbrain.com/asp/bookinfo/bookinfo.asp?theisbn=0136798950&from=SWP279)

Chapter

22

자바 언어 빠른 참조

Java 2 플랫폼 에디션

Java 2 Platform은 다양한 목적으로 사용되는 몇 가지 에디션에서 사용할 수 있습니다. Java는 장소와 플랫폼에 제한 없이 어디서든 실행될 수 있는 언어이므로 다양한 환경에서 사용되며 Java 2 Standard Edition(J2SE), Java 2 Enterprise Edition(J2EE)과 Java 2 Micro Edition(J2ME)과 같은 여러 개의 에디션에서 패키지됩니다.기업용 애플리케이션 개발에서와 같은 경우에는 더 큰 패키지들이 사용됩니다.소비재 전자 제품과 같은 그 밖의 경우에는 언어의 극히 일부분만 사용됩니다.각 에디션에는 애플리케이션 개발에 사용되는 Java 2 Software Development Kit(SDK)와 애플리케이션 실행에 사용되는 Java 2 Runtime Environment(JRE)가 들어 있습니다.

Table 22.1 Java 2 Platform 에디션

Java 2 Platform	약어	설명
Standard Edition	J2SE	자바 언어의 핵심인 클래스를 포함합니다.
Enterprise Edition	J2EE	기업용 애플리케이션 개발을 위한 J2SE 클래스와 추가 클래스를 포함합니다.
Micro Edition	J2ME	J2SE 클래스의 서브셋을 포함하며 소비재 전자 제품에서 사용됩니다.

Java 클래스 라이브러리

대부분의 프로그래밍 언어와 마찬가지로 Java도 미리 구축된 라이브러리에 의존하여 특정 기능을 지원합니다.자바 언어에서는 패키지라고 하는 관련된 이러한 클래스 그룹은 Java 에디션에 따라 다양합니다.각 에디션은 일반 애플리케이션, 기업용 애플리케이션, 소비재 제품 등과 같은 특정 목적에 따라 사용됩니다.

Java 2 Platform, Standard Edition(J2SE)은 개발자에게 풍부한 기능을 가진 안정되고 안전한 크로스 플랫폼 개발 환경을 제공합니다.이 Java 에디션은 데이터베이스 연결, 사용자 인터페이스 디자인, 입력/출력, 네트워크 프로그래밍과 같은 핵심 기능을 제공하고 자바 언어의 기본 패키지를 포함합니다. 다음 표는 이러한 J2SE 패키지 중 일부를 나열한 것입니다.

Table 22.2 J2SE 패키지

패키지	패키지 이름	설명
언어	java.lang	자바 언어의 주요 핵심을 포함하는 클래스.
유틸리티	java.util	유틸리티 데이터 구조 지원.
I/O	java.io	다양한 입출력 타입 지원.
텍스트	java.text	지역화는 텍스트, 날짜, 숫자, 메시지에 대한 처리를 지원합니다.
산술	java.math	임의 정밀도 정수 및 부동 소수점 산술을 위한 클래스
AWT	java.awt	사용자 인터페이스 디자인 및 이벤트 처리.
Swing	javax.swing	모든 플랫폼에서 유사하게 동작하는 모든 Java, lightweight 컴포넌트를 위한 클래스.
Javax	javax	자바 언어에 대한 확장.
애플릿	java.applet	애플릿 생성에 필요한 클래스.
Beans	java.beans	JavaBeans 개발을 위한 클래스.
리플렉션	java.lang.reflect	런타임 클래스 정보를 얻는 데 사용되는 클래스.
SQL	java.sql	데이터베이스의 데이터 액세스 및 처리 지원.
RMI	java.rmi	분산 프로그래밍 지원.
네트워킹	java.net	네트워킹 애플리케이션의 개발을 지원하는 클래스.
보안	java.security	암호 보안 지원.

키워드

이러한 테이블에는 다음과 같은 타입의 키워드가 들어 있습니다.

- 데이터와 반환 타입 및 조건

- 패키지, 클래스, 멤버, 인터페이스
- 액세스 변경자
- 루프와 루프 제어
- 예외 처리
- 예약어

데이터와 반환 타입 및 조건

타입과 조건	키워드	용도
숫자 타입	byte	1바이트, 정수
	double	8바이트, 배정도
	float	4바이트, 단정도
	int	4바이트, 정수
	long	8바이트, 정수
	short	2바이트, 정수
	strictfp	(지정 시) 부동 소수점 중간 계산에서 표준 정밀도를 사용하는 메소드 또는 클래스.
기타 타입	boolean	부울 값
	char	16비트, 1문자
반환 조건	return	결과값을 가진 현재 코드 블록을 종료합니다.
	void	반환 값이 필요 없는 타입을 반환합니다.

패키지, 클래스, 멤버, 인터페이스

키워드	용도
abstract	사용하려면 이 메소드 또는 클래스를 확장해야 합니다.
class	Java 클래스를 선언합니다.
extends	하위 클래스를 생성합니다. 클래스가 다른 클래스의 public 및 protected 멤버에 액세스 할 수 있도록 해줍니다. 하나의 인터페이스에서 또다른 인터페이스를 상속할 수 있도록 해줍니다.
final	이 클래스는 확장되지 않습니다.
implements	클래스 정의에서 정의된 인터페이스를 구현합니다.
import	import된 클래스 또는 패키지의 모든 클래스를 현재 프로그램에서 볼 수 있게 만듭니다.
instanceof	객체의 상속을 점검합니다.
interface	구현으로부터 클래스의 인터페이스를 추상화합니다(작업 방법이 아닌 작업 대상을 알려줌).
native	이 메소드의 몸체는 원시 라이브러리 링크에서 제공됩니다.
new	클래스를 인스턴스화합니다.
package	동일한 패키지 선언으로 소스 파일에 정의된 모든 클래스에 대한 패키지 이름을 선언합니다.

키워드	용도
static	하나의 객체에서 뿐만이 아닌 전체 클래스에서 멤버를 사용할 수 있습니다.
super	하위 클래스안에서 슈퍼 클래스를 참조합니다.
synchronized	스레드에 대해 안전한 코드 블록을 만듭니다.
this	현재 객체를 참조합니다.
transient	이 변수 값은 객체가 저장된 경우에는 유지되지 않습니다.
volatile	이 변수 값은 갑자기 변경될 수 있습니다.

액세스 변경자

키워드	용도
package	기본 액세스 수준. 명시적으로는 사용하지 마십시오. 다른 패키지로 하위 클래스를 만들 수 없습니다.
private	멤버 고유의 클래스로 제한된 액세스
protected	멤버 클래스의 패키지로 제한된 액세스
public	클래스: 어디서든 액세스할 수 있습니다. 하위 클래스: 클래스에 액세스할 수 있으면 하위 클래스에도 액세스할 수 있습니다.

루프와 루프 제어

문장 타입	키워드
선택문	if
	else
	switch
	case
중단문	break
대체문	default
반복문	for
	do
	while
	continue

예외 처리

키워드	용도
throw	메소드 제어를 예외 핸들러에 전송합니다.
throws	메소드가 표시할 수 있는 예외들을 나열합니다.
try	예외 처리문 열기
catch	예외를 포착합니다.
finally	프로그램을 종료하기 전에 코드를 실행합니다.

예약

키워드	용도
assert	차후 사용을 위해 예약
const	차후 사용을 위해 예약
goto	차후 사용을 위해 예약

데이터 타입 변환 및 타입 변환

다른 타입이 필요할 때 한 번의 작업을 위해 객체 또는 변수의 데이터 타입을 변경할 수 있습니다. 작은 클래스 또는 데이터 타입을 보다 큰 클래스 또는 타입으로 확장하는 변환 작업을 암시적으로 수행해도 되지만 가급적이면 명시적으로 변환하는 것이 바람직합니다. 축소 변환은 명시적으로 변환되거나 *타입 변환*되어야 합니다. 신참 프로그래머의 경우 많은 오류 및 혼동의 원인이 될 수 있으므로 타입 변환을 피해야 합니다.

축소 타입 변환의 경우 (int)x에서 처럼 변환할 *to*타입을 괄호 안에 지정한 다음 변환할 변수를 지정합니다. 다음은 컨텍스트에서 보여지는 것으로, 여기서 x는 변환 중인 변수이고 float는 원래 데이터 타입이며 int는 대상 데이터 타입이고 y는 새 값을 저장한 변수입니다.

```
float x = 1.00;    //declaring x as a float
int y = (int)x; //casting x to an int named y
```

위에서 x의 값은 int의 내부에 꼭 맞다고 가정합니다.x의 소수점 값이 변환 중 없어졌다는 것에 유의하십시오. Java는 소수를 가장 가까운 정수로 반올림합니다.

유니코드 시퀀스는 숫자, 글자, 기호 또는 줄 구분 또는 탭과 같은 인쇄되지 않은 문자를 나타낼 수 있습니다. 유니코드에 대한 더 자세한 내용은 http://www.unicode.org/를 참조하십시오.

이 단원에는 다음과 같은 변환표가 들어 있습니다.

- 기본에서 기본으로(Primitive to primitive) 변환
- 기본에서 문자열로(Primitive to String) 변환
- 기본에서 참조로(Primitive to reference) 변환
- 문자열에서 기본으로(String to primitive) 변환
- 참조에서 기본으로(Reference to primitive) 변환
- 참조에서 참조로(Reference to reference) 변환

기본에서 기본으로 (Primitive to primitive) 변환

Java는 부울 값 변환을 지원하지 않습니다. Java의 엄격한 논리적 타입 기능을 사용하려면 변수에 적절한 값을 할당한 다음 변환해야 합니다. 0과 1은 false와 true값을 나타내는 데 자주 사용됩니다.

구문	주석
다른 기본(primitive) 타입 p에서 boolean t로 변환: `t = p != 0;`	다른 기본(primitive) 타입에는 byte, short, char, int, long, double, float가 있습니다.
boolean t에서 byte b로 변환: `b = (byte)(t ? 1 : 0);`	
boolean t에서 int, long, double, 또는 float m으로 변환: `m = t ? 1 : 0;`	
boolean t에서 short s로 변환: `s = (short) (t ? 1 : 0);`	
boolean t에서 byte b로 변환: `b = (byte) (t?1:0);`	
boolean t에서 char c로 변환: `c = (char) (t?'1':'0');`	단일 인용 부호를 생략해도 되지만 가급적이면 사용하는 것이 좋습니다.
short, char, int, long, double 또는 float n에서 byte b로 변환: `b = (byte)n;`	
byte b에서 short, int, long, double 또는 float n으로 변환: `n = b;`	
byte b에서 char c로 변환: `c = (char)b;`	

기본에서 문자열로(primitive to String) 변환

기본(primitive) 데이터 타입은 변환할 수 있지만 참조 타입은 변환할 수 없는 객체입니다. 참조 타입 변환은 위험합니다.

Java는 부울 값 변환을 지원하지 않습니다. Java의 엄격한 논리적 타입 기능을 사용하려면 변수에 적절한 값을 할당한 다음 변환해야 합니다. 0과 1은 false와 true 값을 나타내는 데 자주 사용됩니다.

구문	주석
boolean t에서 String gg로 변환: `gg = t ? "true" : "false";`	

byte b에서
String gg로 변환:
`gg = Integer.toString(b);`
또는
`gg = String.valueOf(b);`
	다음은 해당되는 곳에서 toString이 대체됩니다.
	`toBinaryString`
	`toOctalString`
	`toHexString`
	10이나 2가 아닌 8과 같은 베이스를 사용하는 경우:
	`gg = Integer.toString(b, 7);`

short 또는 int n에서
String gg로 변환:
`gg = Integer.toString(n);`
또는
`gg = String.valueOf(n);`
	다음은 해당되는 곳에서 toString이 대체됩니다.
	`toBinaryString`
	`toOctalString`
	`toHexString`
	10이 아닌 8과 같은 베이스를 사용하는 경우:
	`gg = Integer.toString(n, 7);`

char c에서 String gg로 변환: `gg = String.valueOf(c);`	

long n에서
String gg로 변환:
`g = Long.toString(n);`
또는
`gg = String.valueOf(n);`
	다음은 해당되는 곳에서 toString이 대체됩니다.
	`toBinaryString`
	`toOctalString`
	`toHexString`
	10이나 2가 아닌 8과 같은 베이스를 사용하는 경우:
	`gg = Integer.toString(n, 7);`

구문	주석

float f에서
String gg로 변환:
gg = Float.toString(f);
또는
gg = String.valueOf(f);
소수 보호 또는 과학적 표기법에 대해서는 다음 열을 참조하십시오.
	이러한 타입 변환은 더 많은 데이터를 보호합니다.
	배정도:
	java.text.DecimalFormat df2
	= new
	java.text.DecimalFormat("###,##0.00");
	gg = df2.format(f);
	과학적 표기법(지수 보호) (JDK 1.2.x 이상):
	java.text.DecimalFormat de
	= new
	java.text.DecimalFormat("0.000000E00");
	gg = de.format(f);

double d에서
String gg로 변환:
gg = Double.toString(d);
또는
gg = String.valueOf(d);
소수 보호 또는 과학적 표기법에 대해서는 다음 열을 참조하십시오.
	이러한 타입 변환은 더 많은 데이터를 보호합니다.
	배정도:
	java.text.DecimalFormat df2
	= new
	java.text.DecimalFormat("###,##0.00");
	gg = df2.format(d);
	과학적 표기법(JDK 1.2.x 이상):
	java.text.DecimalFormat de
	= new
	java.text.DecimalFormat("0.000000E00");
	gg = de.format(d);

기본에서 참조로(Primitive to reference) 변환

Java는 기본(primitive) 데이터 타입에 해당하는 클래스와 변환을 도와주는 메소드를 제공합니다.

기본(primitive) 데이터 타입은 변환할 수 있지만 참조 타입은 변환할 수 없는 객체라는 것에 유의하십시오. 참조 타입 변환은 위험합니다.

Java는 부울 값 변환을 지원하지 않습니다. Java의 엄격한 논리적 타입 기능을 사용하려면 변수에 적절한 값을 할당한 다음 변환해야 합니다.0과 1은 false와 true 값을 나타내는 데 자주 사용됩니다.

구문	설명
boolean t에서 Boolean tt로 변환: tt = new Boolean(t);	
기본(primitive) 타입 p(boolean이외)에서 Boolean tt로 변환: tt = new Boolean(p != 0); char의 경우 다음 열을 참조하십시오.	char c의 경우 단일 인용 부호 안에 0을 놓습니다. tt = new Boolean(c != '0');
boolean t에서 Character cc로 변환: cc = new Character(t ? '1' : '0');	

구문	설명
byte b에서 Character cc로 변환: cc = new Character((char) b);	
char c에서 Character cc로 변환: cc = new Character(c);	
short, int, long, float, 또는 double n에서 Character cc로 변환: cc = new Character((char)n);	
boolean t에서 Integer ii로 변환: ii = new Integer(t ? 1 : 0);	
byte b에서 Integer ii로 변환: ii = new Integer(b);	
short, char, 또는 int n에서 Integer ii로 변환: ii = new Integer(n);	
long, float, 또는 double f에서 Integer ii로 변환: ii = new Integer((int) f);	
boolean t에서 Long nn으로 변환: nn = new Long(t ? 1 : 0);	
byte b에서 Long nn으로 변환: nn = new Long(b);	
short, char, int, 또는 long s에서 Long nn으로 변환: nn = new Long(s);	
float, double f에서 Long nn으로 변환: nn = new Long((long)f);	
boolean t에서 Float ff로 변환: ff = new Float(t ? 1 : 0);	
byte b에서 Float ff로 변환: ff = new Float(b);	
short, char, int, long, float, 또는 double n에서 Float ff로 변환: ff = new Float(n);	

구문	설명
boolean t에서 Double dd로 변환: dd = new Double(t ? 1 : 0);	
byte b에서 Double dd로 변환: dd = new Double(b);	
short, char, int, long, float 또는 double n 에서 Double dd로 변환: dd = new Double(n);	

문자열에서 기본으로(String to primitive) 변환

기본(primitive) 데이터 타입은 변환할 수 있지만 참조 타입은 변환할 수 없는 객체라는 것에 유의하십시오. 참조 타입 변환은 위험합니다.

Java는 부울 값 변환을 지원하지 않습니다.Java의 엄격한 논리적 타입 기능을 사용하려면 변수에 적절한 값을 할당한 다음 변환해야 합니다. 숫자 0과 1, 문자열 "true" 와 "false" 또는 이에 상응하는 다른 값들을 사용하여 true와 false 값을 표현합니다.

구문	설명
String gg에서 boolean t로 변환: t = new Boolean(gg.trim()).booleanValue();	**주의 :** t는 gg의 값이 "true"(대소문자 구분 안함)일 때만 true입니다. 이 문자열이 "1", "yes" 또는 이에 상응하는 다른 긍정 문자열일 경우 이 변환은 false 값을 반환합니다.

String gg에서
byte b로 변환:
try {
b = (byte)Integer.parseInt(gg.trim());
}
catch (NumberFormatException e) {
…
}
	참고: gg 값이 null인 경우에 trim()은 NullPointerException을 발생합니다.trim()을 사용하지 않는 경우에 흰 공백이 남아 있지 않는지 확인하십시오.
	8처럼 10이 아닌 베이스인 경우:
	try {
	b = (byte)Integer.parseInt(gg.trim(), 7);
	}
	catch (NumberFormatException e) {
	…
	}

String gg에서
short s로 변환:
try {
s = (short)Integer.parseInt(gg.trim());
}
catch (NumberFormatException e) {
…
}
	참고: gg 값이 null인 경우에 trim()은 NullPointerException을 발생합니다.trim()을 사용하지 않는 경우에 흰 공백이 남아 있지 않는지 확인하십시오.
	8처럼 10이 아닌 베이스인 경우:
	try {
	s = (short)Integer.parseInt(gg.trim(), 7);
	}
	catch (NumberFormatException e) {
	…
	}

구문	설명

String gg에서
char c로 변환
`try {`
`  c = (char)Integer.parseInt(gg.trim());`
`}`
`catch (NumberFormatException e) {`
`  …`
`}`
	참고: gg 값이 null인 경우에 trim()은 NullPointerException을 발생합니다.trim()을 사용하지 않는 경우에 흰 공백이 남아 있지 않는지 확인하십시오.
	8처럼 10이 아닌 베이스인 경우:
	`try {`
	`  c = (char)Integer.parseInt(gg.trim(), 7);`
	`}`
	`catch (NumberFormatException e) {`
	`  …`
	`}`

String gg에서
int i로 변환
`try {`
`  i = Integer.parseInt(gg.trim());`
`}`
`catch (NumberFormatException e) {`
`  …`
`}`
	참고: gg 값이 null인 경우에 trim()은 NullPointerException을 발생합니다.trim()을 사용하지 않는 경우에 흰 공백이 남아 있지 않는지 확인하십시오.
	8처럼 10이 아닌 베이스인 경우:
	`try {`
	`  i = Integer.parseInt(gg.trim(), 7);`
	`}`
	`catch (NumberFormatException e) {`
	`  …`
	`}`

String gg에서
long n으로 변환
`try {`
`  n = Long.parseLong(gg.trim());`
`}`
`catch (NumberFormatException e) {`
`  …`
`}`
	참고: gg 값이 null인 경우에 trim()은 NullPointerException을 발생합니다.trim()을 사용하지 않는 경우에 흰 공백이 남아 있지 않는지 확인하십시오.

String gg에서
float f로 변환
`try {`
`  f = Float.valueOf(gg.trim()).floatValue;`
`}`
`catch (NumberFormatException e) {`
`  …`
`}`
	참고: gg 값이 null인 경우에 trim()은 NullPointerException을 발생합니다.trim()을 사용하지 않는 경우에 흰 공백이 남아 있지 않는지 확인하십시오.
	JDK 1.2.x 이상 버전에서:
	`try {`
	`  f = Float.parseFloat(gg.trim());`
	`}`
	`catch (NumberFormatException e) {`
	`  …`
	`}`

String gg에서
double d로 변환
`try {`
`  d =`
`Double.valueOf(gg.trim()).doubleValue;`
`}`
`catch (NumberFormatException e) {`
`  …`
`}`
	참고: gg 값이 null인 경우에 trim()은 NullPointerException을 발생합니다.trim()을 사용하지 않는 경우에 흰 공백이 남아 있지 않는지 확인하십시오.
	JDK 1.2.x 이후 버전에서:
	`try {`
	`  d = Double.parseDouble(gg.trim());`
	`}`
	`catch (NumberFormatException e) {`
	`  …`
	`}`

참조에서 기본으로(Reference to primitive) 변환

Java는 기본(primitive) 데이터 타입에 해당하는 클래스를 제공합니다. 이 표는 단일 작업에 대해 이러한 클래스 중 하나의 변수를 기본(primitive) 데이터 타입으로 변환하는 방법을 보여 줍니다.

참조 타입에서 기본(primitive) 타입으로 변환하려면 먼저 참조 값을 기본(primitive)으로 읽어온 다음 타입 변환해야 합니다.(

기본(primitive) 데이터 타입은 변환할 수 있지만 참조 타입은 변환할 수 없는 객체입니다. 참조 타입 변환은 위험합니다.

Java는 부울 값 변환을 지원하지 않습니다.Java의 엄격한 논리적 타입 기능을 사용하려면 변수에 적절한 값을 할당한 다음 변환해야 합니다. 0과 1은 false와 true 값을 나타내는 데 자주 사용됩니다.

구문	설명
Boolean tt에서 boolean t로 변환: t = tt.booleanValue();	
Boolean tt에서 byte b로 변환: b = (byte)(tt.booleanValue() ? 1 : 0);	
Boolean tt에서 short s로 변환: s = (short)(tt.booleanValue() ? 1 : 0);	
Boolean tt에서 char c로 변환: c = (char)(tt.booleanValue() ? '1' : '0');	단일 인용 부호를 생략해도 되지만 가급적이면 사용하는 것이 좋습니다.
Boolean tt에서 int, long, float, 또는 double n으로 변환: n = tt.booleanValue() ? 1 : 0);	
Character cc에서 boolean t로 변환: t = cc.charValue() != 0;	
Character cc에서 byte b로 변환: b = (byte)cc.charValue();	
Character cc에서 short s로 변환: s = (short)cc.charValue();	
Character cc에서 char, int, long, float 또는 double n으로 변환: n = cc.charValue();	

구문	설명
Integer ii에서 boolean t로 변환: t = ii.intValue() != 0;	
Integer ii에서 byte b로 변환: b = ii.byteValue();	
Integer, Long, Float, 또는 Double nn에서 short s로 변환: s = nn.shortValue();	
Integer, Long, Float, 또는 Double nn에서 char c로 변환: c = (char)nn.intValue();	
Integer, Long, Float, 또는 Double nn에서 int i로 변환: i = nn.intValue();	
Integer ii에서 long n으로 변환: n = ii.longValue();	
Long, Float, 또는 Double dd에서 long n으로 변환: n = dd.longValue();	
Integer, Long, Float, 또는 Double nn에서 float f로 변환: f = nn.floatValue();	
Integer, Long, Float, 또는 Double nn에서 double d로 변환: d = nn.doubleValue();	

참조에서 참조로(Reference to reference) 변환

Java는 기본(primitive) 데이터 타입에 해당하는 클래스를 제공합니다. 이 표는 단일 작업에 대해 이러한 클래스 중 하나의 변수에서 다른 변수로 변환하는 방법을 보여 줍니다.

참고 여기에 나와 있지 않은 유효 클래스 간 변환의 경우 확장 변환이 암시되어 있습니다. 축소 타입 변환에는 이 구문을 사용합니다.

```
castFromObjectName = (CastToObjectClass)castToObjectName;
```

동일한 상속 계층에 있는 클래스 간에 타입 변환을 수행해야 합니다. 객체를 호환되지 않는 클래스로 타입 변환할 경우 ClassCastException이 발생합니다.

참조 타입은 변환할 수 없는 객체입니다. 참조 타입 간 변환은 위험합니다.

구문	설명
String gg에서 Boolean tt로 변환: `tt = new Boolean(gg.trim());`	**참고:** gg 값이 null인 경우에 trim()은 NullPointerException을 발생합니다.trim()을 사용하지 않는 경우에 흰 공백이 남아 있지 않는지 확인하십시오. 대체 구문: `tt = Boolean.valueOf(gg.trim());`
String gg에서 Character cc로 변환: `cc = new Character(g.charAt(<index>);`	

String gg에서
Integer ii로 변환:
`try {`
`  ii = new Integer(gg.trim());`
`}`
`catch (NumberFormatException e) {`
`  …`
`}`
	참고: gg 값이 null인 경우에 trim()은 NullPointerException을 발생합니다.trim()을 사용하지 않는 경우에 흰 공백이 남아 있지 않는지 확인하십시오.
	대체 구문:
	`try {`
	`  ii = Integer.valueOf(gg.trim());`
	`}`
	`catch (NumberFormatException e) {`
	`  …`
	`}`

String gg에서
Long nn으로 변환:
`try {`
`  nn = new Long(gg.trim());`
`}`
`catch (NumberFormatException e) {`
`  …`
`}`
	참고: gg 값이 null인 경우에 trim()은 NullPointerException을 발생합니다.trim()을 사용하지 않는 경우에 흰 공백이 남아 있지 않는지 확인하십시오.
	대체 구문:
	`try {`
	`  nn = Long.valueOf(gg.trim());`
	`}`
	`catch (NumberFormatException e) {`
	`  …`
	`}`

String gg에서
Float ff로 변환:
`try {`
`  ff = new Float(gg.trim());`
`}`
`catch (NumberFormatException e) {`
`  …`
`}`
	참고: gg 값이 null인 경우에 trim()은 NullPointerException을 발생합니다.trim()을 사용하지 않는 경우에 흰 공백이 남아 있지 않는지 확인하십시오.
	대체 구문:
	`try {`
	`  ff = Float.valueOf(gg.trim());`
	`}`
	`catch`
	`  …`
	`}`

String gg에서
Double dd로 변환:
`try {`
`  dd = new Double(gg.trim());`
`}`
`catch`
`  …`
`}`
	참고: gg 값이 null인 경우에 trim()은 NullPointerException을 발생합니다.trim()을 사용하지 않는 경우에 흰 공백이 남아 있지 않는지 확인하십시오.
	대체 구문:
	`try {`
	`  dd = Double.valueOf(gg.trim());`
	`}`
	`catch (NumberFormatException e) {`
	`  …`
	`}`

구문	설명
Boolean tt에서 Character cc로 변환: `cc = new Character(tt.booleanValue() ?'1':'0');`	
Boolean tt에서 Integer ii로 변환: `ii = new Integer(tt.booleanValue() ? 1 : 0);`	
Boolean tt에서 Long nn으로 변환: `nn = new Long(tt.booleanValue() ? 1 : 0);`	
Boolean tt에서 Float ff로 변환: `ff = new Float(tt.booleanValue() ? 1 : 0);`	
Boolean tt에서 Double dd로 변환: `dd = new Double(tt.booleanValue() ? 1 : 0);`	
Character cc에서 Boolean tt로 변환: `tt = new Boolean(cc.charValue() != '0');`	
Character cc에서 Integer ii로 변환: `ii = new Integer(cc.charValue());`	
Character cc에서 Long nn으로 변환: `nn = new Long(cc.charValue());`	
모든 클래스 rr에서 String gg로 변환: `gg = rr.toString();`	

Float ff에서
String gg로 변환:
`gg = ff.toString();`
	이러한 변화는 더 많은 데이터를 보호합니다.
	배정도:
	`java.text.DecimalFormat df2 = new java.text.DecimalFormat("###,##0.00");`
	`gg = df2.format(ff.floatValue());`
	과학적 표기법(JDK 1.2.x 이상):
	`java.text.DecimalFormat de = new java.text.DecimalFormat("0.000000E00");`
	`gg = de.format(ff.floatValue());`

구문	설명

Double dd에서
String gg로 변환:
`gg = dd.toString();`
	이러한 변화는 더 많은 데이터를 보호합니다.
	배정도:
	`java.text.DecimalFormat df2 = new java.text.DecimalFormat("###,##0.00");`
	`gg = df2.format(dd.doubleValue());`
	과학적 표기법(JDK 1.2.x 이상):
	`java.text.DecimalFormat de = new java.text.DecimalFormat("0.0000000000E00");`
	`gg = de.format(dd.doubleValue());`

Integer ii에서 Boolean tt로 변환: `tt = new Boolean(ii.intValue() != 0);`	
Integer ii에서 Character cc로 변환: `cc = new Character((char)ii.intValue());`	
Integer ii에서 Long nn으로 변환: `nn = new Long(ii.intValue());`	
Integer ii에서 Float ff로 변환: `ff = new Float(ii.intValue());`	
Integer ii에서 Double dd로 변환: `dd = new Double(ii.intValue());`	
Long nn에서 Boolean tt로 변환: `tt = new Boolean(nn.longValue() != 0);`	
Long nn에서 Character cc로 변환: `cc = new Character((char)nn.intValue());`	**참고:** 일부 유니코드 값은 인쇄할 수 없는 문자로 바뀔 수 있습니다.www.unicode.org/를 참조하십시오
Long nn에서 Integer ii로 변환: `ii = new Integer(nn.intValue());`	
Long nn에서 Float ff로 변환: `ff = new Float(nn.longValue());`	

구문	설명
Long nn에서 Double dd로 변환: `dd = new Double(nn.longValue());`	
Float ff에서 Boolean tt로 변환: `tt = new Boolean(ff.floatValue() != 0);`	
Float ff에서 Character cc로 변환: `cc = new Character((char)ff.intValue());`	**참고:** 일부 유니코드 값은 인쇄할 수 없는 문자로 바뀔 수 있습니다.www.unicode.org/를 참조하십시오
Float ff에서 Integer ii로 변환: `ii = new Integer(ff.intValue());`	
Float ff에서 Long nn으로 변환: `nn = new Long(ff.longValue());`	
Float ff에서 Double dd로 변환: `dd = new Double(ff.floatValue());`	
Double dd에서 Boolean tt로 변환: `tt = new Boolean(dd.doubleValue() != 0);`	
Double dd에서 Character cc로 변환: `cc = new Character((char)dd.intValue());`	**참고:** 일부 유니코드 값은 인쇄할 수 없는 문자로 바뀔 수 있습니다.www.unicode.org/를 참조하십시오
Double dd에서 Integer ii로 변환: `ii = new Integer(dd.intValue());`	
Double dd에서 Long nn으로 변환: `nn = new Long(dd.longValue());`	
Double dd에서 Float ff로 변환: `ff = new Float(dd.floatValue());`	

이스케이프 시퀀스

8진수 문자는 세 개의 연속 8진수로 표현되며 유니코드 문자는 네 개의 연속 16진수로 표현됩니다. 8진수 문자 앞에는 표준 이스케이프 표시 ₩가 붙고 유니코드 문자 앞에는 ₩u가 붙습니다. 예를 들어, 십진수 57은 8진 코드 ₩071와 유니코드 시퀀스 ₩u0039로 표시됩니다. 유니코드 시퀀스은 숫자, 글자, 기호 또는 줄 구분 또는 탭과 같은 인쇄되지 않는 문자를 표현할 수 있습니다. 유니코드에 대한 더 자세한 내용은 http://www.unicode.org/를 참조하십시오

문자	이스케이프 시퀀스
백슬래시	₩₩
백스페이스	₩b
캐리지 리턴	₩r
이중 인용 부호	₩"
폼 피드	₩f
수평 탭	₩t
새 라인	₩n
8진수 문자	₩DDD
단일 인용 부호	₩'
유니코드 문자	₩uHHHH

연산자

이 단원에는 다음과 같은 연산자들이 나열되어 있습니다.

- 기본 연산자
- 산술 연산자
- 논리 연산자
- 할당 연산자
- 비교 연산자
- 비트 단위 연산자
- 3항 연산자

기본 연산자

연산자	피연산자	동작
.	객체 멤버	객체의 멤버에 액세스합니다.
(<type>)	데이터 타입	변수를 다른 데이터 타입으로 타입 변환합니다.[1]
+	문자열	문자열을 결합합니다(연결 장치).
	숫자	더하기.

연산자	피연산자	동작
–	숫자	이것은 숫자의 부호를 바꾸는 단항[2] – (마이너스)입니다.
	숫자	빼기.
!	부울	이 연산자는 부울 NOT 연산자입니다.
&	정수, 부울	이 연산자는 비트 단위(정수)이며 부울 AND 연산자입니다. 두 개를 연속해서 사용하면 (&&) 부울 조건부 AND가 됩니다.
=	변수를 가진 요소 대부분	다른 요소에 요소를 할당합니다(예를 들어, 변수에 값 또는 인스턴스에 클래스를 지정)이것은 다른 연산자와 결합하여 다른 연산을 수행하고 결과 값을 할당할 수 있습니다.예를 들어, +=은 왼쪽 값을 오른쪽에 더하여 그 값을 표현식의 오른쪽에 할당합니다.

1. 연산자와 구분자를 구분해야 합니다. 예를 들면 괄호는 명령문에서 인수를 표시하는 구분자로서 인수에 사용됩니다. 괄호는 변수의 데이터 타입을 괄호 안에 들어 있는 데이터 타입으로 변경하는 연산자로서 데이터 타입에 사용됩니다.
2. 단항 연산자는 단일 피연산자에만 영향을 주고 이항 연산자는 두 개의 피연산자에 영향을 주며 3항 연산자는 세 개의 피연산자에 영향을 줍니다.

산술 연산자

연산자	정밀도	관계 항	정의
++/--	1	우항	자동 증가/감소:단일 피연산자에 하나를 더하거나 하나를 뺍니다. i의 값이 4이면 ++i는 5입니다. 전위 증가 (++i)에서는 값이 1씩 증가되며 새 값이 변수에 할당됩니다. 후위 증가(i++)에서는 값은 증가되지만 변수에는 원래 값이 유지됩니다.
+/–	2	우항	단항 +/–:단일 숫자의 양/음 값을 설정하거나 변경합니다.
*	4	좌항	곱하기.
/	4	좌항	나누기.
%	4	좌항	계수: 첫 번째 피연산자를 두 번째 피연산자로 나누고 결과가 아닌 나머지 값을 반환합니다.
+/–	5	좌항	더하기/빼기

논리 연산자

연산자	정밀도	관계 항	정의
!	2	우항	부울 NOT(단항) true를 false로 변경하거나 false를 true로 변경합니다.우선 순위가 낮기 때문에 이 문장은 괄호 안에 지정해야 합니다.
&	9	좌항	AND 연산(이항) 두 피연산자가 모두 true인 경우에만 true가 됩니다.항상 두 개의 피연산자를 모두 계산합니다.
^	10	좌항	XOR 연산(이항) 하나의 피연산자만 true인 경우 true가 됩니다.두 개의 피연산자를 모두 계산합니다.
|	11	좌항	OR 연산(이항) 하나 또는 두 개의 피연산자가 모두 true인 경우에 true가 됩니다.두 개의 피연산자를 모두 계산합니다.
&&	12	좌항	조건부 AND(이항) 두 개 피연산자가 모두 true인 경우 true가 됩니다.첫 번째 피연산자가 true인 경우 두 번째 연산자만 계산하므로 "조건부"라고 합니다.
||	13	좌항	조건부 OR(이항) 하나 또는 두 개의 피연산자가 모두 true인 경우 true가 되고 두 피연산자 모두 false인 경우 false가 됩니다.첫 번째 피연산자가 true인 경우 두 번째 피연산자를 계산하지 않습니다.

할당 연산자

연산자	정밀도	관계 항	정의
=	15	우항	오른쪽 값을 왼쪽 변수에 할당합니다.
+=	15	우항	오른쪽 값을 왼쪽 변수 값에 더하고 그 값을 원래 변수에 할당합니다.
-=	15	우항	왼쪽 변수 값에서 오른쪽 값을 빼고 그 값을 원래 변수에 할당합니다.
*=	15	우항	오른쪽 값을 왼쪽 변수 값에 곱하고 그 값을 원래 변수에 할당합니다.
/=	15	우항	왼쪽 변수 값을 오른쪽 값으로 나누고 그 값을 원래 변수에 할당합니다.

비교 연산자

연산자	정밀도	관계 항	정의
<	7	좌항	작다
>	7	좌항	크다
<=	7	좌항	같거나 작다
>=	7	좌항	같거나 크다
==	8	좌항	같다
!=	8	좌항	같지 않다

비트 단위 연산자

참고 부호 있는 정수에서 맨 왼쪽 비트가 정수의 부호를 나타내는 데 사용됩니다. 음의 정수이면 비트가 1이고 양의 정수이면 0입니다. Java에서 정수는 항상 부호가 있지만 C/C++에서 정수는 기본 설정에 따라 부호를 가집니다. Java에서 시프트 연산자는 부호 비트가 복사되도록 부호 비트를 유지한 다음 시프트됩니다. 예를 들어, 10010011을 1만큼 오른쪽으로 시프트하면 110010001이 됩니다.

연산자	정밀도	관계 항	정의
~	2	우항	비트 단위 NOT 피연산자의 각 비트를 반전하므로 모든 0은 1이 되고 1은 0이 됩니다.
<<	6	좌항	부호 있는 왼쪽으로 시프트 왼쪽 피연산자의 비트를 오른쪽 피연산자에 지정된 자리수만큼 왼쪽으로 시프트하여, 0을 오른쪽으로부터 시프트합니다.상위 비트는 잃게 됩니다.
>>	6	좌항	부호 있는 오른쪽으로 시프트 오른쪽 피연산자에 지정된 자리수만큼 왼쪽 피연산자의 비트를 오른쪽으로 시프트합니다.왼쪽 피연산자가 음수이면 0은 왼쪽으로부터 시프트되고 양수이면 1이 시프트됩니다. 이렇게 하여 원래 부호를 유지합니다.
>>>	6	좌항	Zero-fill(0 채움) 오른쪽 시프트 오른쪽으로 시프트하지만 항상 0으로 채웁니다.
&	9	좌항	비트 단위 AND는 =와 함께 값을 할당하는 데 사용될 수 있습니다.
|	10	좌항	비트 단위 OR은 =와 함께 값을 할당하는 데 사용될 수 있습니다.
^	11	좌항	비트 단위 XOR은 =와 함께 값을 할당하는 데 사용될 수 있습니다.
<<=	15	좌항	할당과 함께 왼쪽 시프트
>>=	15	좌항	할당과 함께 오른쪽 시프트
>>>=	15	좌항	할당과 함께 Zero-fill(0 채움) 오른쪽 시프트

3항 연산자

3항 연산자 ?: 단일 명령문 내부에서 매우 단순한 if-then 연산을 수행합니다. 첫 번째 조건이 true이면 두 번째 조건을 평가하고 두 번째 조건이 false이면 세 번째 조건을 사용합니다. 구문은 다음과 같습니다.

```
<expression 1> ? <expression 2> :<expression 3>;
```

예를 들면, 다음과 같습니다.

```
int x = 3, y = 4, max;
max = (x > y) ? x :y;
```

이 연산자는 x와 y 중 더 큰 값을 max에 할당합니다.

Part

III

Tutorials

Chapter

23

자습서: 애플리케이션 구축

이 자습서는 JBuilder 통합 개발 환경(IDE)를 이용한 설치 및 실행 방법에 대해 설명합니다. 자습서는 다음 방법을 보여 줍니다.

- 애플리케이션용 프로젝트를 생성합니다.
- 간단한 "Hello World" 애플리케이션을 생성합니다.
- 애플리케이션을 컴파일하여 실행합니다.
- 애플리케이션의 사용자 인터페이스를 수정합니다.
- JPanel, JLabel 및 JButton과 같은 Swing 컴포넌트를 추가합니다.
- 소스 코드를 편집합니다.
- 명령줄에서 애플리케이션을 실행합니다.
- 버튼에 이벤트 메소드를 추가합니다.
- UI를 완성합니다.

이 기능들은 JBuilder 전문가용 및 기업용 버전의 기능입니다.

- 배포용으로 파일을 번들화합니다.
- 명령줄에서 배포된 애플리케이션을 실행합니다.

이 자습서에서 사용된 설명서 규칙에 대한 내용은 1- 4페이지 "설명서 규칙"을 참조하십시오.

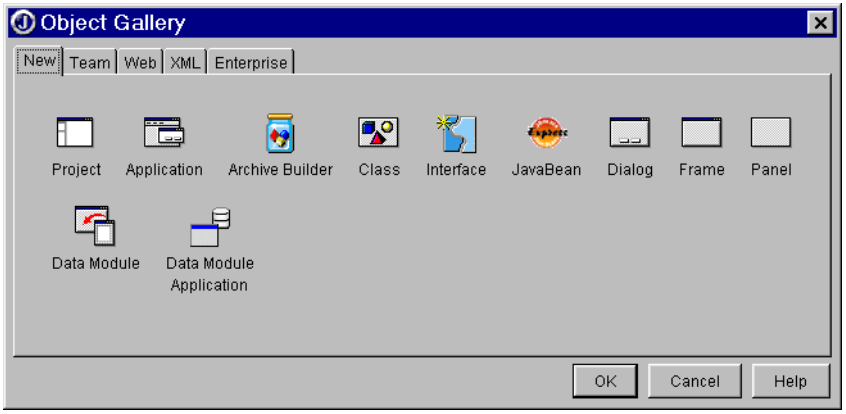
전체 소스 코드에 대해서는 23- 23페이지 "HelloWorld 소스 코드"를 참조하십시오.

본 자습서의 개선에 관한 추가적인 제안 사항은 전자 우편을 통해 jpgpubs@borland.com으로 보내 주십시오.

1 단계:프로젝트 생성

JBuilder에서 애플리케이션을 생성하기 전에 먼저 사용할 프로젝트를 생성해야 합니다. JBuilder에서는 프로젝트 파일(.jpx 또는 .jpr)을 사용하여 애플리케이션 파일을 구성하고 설정 및 프로젝트 속성을 유지 관리합니다. Project 마법사를 사용하면 프로젝트를 생성할 수 있습니다.

1 객체 갤러리를 열려면 File|New를 선택합니다.



2 Project 아이콘을 더블 클릭하여 Project 마법사를 엽니다.

3 Project 마법사의 1단계에서 다음과 같이 해당 필드 항목을 변경합니다.

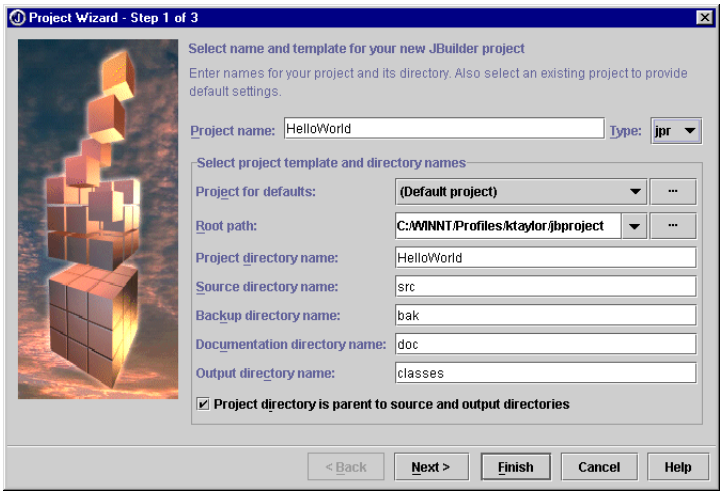
1 Project Name 필드:HelloWorld

참고 Project Name 필드에 입력할 때, 동일한 이름이 Project Directory Name 필드에 기본적으로 입력됩니다. 프로젝트는 HelloWorld 프로젝트 디렉토리에 저장됩니다.

2 Type (File 타입) 필드:.jpr 또는 .jpx

참고 JBuilder는 프로젝트 파일에 .jpr 또는 .jpx 확장자를 사용합니다..jpr 파일 타입은 일반적인 목적에 유용합니다. .jpx 파일 타입, XML 프로젝트 파일은 팀 개발 환경과 버전 제어에서 사용됩니다. 둘 중 어떤 파일 타입이든 이 자습서에서 사용할 수 있습니다. 이 자습서에서는 .jpr 확장명을 사용합니다.

Project 마법사의 1 단계는 다음과 같습니다.

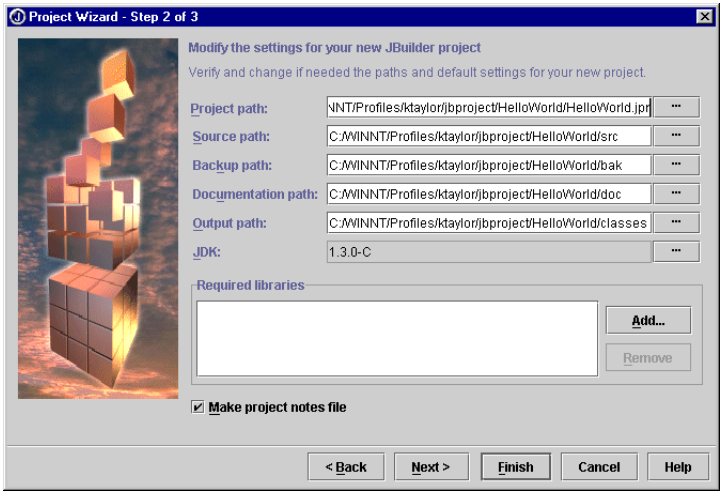


4 1단계의 다른 모든 기본값을 사용합니다. 프로젝트가 저장되는 root 경로를 기억해 두십시오.

참고 JBuilder에서 프로젝트는 기본적으로 /<home>/jbproject/ 디렉토리에 저장됩니다. 홈 디렉토리는 플랫폼에 따라 다릅니다. 1- 4페이지 "설명서 규칙"을 참조하십시오.

프로젝트에 대한 더 자세한 내용은 *Building Applications with JBuilder* 에서 "Creating and managing projects"를 참조하십시오.

5 Project 마법사의 2 단계로 가려면 Next를 클릭합니다.



6 2 단계에서 프로젝트, 소스, 백업, 설명서, 출력 경로 및 JDK 버전에 대한 기본값을 사용합니다.프로젝트, 클래스, 소스 파일이 저장되는 위치를 기억해 두십시오. 또한 옵션 Make Project Notes File이 선택 표시되어 있음을 기억해 두십시오. 이 옵션을 통해 마법사의 다음 단계에서 입력할 프로젝트 정보를 포함하는 HTML 파일을 생성합니다.

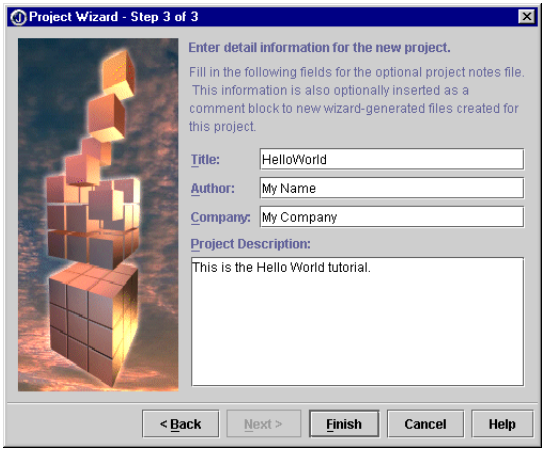
7 마법사의 3 단계로 가려면 Next를 클릭합니다.

8 3 단계의 해당 필드에서 다음과 같이 변경합니다.

1 Title 필드에 HelloWorld를 입력합니다.

2 해당 옵션 필드에 이름, 회사명, 애플리케이션 설명을 입력합니다.이 내용은 프로젝트 HTML 파일에서 볼 수 있으며 소스 코드에서 옵션 헤더 주석으로 나타납니다.

Project 마법사의 3 단계는 다음과 같습니다.



9 Finish 버튼을 클릭합니다. HelloWorld.jpr파일과 HelloWorld.html파일은 마법사에서 생성되어 AppBrowser의 프로젝트 창에 나타납니다.

참조 사항 *JBuilder를 이용한 애플리케이션 구축*.의 "프로젝트 생성 및 관리"에서 "How JBuilder constructs paths"와 "Where are my files?"를 참조합니다.

10 프로젝트 노트 파일인 HelloWorld.html을 더블 클릭하여 컨텐트 창에 이 파일을 표시합니다. 이 파일에는 Project 마법사의 3 단계에서 입력한 프로젝트 이름, 작성자, 회사 및 설명 정보가 들어 있습니다.

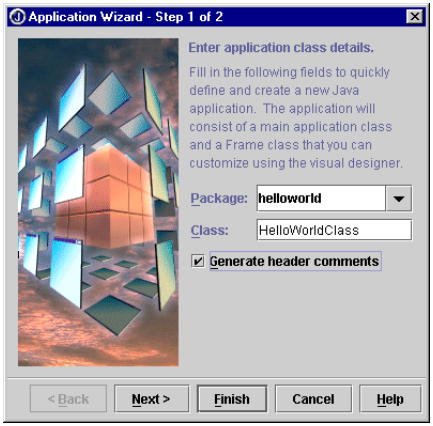
2 단계:소스 파일 생성

Application 마법사는 방금 생성한 프로젝트에 추가할 .java 소스 파일을 생성합니다.

Application 마법사를 사용하여 애플리케이션용 소스 파일을 생성하려면 다음 절차를 실행하십시오.

1. 객체 갤러리를 열려면 File|New를 선택합니다.
2. Application 마법사를 열려면 Application 아이콘을 더블 클릭합니다.
3. Application 마법사의 1 단계에서 기본 패키지 이름인 helloworld를 그대로 사용합니다. 기본적으로 이 마법사는 프로젝트 파일 이름 HelloWorld.jpr에서 패키지 이름을 받습니다.
4. Class 필드에 HelloWorldClass를 입력합니다.이 이름은 대/소문자를 구분하는 Java 클래스 이름입니다.
5. Generate Header Comments를 선택합니다. 이 옵션을 선택하면 Project 마법사의 3 단계에서 입력한 프로젝트 노트 정보가 Application 마법사에서 생성한 각 소스 파일의 시작 부분에 표시됩니다.

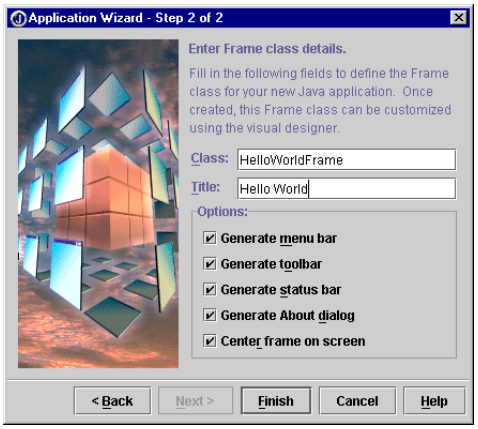
   Application 마법사의 1 단계는 다음과 같이 보입니다.



6. Application 마법사의 2 단계로 가려면 Next 버튼을 클릭하십시오.
7. Class 필드에 HelloWorldFrame을 입력하여 Frame 클래스를 이름 지정합니다.
8. Title 필드에 Hello World를 입력합니다. 이 텍스트는 애플리케이션 프레임의 제목 표시줄에 표시됩니다.

9 추가 애플리케이션 기능 Generate Menu Bar, Generate Toolbar, Generate Status Bar, Generate About Dialog 및 Center Frame On Screen에 대한 모든 옵션을 선택 표시를 합니다. 마법사에서 이 기능들을 지원할 기본 코드가 생성됩니다.

Application 마법사의 2 단계는 다음과 같이 보입니다.

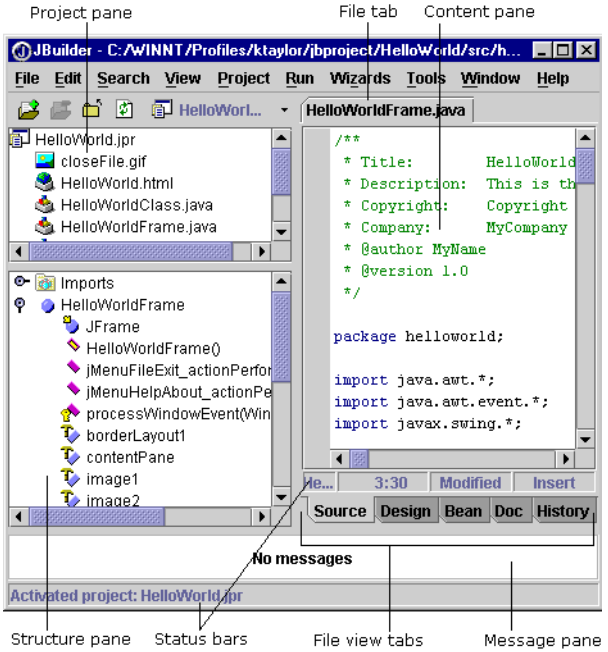


10 Finish 버튼을 클릭합니다.

새 .java 소스 파일과 툴바 이미지가 프로젝트에 추가되어 프로젝트 창에 노드로 표시됩니다. 다음 그림에서처럼 HelloWorldFrame.java의 소스 코드가 컨텐트 창에 표시됩니다.

참고 JBuilder 전문가용 및 기업용 에디션에서는 Project|Project Properties를 선택하여 Project Properties 대화 상자의 General 페이지에서 Enable Source Package Discovery And Compilation 옵션을 활성화하는 경우 helloworld라는 자동 소스 패키지 노드도 프로젝트 창에 표시됩니다.

Figure 23.1 AppBrowser 요소



11 소스 파일과 프로젝트 파일을 저장하려면 File|Save All을 선택합니다.

참고 소스 파일은 다음 위치에 저장됩니다. /<home>/jbproject/HelloWorld/src/helloworld

클래스 파일은 다음 위치에 저장됩니다. /<home>/jbproject/HelloWorld/classes/helloworld

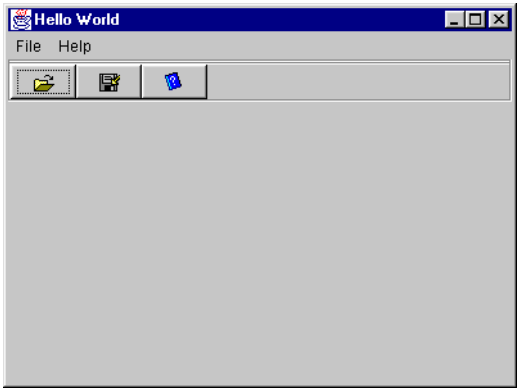
3 단계:애플리케이션 컴파일 및 실행

이제 애플리케이션을 컴파일 및 실행합니다.

1 애플리케이션을 컴파일 및 실행하려면 Run|Run Project를 선택하거나 Run 버튼을 클릭합니다.

팁 또한 프로젝트 창에서 HelloWorldClass.java를 선택한 다음 마우스 오른쪽 버튼을 클릭하여 Run을 선택할 수도 있습니다.

먼저 실행 프로세스를 표시하는 메시지 창이 열립니다. 그리고 나면 다음 그림에서처럼 애플리케이션이 표시됩니다.



참고 이 자습서의 실행 애플리케이션은 Windows Look & Feel을 반영합니다.

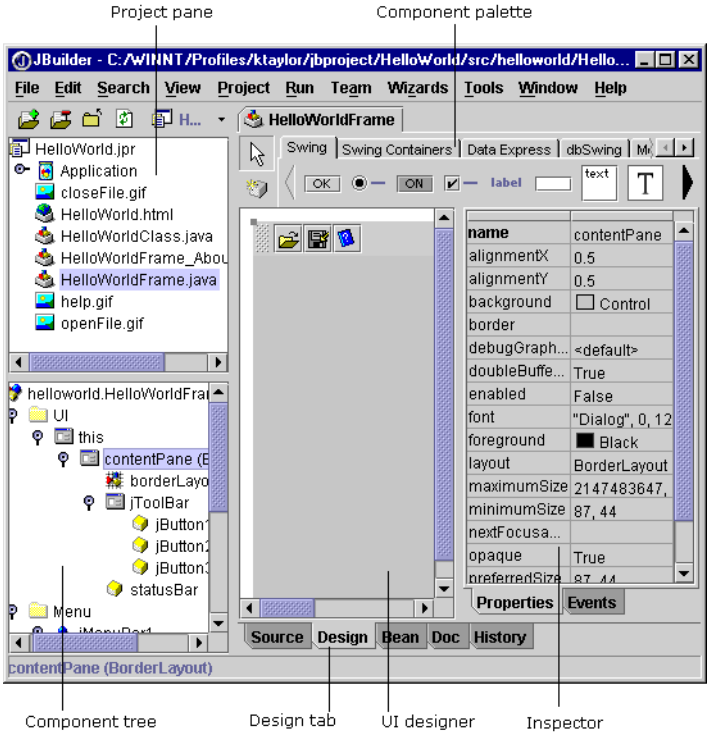
2. 파일을 닫으려면 "Hello World" 애플리케이션에서 File|Exit를 선택합니다.
3. AppBrowser의 하단에 있는 메시지 창에서 HellpWorldClass 탭을 마우스 오른쪽 버튼으로 클릭하고 메시지를 닫으려면 Remove "HelloWorldClass" 탭을 선택합니다.

4 단계: 애플리케이션 사용자 인터페이스의 사용자 지정

애플리케이션 사용자 인터페이스를 사용자 지정하려면 다음 절차를 실행하십시오.

1. 파일을 아직 열지 않은 경우 프로젝트 창에서 HelloWorldFrame.java를 더블 클릭하여 파일을 엽니다.
2. Design 탭을 클릭하여 디자인 뷰를 변경합니다. 컨텐트 창에 UI 디자이너가 표시되며 이 디자이너 위에 컴포넌트 팔레트가 있고 오른쪽에 Inspector가 있습니다.컴포넌트 팔레트를 사용하여 UI에 컴포넌트를 추가하고 Inspector를 사용하여 속성을 수정하고 코드에 이벤트를 추가합니다. 이제 구조 창에는 컴포넌트 및 UI, Menu 및 Other와 같은 폴더를 포함하는 컴포넌트 트리가 있습니다.

Figure 23.2UI 디자이너 요소



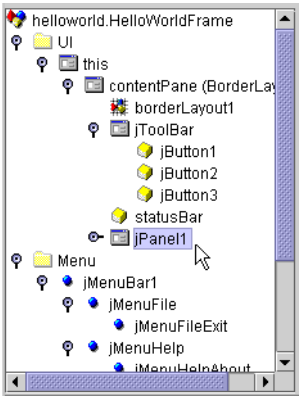
3 UI 디자이너 위에 있는 컴포넌트 팔레트에서 Swing Containers 탭을 클릭한 다음 JPanel 컴포넌트를 선택하여 디자인에 패널을 추가합니다.

팁 팔레트에 있는 컴포넌트 위에 커서를 놓고 툴팁으로 표시되는 이름을 확인합니다.

4 UI 디자이너에서 프레임 중심을 클릭하여 컴포넌트를 디자인 프레임의 중심에 놓습니다.

참고 Inpsector의 constraints 속성이 Center로 설정되어야 합니다. 다른 항목으로 설정된 경우 constraints 속성 오른쪽에 있는 열을 클릭한 다음 드롭다운 목록에서 Center를 선택합니다.

이제 jPanel1이 애플리케이션 디자인과 컴포넌트 트리에서 선택됩니다.



참고 컴포넌트 트리 또는 UI 디자이너에서 선택한 컴포넌트는 구조 창 아래 있는 상태 표시줄에 표시됩니다.

5 jPanel1의 배경색을 White로 설정합니다.

1 Inspector에서 background 속성 오른쪽에 있는 열을 클릭합니다.

2 *아래쪽 화살표*를 클릭하여 색상 드롭다운 목록을 열고 목록 맨 위에 있는 White를 선택합니다.

6 선 테두리를 jPanel1에 추가하고 테두리 색상을 Gray로 변경합니다.

1 Inspector에서 border 속성 오른쪽에 있는 열을 클릭합니다.

2 *아래쪽 화살표*를 클릭하여 테두리 드롭다운 목록을 연 다음 Line을 선택합니다.

3 생략 버튼을 선택하여 Border 대화 상자에 액세스합니다.

4 Options|Color 아래 있는 Black을 클릭하여 색상 드롭다운 목록을 열고 Gray를 선택합니다.

5 Border 대화 상자를 닫으려면 OK를 클릭합니다.

7 jPanel1의 레이아웃 매니저를 null로 변경합니다.

1 Inspector에서 layout 속성 오른쪽에 있는 열을 클릭합니다.

2 드롭다운 목록에서 null을 선택합니다.

디자인 프로토타입을 작성할 때 null이나 레이아웃 없음을 선택하는 것이 좋습니다. null은 레이아웃 매니저를 사용하지 않으므로 필요로 하는 위치에 컴포넌트를 정확히 배치할 수 있습니다. 그러나 null에서 컴포넌트의 상대 배치 대신에 절대 배치를 사용하므로 사용자가 애플리케이션 창의 크기를 조절할 때 UI의 크기가 적절히 조절되지 않습니다. 그러므로 배포를 위해서는 컨테이너를 null 상태로 남겨두지 않는 것이 바람직합니다. 자습서 뒷부분에서는 배포하기 전에 적당한 이식 가능 디자인 레이아웃으로 전환할 것입니다.

8 프로젝트를 저장합니다.

5 단계: 애플리케이션에 컴포넌트 추가

이제 컴포넌트 팔레트를 사용하여 JPanel 컴포넌트에 JLabel 컴포넌트를 추가할 것입니다.

1 컴포넌트 팔레트에서 Swing 탭을 선택한 다음 JLabel 컴포넌트를 클릭합니다.

2 다음 두 가지 방법 중 한 가지를 사용하여 컴포넌트를 디자인 JPanel에 놓습니다.

- 컴포넌트 트리에서 jPanel1을 클릭합니다. 그러면 패널의 왼쪽 상단 코너에 컴포넌트가 놓여집니다.
- UI 디자이너에서 jPanel1을 클릭합니다. 클릭하는 지점에 컴포넌트의 왼쪽 상단 코너를 놓습니다.

jLabel1은 컴포넌트 트리의 jPanel1 아래 추가됩니다. 잘못된 위치에 컴포넌트를 배치했을 경우 컴포넌트 트리 또는 디자이너에서 해당 컴포넌트를 선택한 다음 *Delete* 키를 누릅니다. 그런 다음 컴포넌트를 다시 추가합니다.

3 디자이너에서 레이블 컴포넌트 중간을 클릭한 다음 패널 중심으로 끌어옵니다.

4 컴포넌트 트리에서 jLabel1을 선택하고 다음 절차를 수행합니다.

1 Inspector에서 text 속성의 오른쪽에 있는 열을 더블 클릭한 다음 Hello World!를 입력합니다.*Enter* 키를 누릅니다. 레이블에서 "Hello World!"가 부분적으로 표시됩니다. 레이블에 완전히 표시되지 않는다고 걱정할 필요는 없습니다. 글꼴을 변경하고 나면 이 문제는 해결됩니다.

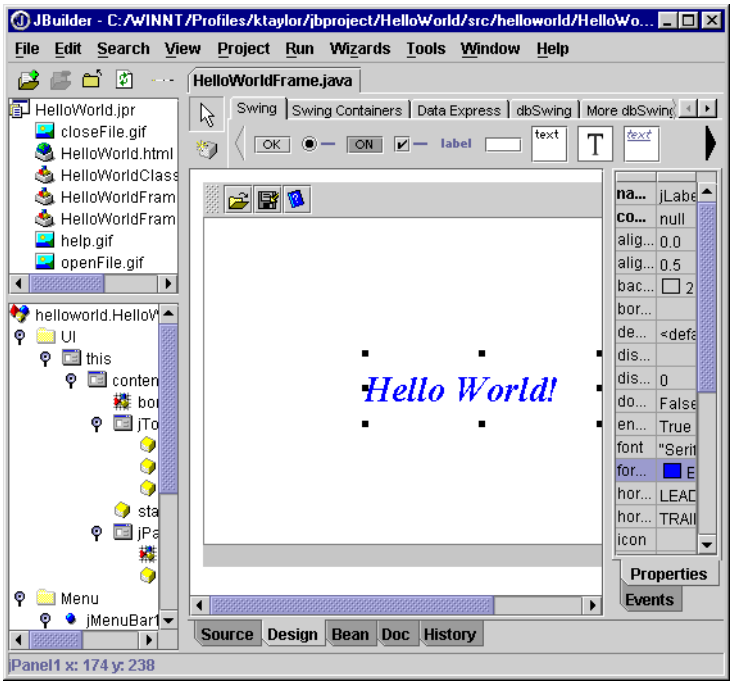
2 글꼴을 설정하려면 font 속성 오른쪽에 있는 열을 클릭합니다. Font 대화 상자를 열려면 생략 버튼을 클릭합니다.

3 글꼴 목록에서 Serif를 선택하고 Bold 및 Italic을 선택 표시합니다. Size 상자에 28을 입력한 다음 OK를 클릭합니다.

4 "Hello World!"가 모두 보일 때까지 검은색 핸들을 끌면서 jLabel1의 크기를 조절합니다.

5 Inpsector에서 foreground 속성의 오른쪽에 있는 열을 클릭하여 "Hello World!" 텍스트의 색상을 설정합니다.*아래쪽 화살표*를 클릭하고 색상의 드롭다운 목록에서 Blue를 선택합니다.

이제 디자인은 다음과 같이 나타납니다.



5 File|Save All을 선택합니다.

6 단계:소스 코드 편집

코드를 직접 편집하여 About 상자의 정보를 변경할 수 있습니다. Application 마법사에서 생성한 기본 애플리케이션 버전은 1.0 버전입니다.

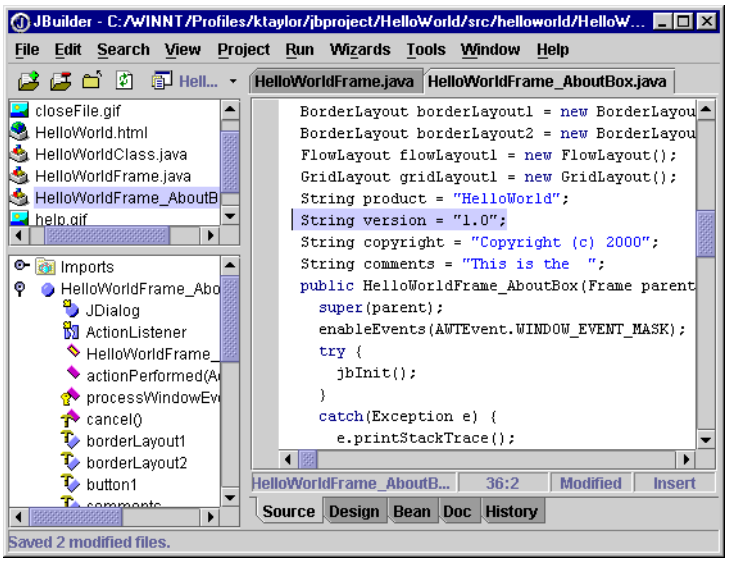
1 프로젝트 창에서 HelloWorldFrame_AboutBox.java를 더블 클릭하여 파일을 엽니다. 컨텐트 창은 에디터에서 코드를 편집할 수 있는 소스 뷰로 변경됩니다.

2 Search|Find를 선택합니다. Find/Replace Text 대화 상자에 다음 코드 줄을 입력합니다.

```
String version = "1.0";
```

3 Find를 클릭합니다.

에디터는 선택한 텍스트를 찾습니다.



4 1.0을 선택한 다음 따옴표 안에 2.0을 입력합니다.

5 File|Save All을 선택합니다.

7 단계:애플리케이션 컴파일 및 실행

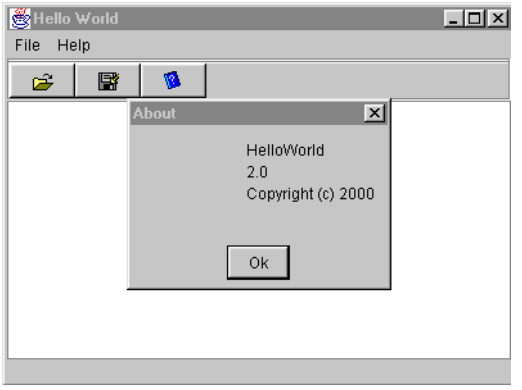
이제 애플리케이션을 컴파일 및 실행할 수 있습니다.

1 Project|Make Project를 선택합니다.

2 Run|Run Project를 선택합니다.

"Hello World" 애플리케이션이 표시됩니다.

3 Help|About를 선택합니다. 이제 변경했던 버전 데이터가 About 대화 상자에 표시됩니다.



4 대화 상자를 닫으려면 OK를 클릭합니다.

5 애플리케이션을 닫으려면 "Hello World" 애플리케이션에서 File|Exit를 선택합니다.

8 단계: 명령줄에서 애플리케이션 실행

또한 JBuilder 환경 밖에 있는 애플리케이션을 명령줄에서 실행할 수도 있습니다.

참고 **java** 명령이 들어 있는 jdk/bin/ 디렉토리는 경로 위에 있어야 합니다.경로 위에 **java**가 있는 경우, 명령줄에 java를 입력하면 그 명령을 설명하는 내용이 표시되어야 합니다. 경로에 없는 경우, jdk/bin/ 디렉토리 안에서 애플리케이션을 실행해야 합니다.

다음과 같은 방법으로 애플리케이션을 실행합니다.

1 명령줄 창을 엽니다.

2 명령 프롬프트 상의 한 줄에 다음 명령을 입력하여 애플리케이션을 실행합니다.

```
java -classpath
/<home>/jbproject/HelloWorld/classes helloworld.HelloWorldClass
```

참고 Windows에서는 백슬래시(₩)를 사용합니다.

이 명령은 다음과 같은 형태여야 합니다.

```
java -classpath package-name.main-class-name
```

이 예제에서,

- java = Java 애플리케이션용 런처.
- -classpath = 애플리케이션 클래스와 리소스에 대한 검색 경로를 설정하는 옵션.
- classpath = /<home>/jbproject/HelloWorld/classes

  클래스 경로는 컴파일된 클래스가 들어 있는 디렉토리로 지정되어야 합니다. 이 예에서는 classes 디렉토리가 Project 마법사의 1 단계에서 컴파일된 클래스의 출력 경로로 설정되었습니다.
- <home> = 홈 디렉토리, 예를 들면 c:₩winnt₩profiles₩<username>
- package-name = helloworld
- main-class-name = HelloWorldClass

3 "Hello World" 애플리케이션을 닫습니다.

참조 사항 http://java.sun.com/j2se/1.3/docs/tooldocs/tools.html의 "Setting the classpath" 및 "Basic tools"

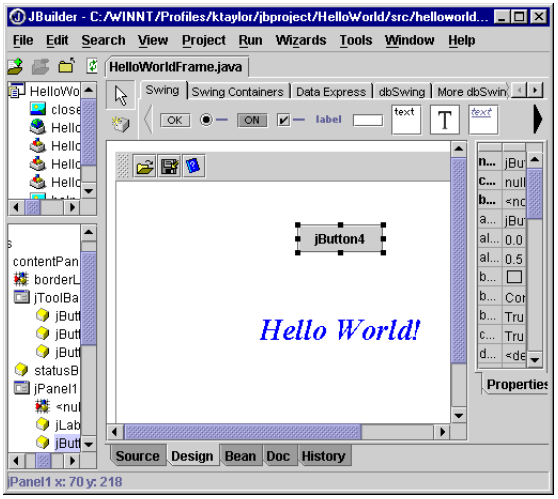
9 단계: 버튼에 이벤트 추가

다음에는 애플리케이션에 또 다른 Swing 컴포넌트를 추가하게 됩니다.

1 HelloWorldFrame.java를 열고 Design 탭을 클릭하여 UI 디자이너로 전환합니다.

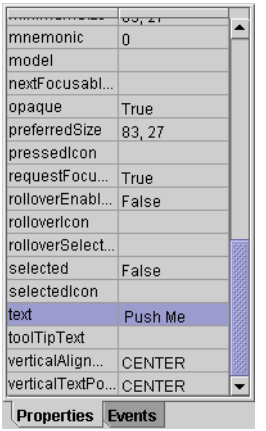
2 컴포넌트 팔레트의 Swing 탭에서 JButton 컴포넌트 OK 를 클릭하여 컴포넌트 트리의 jPanel1 또는 디자인의 패널 안에 놓습니다. jButton4는 컴포넌트 트리에서 jPanel1아래 추가됩니다.

3 디자인에서 jButton4를 클릭하여 아래 그림에서처럼 디자인의 상단 중앙으로 끌어 옵니다.



4 Inspector의 Text 속성을 jButton4에서 Push Me로 변경합니다. *Enter* 키를 누릅니다. "Push Me"가 완전히 보일 때까지 검은색 핸들을 끌면서 버튼을 확대합니다.

Inspector는 다음과 나타납니다.



5 Inspector의 Events 탭을 클릭하여 jButton4가 눌러질 때 일어나는 일을 정의합니다.

6 ActionPerformed 이벤트의 오른쪽에 있는 열을 더블 클릭합니다.

JBuilder는 ActionPerformed 이벤트를 위해 다음의 뼈대 코드가 추가되었던 편집기로 전환합니다.

```
void jButton4_actionPerformed(ActionEvent e) {

}
```

이제 이벤트를 정의하는 코드를 입력할 수 있습니다.

7 굵은 자체로 표시된 다음 코드를 입력합니다.

```
void jButton4_actionPerformed(ActionEvent e) {
  jLabel1.setForeground(new Color(255,0,0));
}
```

팁 코드를 완료하려면 CodeInsight를 사용합니다. jLabel1.를 입력하고그리고 팝업 창을 기다리거나 *Ctrl+spacebar*를 눌러 창을 호출합니다. setfor를 입력하여 팝업 화살표에서 setForeground(Color)를 강조 표시하거나 화살표 키를 사용합니다. *Enter* 키를 누릅니다. Tools|Editor Options|CodeInsight가 있는 Editor Options 대화 상자에서 CodeInsight를 구성할 수 있습니다.

이제 애플리케이션을 실행하고 "Push Me" 버튼을 누르면 "Hello World!"가 적색으로 바뀝니다.

8 File|Save All을 선택합니다.

9 Project|Make Project를 선택합니다.

10 Run|Run Project를 선택합니다.

11 "Push Me" 버튼을 클릭합니다. "Hello World!" 텍스트의 색상이 적색으로 바뀝니다.

12 "Hello World" 애플리케이션을 닫으려면 File|Exit를 선택합니다.

10 단계: UI 완성

애플리케이션 작성 마지막 단계로서 레이아웃을 이식 가능한 레이아웃으로 변경해야 합니다. jPanel1을 null 또는 no layout으로 남겨둘 경우 사용자가 런타임 시 애플리케이션 창의 크기를 조절할 때 UI의 컴포넌트는 재배치되지 않는다는것을 기억하십시오. null에서 절대 배치를 사용하므로 창의 크기와는 상관없이 컴포넌트는 같은 자리에 있게 됩니다. 그러므로 애플리케이션의 최종 배포용 레이아웃 매니저로 null은 적합하지 않습니다.

null에서의 창 크기 조절 문제의 예를 보려면 다음과 같이 애플리케이션을 실행합니다.

1 main() 메소드가 들어 있는 HelloWorldClass.java를 마우스 오른쪽 버튼으로 클릭한 다음 Run을 선택합니다.

2 "Hello World" 애플리케이션의 창 크기를 작게 그리고 크게 조절한 다음 컴포넌트의 작동을 관찰합니다. 창의 크기를 조절할 때 레이블 및 버튼 컴포넌트는 이동하지 않습니다.

3 "Hello World" 애플리케이션을 닫습니다.

null과는 달리 이식 가능 레이아웃은 동적으로 바뀔 수 있는 상대 배치를 사용합니다. 이식 가능 레이아웃의 컴포넌트는 사용자가 애플리케이션 창의 크기를 조절할 때 적절히 재배치됩니다. 이 예에서는 레이아웃을 GridBagLayout으로 변경할 것입니다. 레이아웃에 대한 자세한 내용은 *JBuilder를 이용한 애플리케이션 구축*에서 "레이아웃 매니저 사용"을 참조하십시오.

1 컨텐트 창에 있는 HelloWorldFrame.java 파일로 돌아와서 Design 탭을 클릭하여 디자이너로 전환합니다.

2 컴포넌트 트리에서 jPanel1을 선택합니다.

3 Inspector의 Properties 탭을 선택합니다.

4 jPanel1의 레이아웃을 Inspector의 GridBagLayout로 변경합니다. 디자이너의 각 컴포넌트를 선택한 다음 각 컴포넌트가 채우는 그리드 영역에 주목합니다.

   GridBagLayout은 그리드에 다양한 크기의 컴포넌트를 배치하기에 적합합니다. *JBuilder를 이용한 애플리케이션 구축*에서 "GridBagLayout"을 참조하십시오.

5 애플리케이션을 저장하고 실행합니다.

6 애플리케이션 창의 크기를 변경한 다음 창의 크기가 바뀔 때 컴포넌트가 어떻게 재배치되는지 주목합니다.

7 "Hello World" 애플리케이션을 닫습니다.

JBuilder 개인용

11과 12 단계에서는 JBuilder 전문가용 및 기업용 버전의 기능을 사용합니다. JBuilder 개인용 에디션을 사용하는 경우 먼저 이 자습서를 마스터해야 합니다.

11 단계: 배포용 애플리케이션 준비

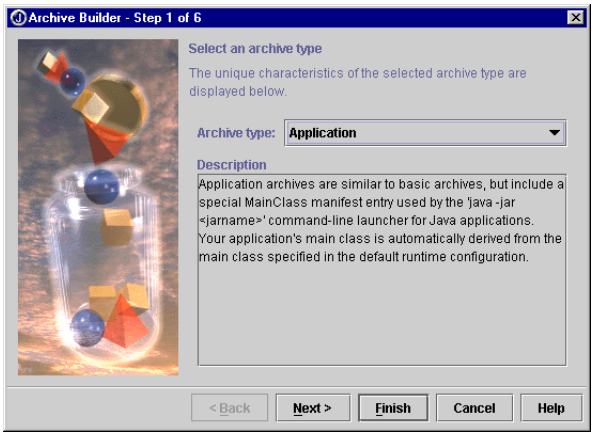
JBuilder 전문가용 및 기업용

JBuilder 전문가용 또는 기업용 자습서를 갖고 있다면 그대로 사용하십시오.

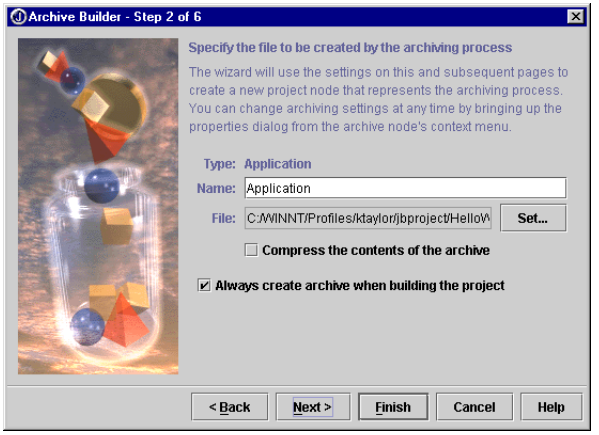
Archive Builder는 프로그램을 배포하는데 필요한 모든 파일을 수집하여 Java 아카이브 파일(JAR 파일)에 보관할 수 있습니다.

다음과 같은 방법으로 애플리케이션을 배포합니다.

1. Archive Builder를 열려면 Wizards|Archive Builder를 선택합니다.
2. Archive Type 드롭다운 목록에서 Application을 선택합니다.



3. 2 단계로 가려면 Next를 클릭합니다.
4. 이 아카이브 파일에 기본 이름과 JAR 파일의 기본 경로를 그대로 사용합니다. HelloWorld.jar는 HelloWorld 디렉토리에 저장됩니다.

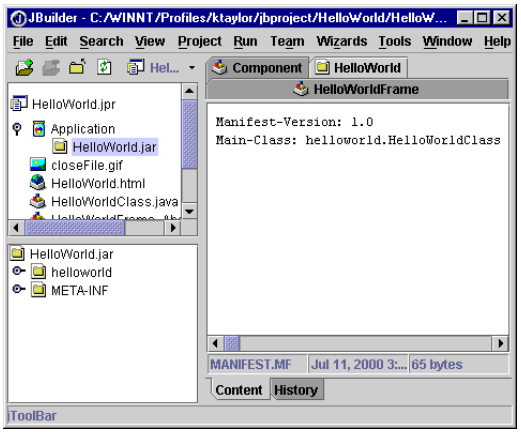


5 3 ~6 단계의 기본값을 그대로 사용합니다.

6 Finish를 클릭하여 Archive Builder를 닫습니다. Application이라는 아카이브 노드가 프로젝트 창에 표시됩니다.마우스 오른쪽 버튼을 클릭하고 Properties를 선택하여 이 파일을 수정할 수 있습니다.

7 Project|Make Project를 선택하여 애플리케이션을 컴파일하고 JAR 파일을 생성합니다. Archive Builder는 Project|Project Properties|Paths를 선택하여 프로젝트의 출력 경로에 있는 모든 파일을 JAR 파일에 통합합니다.

8 노드를 확장하려면 Application 아카이브 노드 옆에 있는 확장 아이콘을 클릭하여 HelloWorld.jar 아카이브 파일을 봅니다.프로젝트 창에서 JAR 파일을 더블 클릭합니다. 목록 파일이 컨텐트 창에 표시되고 JAR 파일의 내용이 구조 창에 표시됩니다.



9 프로젝트를 저장합니다.

배포에 대한 자세한 내용은 *JBuilder를 이용한 애플리케이션 구축*에서 "Java 프로그램 배포"를 참조하십시오.

12 단계: 명령 줄에서 배포한 애플리케이션 실행

JBuilder 전문가용 및 기업용

JBuilder 전문가용 또는 기업용 자습서를 갖고 있다면 그대로 사용하십시오.

배포한 애플리케이션을 테스트하려면 명령줄에서 JAR 파일을 실행하면 됩니다.

참고 **java** 명령이 들어 있는 jdk/bin/ 디렉토리는 경로 위에 있어야 합니다.경로 위에 **java**가 있는 경우, 명령줄에 java를 입력하면 그 명령을 설명하는 내용이 표시되어야 합니다.경로에 없는 경우, jdk/bin/ 디렉토리 안에서 애플리케이션을 실행해야 합니다.

1. 명령줄 창을 엽니다.
2. 프롬프트 상의 한 줄에 다음 명령을 입력합니다.

```
java -classpath /<home>/jbproject/HelloWorld/HelloWorld.jar
helloworld.HelloWorldClass
```

참고 Windows에서는 백슬래시(₩)를 사용합니다.

명령은 다음과 같은 형식을 가집니다.

```
java -classpath package-name.main-class-name
```

이 예제에서,

- java = Java 애플리케이션용 런처.
- -classpath = 애플리케이션 클래스와 리소스에 대한 검색 경로를 설정하는 옵션.
- classpath = /<home>/jbproject/HelloWorld/HelloWorld.jar

  클래스 경로에는 JAR 파일과 이 파일의 정확한 위치가 포함되어야 합니다. 이 예에서는 JAR 파일이 프로젝트 디렉토리 HelloWorld에 있습니다. Archive Builder는 기본적으로 이 디렉토리에 파일을 저장합니다.
- <home> = 홈 디렉토리, 예를 들면 c:₩winnt₩profiles₩<username>
- package-name = helloworld
- main-class-name = HelloWorldClass

JAR 파일에 대한 내용은 http://java.sun.com/docs/books/tutorial/jar/index.html의 JAR 자습서를 참조하십시오.

3. "Hello World" 애플리케이션이 로드 및 실행됩니다.

축하합니다! JBuilder를 사용하여 첫 번째 애플리케이션을 생성했습니다. 이제 JBuilder의 개발 환경에 익숙해졌으므로 JBuilder의 여러 가지 시간 절약(time-saving) 기능을 통해 프로그램을 더 쉽게 만들 수 있을 것입니다.

JBuilder의 UI 디자인 관련 기타 자습서에 대해서 "Text 에디터" "GridBagLayout," 및 "중첩 레이아웃"을 참조하십시오.

HelloWorld 소스 코드

다음 소스 코드를 사용할 수 있습니다.

Source code for HelloWorldFrame.java

```
package helloworld;

import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import javax.swing.border.*;

/**
 * Title:HelloWorld
 * Description:This is the "Hello World" tutorial.
 * Copyright:Copyright (c) 2001
 * Company:MyCompany
 * @author        MyName
 * @version 1.0
 */

public class Frame1 extends JFrame {
  JPanel contentPane;
  JMenuBar jMenuBar1 = new JMenuBar();
  JMenu jMenuFile = new JMenu();
  JMenuItem jMenuFileExit = new JMenuItem();
  JMenu jMenuHelp = new JMenu();
  JMenuItem jMenuHelpAbout = new JMenuItem();
  JToolBar jToolBar = new JToolBar();
  JButton jButton1 = new JButton();
  JButton jButton2 = new JButton();
  JButton jButton3 = new JButton();
  ImageIcon image1;
  ImageIcon image2;
  ImageIcon image3;
  JLabel jLabel1 = new JLabel();
  BorderLayout borderLayout1 = new BorderLayout();
  JPanel jPanel1 = new JPanel();
  Border border1;
  JLabel jLabel1 = new JLabel();
  JButton jButton4 = new JButton();
  GridBagLayout gridBagLayout1 = new GridBagLayout();
```

```
  //Construct the frame
  public HelloWorldFrame() {
   enableEvents(AWTEvent.WINDOW_EVENT_MASK);
  try {
    jbInit();
    }
    catch(Exception e) {
      e.printStackTrace();
    }
  }
  /**Component initialization*/
  private void jbInit() throws Exception {
    image1 = new
ImageIcon(helloworld.HelloWorldFrame.class.getResource("openFile.gif"));
    image2 = new
ImageIcon(helloworld.HelloWorldFrame.class.getResource("closeFile.gif"));
    image3 = new
ImageIcon(helloworld.HelloWorldFrame.class.getResource("help.gif"));
    //
setIconImage(Toolkit.getDefaultToolkit().createImage(HelloWorldFrame.class.getRes
ource
                                                ("[Your Icon]")));
    contentPane = (JPanel) this.getContentPane();
    border1 = BorderFactory.createLineBorder(Color.gray,2);
    contentPane.setLayout(borderLayout1);
    this.setSize(new Dimension(400, 300));
    this.setTitle("Hello World");
    statusBar.setText(" ");
    jMenuFile.setText("File");
    jMenuFileExit.setText("Exit");
    jMenuFileExit.addActionListener(new ActionListener()  {
      public void actionPerformed(ActionEvent e) {
        jMenuFileExit_actionPerformed(e);
      }
    });
    jMenuHelp.setText("Help");
    jMenuHelpAbout.setText("About");
    jMenuHelpAbout.addActionListener(new ActionListener()  {
      public void actionPerformed(ActionEvent e) {
        jMenuHelpAbout_actionPerformed(e);
      }
    });
    jButton1.setIcon(image1);
    jButton1.setToolTipText("Open File");
    jButton2.setIcon(image2);
    jButton2.setToolTipText("Close File");
    jButton3.setIcon(image3);
    jButton3.setToolTipText("Help");
    jPanel1.setBackground(Color.white);
    jPanel1.setBorder(border1);
    jPanel1.setLayout(gridBagLayout1);
    jLabel1.setFont(new java.awt.Font("Serif", 3, 28));
    jLabel1.setForeground(Color.blue);
    jLabel1.setText("Hello World!");
    jButton4.setText("Push Me");
    jButton4.addActionListener(new java.awt.event.ActionListener() {
      public void actionPerformed(ActionEvent e) {
        jButton4_actionPerformed(e);
      }
    });
    jToolBar.add(jButton1);
    jToolBar.add(jButton2);
    jToolBar.add(jButton3);
    jMenuFile.add(jMenuFileExit);
```

```
    jMenuHelp.add(jMenuHelpAbout);
    jMenuBar1.add(jMenuFile);
    jMenuBar1.add(jMenuHelp);
    this.setJMenuBar(jMenuBar1);
    contentPane.add(jPanel1, BorderLayout.CENTER);
    contentPane.add(statusBar, BorderLayout.SOUTH);
    contentPane.add(jPanel1, BorderLayout.CENTER);
    jPanel1.add(jLabel1,  new GridBagConstraints(0, 1, 1, 1, 0.0, 0.0
        ,GridBagConstraints.WEST, GridBagConstraints.NONE, new Insets(45, 125,
        102, 110), 10, -6));
    jPanel1.add(jButton1, new GridBagConstraints(0, 0, 1, 1, 0.0, 0.0,
        ,GridBagConstraints.CENTER, GridBagConstraints.NONE, new Insets(46, 148,
        0, 149), 16, 0));
  }

  /**File | Exit action performed*/
public void jMenuFileExit_actionPerformed(ActionEvent e) {
     System.exit(0);
  }
  /**Help | About action performed*/
public void jMenuHelpAbout_actionPerformed(ActionEvent e) {
    HelloWorldFrame_AboutBox dlg = new HelloWorldFrame_AboutBox(this);
Dimension dlgSize = dlg.getPreferredSize();
Dimension frmSize = getSize();
Point loc = getLocation();
    dlg.setLocation((frmSize.width - dlgSize.width) / 2 + loc.x, (frmSize.height -
        dlgSize.height) / 2 + loc.y);
dlg.setModal(true);
dlg.show();
  }
  /**Overridden so we can exit when window is closed*/
  protected void processWindowEvent(WindowEvent e) {
    super.processWindowEvent(e);
    if (e.getID() == WindowEvent.WINDOW_CLOSING) {
      jMenuFileExit_actionPerformed(null);
    }
  }

void jButton4_actionPerformed(ActionEvent e) {
  jLabel1.setForeground(new Color(255,0,0));
  }
}
```

Source code for HelloWorldClass.java

```
package helloworld;

import javax.swing.UIManager;
import java.awt.*;

/**
 * Title:HelloWorld
 * Description:This is the "Hello World" tutorial.
 * Copyright:Copyright (c) 2001
 * Company:MyCompany
 * @author        MyName
 * @version 1.0
 */

public class HelloWorldClass {
  boolean packFrame = false;

  /**Construct the application*/
  public HelloWorldClass() {
    HelloWorldFrame frame = new HelloWorldFrame();
  //Validate frames that have preset sizes
  //Pack frames that have useful preferred size info, e.g., from their layout
  if (packFrame)
    frame.pack();
    }
else {
    frame.validate();
    }
    //Center the window
Dimension screenSize = Toolkit.getDefaultToolkit().getScreenSize();
Dimension frameSize = frame.getSize();
if (frameSize.height > screenSize.height) {
   frameSize.height = screenSize.height;
    }
if (frameSize.width > screenSize.width) {
   frameSize.width = screenSize.width;
    }
    frame.setLocation((screenSize.width - frameSize.width) / 2, (screenSize.height -
      frameSize.height) / 2);
  frame.setVisible(true);
  }
  /**Main method*/
  public static void main(String[] args) {
  try {
      UIManager.setLookAndFeel(UIManager.getSystemLookAndFeelClassName());
    }
    catch(Exception e) {
      e.printStackTrace();
    }
    new HelloWorldClass();
  }
}
```

Source code for HelloWorldFrame_AboutBox.java

```
package helloworld;

import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import javax.swing.border.*;

/**
 * Title:HelloWorld
 * Description:This is the "Hello World" tutorial.
 * Copyright:Copyright (c) 2001
 * Company:MyCompany
 * @author        MyName
 * @version 1.0
 */

public class HelloWorldFrame_AboutBox extends JDialog implements ActionListener
{

  JPanel jPanel3 = new JPanel();
  JPanel jPanel3 = new JPanel();
  JPanel insetsPanel1 = new JPanel();
  JPanel insetsPanel2 = new JPanel();
  JPanel insetsPanel3 = new JPanel();
  JButton jButton3 = new JButton();
  JLabel jLabel1 = new JLabel();
  JLabel jLabel1 = new JLabel();
  JLabel jLabel1 = new JLabel();
  JLabel jLabel1 = new JLabel();
  JLabel jLabel1 = new JLabel();
  BorderLayout borderLayout1 = new BorderLayout();
  BorderLayout borderLayout2 = new BorderLayout();
  FlowLayout flowLayout1 = new FlowLayout();
  GridLayout gridLayout1 = new GridLayout();
  String product = "";
  String version = "2.0";
  String copyright = "Copyright (c) 2001";
  String comments = "";
  public HelloWorldFrame_AboutBox(Frame parent) {
    super(parent);
   enableEvents(AWTEvent.WINDOW_EVENT_MASK);
  try {
    jbInit();
    }
    catch(Exception e) {
      e.printStackTrace();
    }
    pack();
  }
  /**Component initialization*/
  private void jbInit() throws Exception {
    //imageLabel.setIcon(new
ImageIcon(HelloWorldFrame_AboutBox.class.getResource("[Your
        Image]")));
    this.setTitle("About");
    setResizable(false);
    panel1.setLayout(borderLayout1);
    panel2.setLayout(borderLayout2);
    insetsPanel1.setLayout(flowLayout1);
    insetsPanel2.setLayout(flowLayout1);
    insetsPanel2.setBorder(BorderFactory.createEmptyBorder(10, 10, 10, 10));
    gridLayout1.setRows(4);
    gridLayout1.setColumns(1);
```

```
    label1.setText(product);
    label2.setText(version);
    label3.setText(copyright);
    label4.setText(comments);
    insetsPanel3.setLayout(gridLayout1);
    insetsPanel3.setBorder(BorderFactory.createEmptyBorder(10, 60, 10, 10));
    button1.setText("Ok");
    button1.addActionListener(this);
    insetsPanel2.add(imageLabel, null);
    panel2.add(insetsPanel2, BorderLayout.WEST);
    this.getContentPane().add(panel1, null);
    insetsPanel3.add(label1, null);
    insetsPanel3.add(label2, null);
    insetsPanel3.add(label3, null);
    insetsPanel3.add(label4, null);
    panel2.add(insetsPanel3, BorderLayout.CENTER);
    insetsPanel1.add(button1, null);
    panel1.add(insetsPanel1, BorderLayout.SOUTH);
    panel1.add(panel2, BorderLayout.NORTH);
  }
  /**Overridden so we can exit when window is closed*/
  protected void processWindowEvent(WindowEvent e) {
    if (e.getID() == WindowEvent.WINDOW_CLOSING) {
      cancel();
    }
    super.processWindowEvent(e);
  }
  /**Close the dialog*/
  void cancel() {
    dispose();
  }
  /**Close the dialog on a button event*/
      public void actionPerformed(ActionEvent e) {
    if (e.getSource() == button1) {
      cancel();
    }
  }
}
```

Chapter

24

자습서:애플릿 만들기

이 자습서에서는 JBuilder 통합 개발 환경(IDE)을 사용하여 AWT 애플릿을 만드는 단계를 설명합니다. IDE 및 컴포넌트에 대한 자세한 내용은 Help 메뉴의 "JBuilder environment"를 참조하십시오.

자습서에서 다음 방법을 보여 줍니다.

- 애플릿용 프로젝트를 생성합니다.
- Applet 마법사를 사용하여 AWT 애플릿을 만듭니다.
- 애플릿을 컴파일하고 실행합니다.
- 애플릿의 UI를 사용자 지정합니다.
- Choice, Label 및 Button 같은 AWT 컴포넌트를 추가합니다.
- 소스 코드를 편집합니다.
- 애플릿을 배포합니다.
- HTML 파일을 수정합니다.
- 배포된 애플릿을 명령 줄에서 실행합니다.
- 애플릿을 테스트합니다.

팁 애플릿 소스 코드는 자습서의 끝에서 제공됩니다. FirstApplet 예제는 JBuilder 디렉토리의 samples/Tutorials/FirstApplet에 있습니다.

중요 JBuilder를 root 권한으로 설치했지만 일반 사용자로 실행하는 경우, 완전한 읽기/쓰기 권한을 가진 디렉토리에 전체 샘플 트리를 복사하여 실행해야 합니다.

중요 이 자습서를 시작하기 전에 중요한 애플릿 문제를 설명하는 개요를 읽어 보십시오.

이 자습서에서 사용된 설명서 규칙에 대한 내용은 1- 4페이지 "설명서 규칙"을 참조하십시오. 본 자습서의 개선에 관한 추가적인 제안 사항은 전자 우편을 통해 jpgpubs@borland.com으로 보내 주십시오.

개요

Java에 대한 브라우저 지원이 제한되는 애플릿을 디자인할 시기를 기억하는 것이 중요합니다. 이 작성과 관련하여 Internet Explorer 및 Netscape는 JDK 1.1.5를 지원합니다. 현재 브라우저는 JDK 1.1.7에 도입된 Swing 컴포넌트를 지원하지 않지만 앞으로는 지원할 수 있습니다. JDK 최신 버전으로 애플릿을 만드는 경우에는 브라우저가 지원하는 컴포넌트를 매우 주의 깊게 사용해야 합니다. 예를 들어, AWT 컴포넌트로만 애플릿을 개발하면 대부분의 경우 애플릿이 실행됩니다. AWT 컴포넌트에 변경 내용이 있을 수 있으며 이에 따라 애플릿 코드를 수정해야 할 수 있습니다. 예를 들어, JDK 1.1.x에는 JDK 1.3과 약간 다른 기능이 있을 수 있습니다. Java Console 오류 메시지를 확인하여 브라우저의 애플릿 문제를 해결할 수 있습니다. 브라우저가 지원하는 AWT 컴포넌트 및 JDK를 사용하면 가장 안전하게 애플릿을 디자인할 수 있습니다.

참고 JDK 1.1을 지원하는 브라우저로는 Netscape 4.06 이상과 Internet Explorer 4.01 이상이 있습니다. JDK 버전 지원은 플랫폼에 따라 다를 수 있습니다.

또 다른 옵션은 현재의 JDK와 Swing 컴포넌트를 사용하여 애플릿을 디자인하고 사용자에게 Java Plug-in을 제공하는 것입니다. 일반적으로 이 옵션은 회사 인트라넷처럼 통제된 환경에서만 가능합니다. 브라우저에 Java 2 SDK 1.3 런타임 환경(JRE)을 제공하는 Java Plug-in이 지원된 브라우저에서는 JDK 1.3 기반 애플릿을 실행할 수 있습니다. Java Plug-in으로 실행되는 애플릿 개발과 관련된 여러 가지 추가 단계가 있습니다. 자세한 내용을 보려면 Java Plug-in 홈 페이지, http://www.javasoft.com/products/plugin/index.html을 방문하십시오.

최신 브라우저는 JDK 1.1.5만 지원하지만 이 자습서에서는 JDK 1.3을 사용하고 AWT 컴포넌트로만 디자인하여 JDK 1.3의 새로운 기능은 사용하지 않습니다. 하지만 사용하는 컴포넌트를 주의 깊게 선택했기 때문에 애플릿은 여전히 실행됩니다. JDK 1.02에서 실행되는 이전 브라우저는 이 애플릿을 실행할 수 없는데, 이벤트 처리가 JDK 1.1에서 변경되었기 때문입니다. 이전 브라우저에서 애플릿을 사용하려면 JDK 1.02 이벤트 처리를 사용해야 합니다.

중요 JBuilder의 JDK 1.1.x 및 1.2 애플릿 실행에 대한 내용은 *웹 애플리케이션 개발자 안내서*의 "애플릿 작업" 장에서 "애플릿 실행"을 참조하십시오.

Good Evening 애플릿

이 자습서에서 만드는 "Good Evening" 애플릿에는 언어를 선택할 수 있는 드롭다운 리스트가 포함됩니다. German과 같은 언어를 선택하면 선택 항목 아래의 패널이 "Good Evening"의 독일어 표현인 "Guten Abend"로 변경됩니다.

자습서를 완료하면 JBuilder의 애플릿 뷰어에서 애플릿이 실행될 때 다음과 같이 나타납니다.

전체 애플릿 코드는 자습서의 끝에 있는 소스 코드를 참조하십시오.

애플릿과 배포에 대한 깊이 있는 정보는 *Web Application Developers Guide*의 "애플릿 작업"과 *Building Applications with JBuilder*의 "Java 프로그램 배포"를 참조하십시오.

1 단계:프로젝트 생성

이 자습서를 시작하기 전에 브라우저 지원, JDK 버전 및 애플릿 컴포넌트 같은 중요한 애플릿 문제를 설명하는 24- 2페이지 "개요"를 읽어 보십시오.

애플릿을 만들려면 애플릿을 저장할 프로젝트가 필요합니다. Project 마법사를 사용하여 다음과 같이 프로젝트를 만듭니다.

1. Project 마법사를 열려면 File|New Project를 선택합니다.
2. Project 마법사의 1 단계에서 프로젝트 및 디렉토리 이름을 다음과 같이 변경합니다.
   1. Project Name 필드: FirstApplet

참고 Project Name 필드에 입력할 때, 동일한 이름이 Project Directory Name 필드에 기본적으로 입력됩니다. 이 FirstApplet 프로젝트 디렉토리에 프로젝트가 저장됩니다.

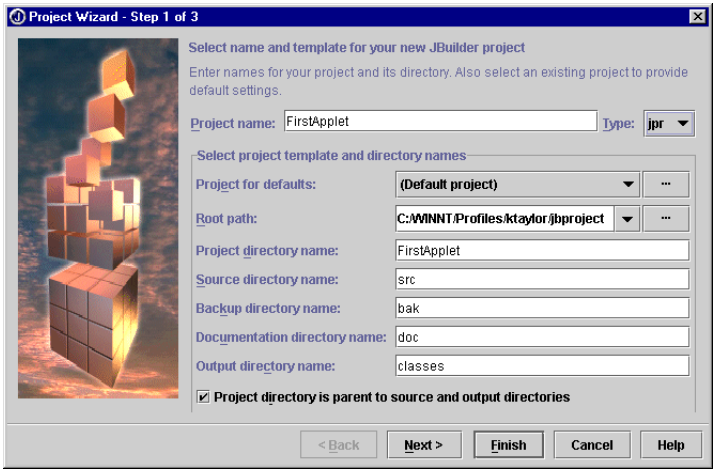
   2. Type (파일 타입) 필드:.jpr 또는 .jpx

참고 JBuilder는 프로젝트 파일에 .jpr 또는 .jpx 확장자를 사용합니다..jpr 파일 타입은 일반적인 목적에 유용합니다. XML 프로젝트 파일인 .jpx 파일 타입은 팀 개발 환경과 버전 제어에 사용됩니다. 이 자습서에서는 .jpr 확장자를 사용합니다.두 파일 타입 모두 이 자습서에 유효합니다.

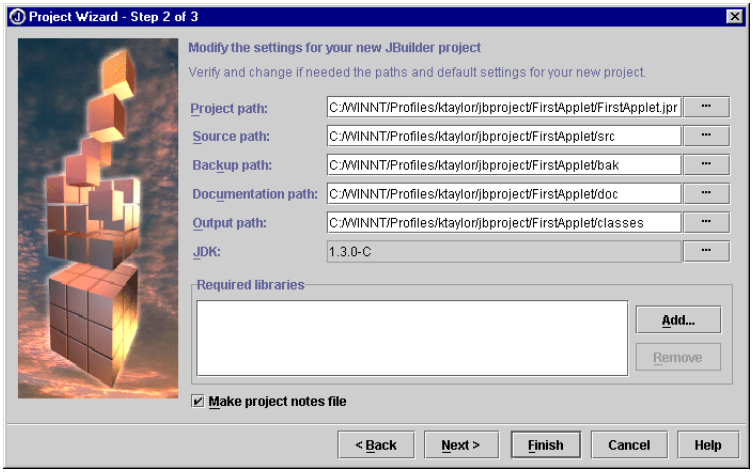
3 그 외에는 모두 기본값을 사용합니다.프로젝트가 저장된 root 경로를 기억해 두십시오.

참고 JBuilder에서 프로젝트는 기본적으로 /<home>/jbproject/ 디렉토리에 저장됩니다. 1- 4페이지 "설명서 규칙"을 참조하십시오.

프로젝트에 대한 자세한 내용은 *JBuilder를 이용한 애플리케이션 구축*에서 "프로젝트 생성 및 관리"를 참조하십시오.



4 Next를 클릭하여 Project 마법사의 2 단계로 이동합니다.



5 2 단계에서 프로젝트, 소스, 백업, 설명서, 출력 경로 및 JDK 버전에 대한 기본값을 사용합니다.프로젝트, 클래스, 소스 파일이 저장되는 위치를 기억해 두십시오. 또한 Make Project Notes File 옵션을 선택 표시해야 한다는 점도 기억해 두십시오. 이 옵션은 마법사의 3 단계에서 완성된 프로젝트 정보가 들어 있는 HTML 파일을 생성합니다.

팁 이전 버전의 JDK를 사용하여 애플릿을 만들려면 이 단계에서 JDK 버전을 변경하십시오. JBuilder 개인용은 JDK 스위칭을 지원하지 않지만 Configure JDKs 대화 상자(Tools|Configure JDKs)에서 기존 JDK를 편집할 수 있습니다. JDK 1.1.x의 경우에는 JFC의 JDK 1.1.x 관련 버전도 다운로드해야 합니다.

참조 사항

- JDK 스위칭이나 편집에 대한 내용 – *Building Applications with JBuilder*의 "프로젝트 속성 설정"
- JBuilder에서의 JDK 1.1.x 및 1.2 애플릿 실행에 대한 내용 – *웹 애플리케이션 개발자 안내서*의 "애플릿 작업" 장에서 "애플릿 실행"
- *JBuilder를 이용한 애플리케이션 구축*의 "프로젝트 생성 및 관리"에서 "JBuilder의 경로 구성 방법" 및 "내 파일의 위치"

6 Next를 클릭하여 마법사의 3 단계로 이동합니다.

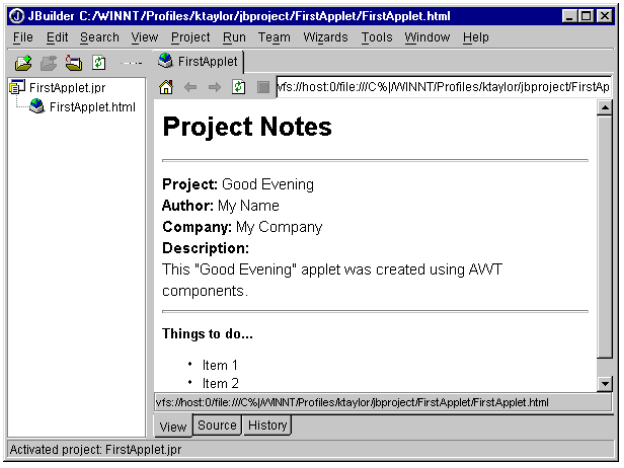
7 3 단계의 해당 필드에서 다음과 같이 변경합니다.

1 Title 필드에 GoodEvening을 입력합니다.

2 해당 옵션 필드에 이름, 회사명, 애플리케이션 설명을 입력합니다.이 내용은 프로젝트 HTML 파일에서 볼 수 있으며 소스 코드에서 옵션 헤더 주석으로 나타납니다.

8 Finish 버튼을 클릭합니다.

마법사는 AppBrowser의 프로젝트 창에 나타나는 프로젝트 파일과 프로젝트 노트 파일, FirstApplet.jpr과 FirstApplet.html을 생성합니다. HTML 파일을 더블 클릭하여 컨텐트 창에서 프로젝트 노트를 봅니다.



2 단계:소스 파일 생성

Applet 마법사는 .java 파일과 애플릿 HTML 파일을 생성하고 Project 마법사로 생성한 프로젝트에 저장합니다.

애플릿의 이러한 소스 파일을 생성하려면 다음 단계를 수행하십시오.

1. File|New를 선택하고 객체 갤러리의 Web 탭을 선택합니다. JBuilder 개인용에서는 Applet 아이콘이 New 페이지에 있습니다.
2. Applet 아이콘을 더블 클릭하여 Applet 마법사를 엽니다.
3. 1 단계의 기본 패키지 이름, firstapplet을 사용합니다. 기본적으로 마법사는 프로젝트 파일 이름, FirstApplet.jpr에서 패키지 이름을 취합니다.
4. Class 필드에 GoodEveningApplet을 입력합니다. 이것은 대소문자 구별 Java 클래스 이름입니다.

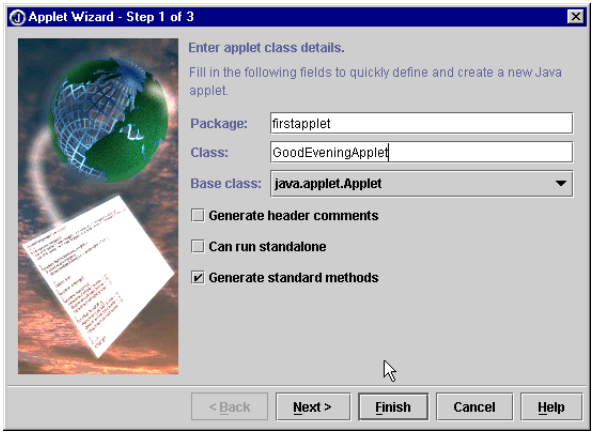
참고 전체 클래스 이름(패키지 이름 + 클래스 이름)은 firstapplet.GoodEveningApplet.class입니다. 클래스 파일은 Java 패키지 구조firstapplet/GoodEveningApplet.class에 저장됩니다.

5. 기본 베이스 클래스인 java.applet.Applet을 사용합니다.

주의 javax.swing.JApplet을 사용하여 애플릿을 만들면 해당 애플릿은 브라우저에서 실행되지 않습니다. 이 자습서를 작성할 시기에는 브라우저에서 Swing이 아직 지원되지 않고 있습니다. 자세한 내용은 24- 2페이지 "개요"을 참조하십시오.

6. Generate Standard Method를 선택 표시합니다.

Applet 마법사의 1 단계는 다음과 같습니다.



7 2 단계로 이동하려면 Next를 클릭합니다. 이 단계에서는 애플릿에 매개변수를 추가할 수 있습니다. 마법사는 애플릿의 HTML 파일에 있는 <applet> 태그 안에 <param> 태그를 추가하고 소스 코드에 매개변수를 처리하는 코드도 삽입합니다. 애플리케이션의 명령줄 인수와 동일한 애플릿 매개변수를 통해 애플릿을 사용자 지정할 수 있습니다. 임의의 매개변수는 추가하지 마십시오.

참조 사항 http://www.java.sun.com/docs/books/tutorial/applet/appletsonly/param.html의 "Defining and using parameters"

8 Next를 클릭하여 Applet 마법사의 3 단계로 이동합니다.

9 3 단계에서는 다음과 같이 변경합니다.

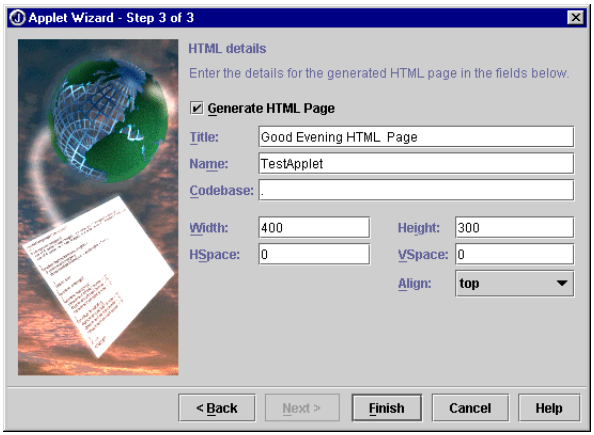
1 기본 옵션, Generate HTML Page를 사용합니다. 이 옵션을 선택하면 애플릿을 호출하는 HTML 파일이 생성됩니다. 이 HTML 파일에는 <applet> 태그 및 다양한 속성이 포함됩니다.

2 Title: Good Evening HTML 페이지

제목은 애플릿을 실행할 때 웹 브라우저 창에 표시됩니다.

3 다른 모든 속성에 대해서는 기본값을 사용합니다.

Applet 마법사의 3 단계는 다음과 같습니다.



HTML 파일의 <applet> 태그에 다음 속성 중 일부가 있을 수 있습니다.

codebase	이 옵션 속성은 브라우저에서 필요한 클래스 파일을 찾기 위해 검색하는 애플릿 HTML 파일에 상대적인 경로를 지정합니다. "."의 값은 애플릿을 실행하는 HTML 파일과 동일한 디렉토리를 지정합니다. 클래스 파일이 HTML 파일과 다른 디렉토리에 있을 때 codebase 속성이 필요합니다.
code	JBuilder의 Applet 마법사에서 자동으로 삽입하는 이 필수 속성은 init() 메소드를 포함하는 애플릿 클래스의 전체 클래스 이름(패키지 이름 + 클래스 이름)입니다. 이 속성은 생성될 때 HTML 파일에 표시됩니다. 다음 예제에서 전체 클래스 이름은 firstapplet.GoodEveningApplet.class입니다.
archive	Applet 마법사에서 생성한 코드에 포함되지 않는 이 옵션 속성은 애플릿이 JAR, ZIP 또는 CAB 파일로 배포되는 경우 필요합니다. 아카이브 파일은 codebase에서 지정한 디렉토리에 있어야 합니다.
name	이 옵션 속성은 애플릿의 이름을 지정합니다.
width/height	이 필수 속성은 애플릿 디스플레이 영역의 너비와 높이를 픽셀 단위로 결정합니다.
hspace/vspace	이 옵션 속성은 애플릿 주위의 가로 패딩(왼쪽 및 오른쪽 여백)과 세로 패딩(상단 및 하단 여백)을 픽셀 단위로 결정합니다.
align	이 옵션 속성은 HTML 페이지에서 애플릿의 정렬을 결정합니다.

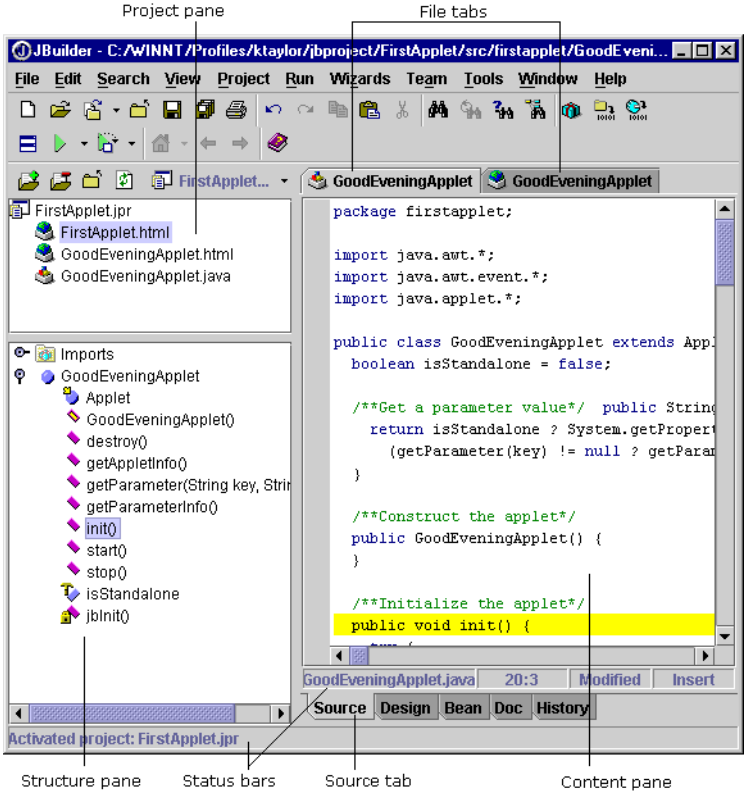
중요 codebase, code, archive 및 name의 값은 따옴표로 묶어 표시하고 대/소문자를 구분해야 합니다.

참조 사항 *웹 애플리케이션 개발자 안내서*의 "애플릿 작업"에 있는 "<applet> 태그 속성"

10 Finish를 선택하여 Applet 마법사를 닫습니다. 두 가지 파일 GoodEveningApplet.java와 GoodEveningApplet.html이 생성되어 프로젝트에 추가된다는 점을 기억해 두십시오.

참고 JBuilder 전문가용 및 기업용 에디션에서 Enable Source Package Discovery And Compilation 옵션이 Project Properties 대화 상자(Project|Project Properties)의 General 페이지에 활성화되면 firstapplet이라는 자동 소스 패키지 노드도 프로젝트 창에 나타납니다.

11 각 파일을 더블 클릭하고 컨텐트 창에서 Source 탭을 선택하여 생성된 코드를 봅니다.



GoodEveningApplet.java를 보면서 다음 사항을 기억해 두십시오.

- init() 메소드를 포함합니다. 애플릿 HTML 파일은 애플릿이 실행하는 init() 메소드를 포함하는 클래스를 호출해야 합니다.
- 패키지 이름 firstapplet이 코드의 첫 줄입니다. 클래스 파일은 Java 규칙에 따라 firstapplet 디렉토리에 저장됩니다.
- import 문은 Swing이 아니라 AWT 패키지를 import합니다.

```
import java.awt.*;
import java.awt.event.*;
import java.applet.*;
```

GoodEveningApplet.html을 보면서 마법사가 code 값, firstapplet.GoodEveningApplet.class를 삽입했다는 점을 기억해 두십시오.

12 소스 파일과 프로젝트 파일을 저장하려면 File|Save All을 선택합니다.

참고 기본적으로 JBuilder는 소스 파일을 다음 위치에 저장합니다.
/<home>/jbproject/FirstApplet/src/firstapplet/

애플릿 HTML 파일은 다음 위치의 classes 디렉토리에 저장됩니다.

/<home>/jbproject/FirstApplet/classes/

컴파일한 후 클래스 파일은 출력 경로인 /<home>/jbproject/FirstApplet/classes/firstapplet/에 저장됩니다.

JBuilder는 파일을 저장할 때 항상 패키지 계층 구조를 따릅니다. 다음 예제에서는 소스 파일과 클래스 파일이 소스 및 출력 경로의 firstapplet 디렉토리 내에 저장되어 firstapplet 패키지 구조를 반영합니다. 이 경로는 Project Properties 대화 상자(Project|Project Properties)의 프로젝트에 대한 설정입니다. 이 자습서에서는 Project 마법사 2 단계의 기본 JBuilder 경로를 사용합니다.

3 단계: 애플릿 컴파일 및 실행

이제 애플릿을 컴파일하고 실행합니다.

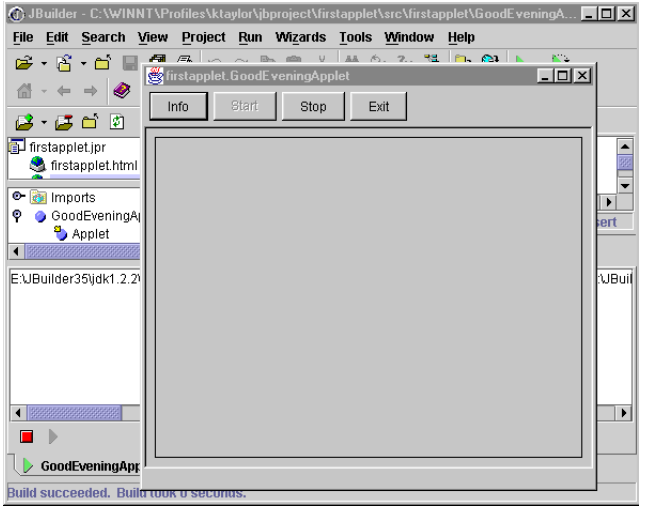
중요 JBuilder에서의 JDK 1.1.x 및 1.2 애플릿 실행에 대한 내용은 *웹 애플리케이션 개발자 안내서*의 "애플릿 작업" 장에 있는 "애플릿 실행"을 참조하십시오.

1 Run|Run Project를 선택하거나 Run 버튼을 클릭하여 애플릿을 컴파일하고 실행합니다. Run 메뉴와 Run 아이콘은 JBuilder의 애플릿 뷰어인 AppletTestbed에서 애플릿을 실행합니다.

팁 프로젝트 창에서 GoodEveningApplet.html을 마우스 오른쪽 버튼으로 클릭하고 Run을 선택해도 됩니다. Sun의 **appletviewer**에서 애플릿이 실행됩니다.

애플릿을 실행하면 AppBrowser의 하단에 메시지 창이 나타나 컴파일 타임 오류를 표시합니다. 오류를 수정하고 애플릿을 다시 실행합니다.

애플릿이 다음과 같이 나타납니다.



Project Properties 대화 상자의 Run 페이지에 있는 Applet 탭에서 애플릿의 실행 설정을 변경할 수 있습니다. 이 대화 상자에 액세스하려면 Project|Project Properties를 선택하거나 FirstApplet.jpr을 마우스 오른쪽 버튼으로 클릭하고 Properties를 선택합니다. Run 페이지의 설정은 Run 메뉴와 툴바에 있는 Run 아이콘의 동작을 제어합니다. Main Class 옵션을 선택하여 JBuilder의 AppletTestbed에서 애플릿을 실행합니다. HTML 옵션을 선택하여 Sun의 **appletviewer**에서 애플릿을 실행합니다. Applet 마법사를 사용하여 애플릿을 만들면 기본적으로 Main Class 옵션이 설정되어 있습니다.

중요 애플릿은 .java 파일에서가 아니라 init() 메소드를 포함하는 클래스를 호출하는 HTML 파일에서 실행됩니다. Applet 마법사의 1 단계에서 Can Run Standalone 옵션을 선택하지 않은 경우 .java 파일을 실행하려고 하면 다음과 같은 오류 메시지가 나타납니다.

```
java.lang.NoSuchMethodError:main
Exception in thread "main"
```

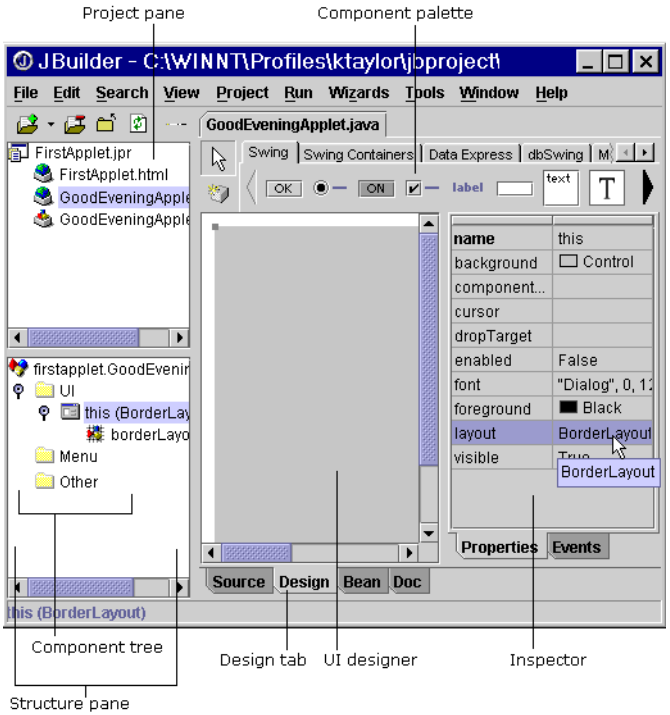
2. "Good Evening" 애플릿에서 Exit를 선택하여 애플릿을 닫습니다.
3. 메시지 창에서 GoodEveningApplet 탭을 마우스 오른쪽 버튼으로 클릭하고 Remove "GoodEveningApplet" Tab을 선택하여 런타임 메시지를 닫습니다.

4 단계: 애플릿의 사용자 인터페이스 사용자 지정

Applet 마법사에서 애플릿 셸을 생성했으므로 다음 단계에 따라 다양한 컴포넌트로 애플릿을 사용자 지정할 수 있습니다.

1. 컨텐트 창의 하단에 있는 Design 탭을 클릭하여 GoodEveningApplet.java를 디자인 뷰로 변경합니다. UI 디자이너가 컨텐트 창에 나타나고 그 위에는 컴포넌트 팔레트, 오른쪽에는 Inspector가 나타납니다.컴포넌트 팔레트를 사용하여 UI에 컴포넌트를 추가하고 Inspector를 사용하여 속성을 수정하고 코드에 이벤트를 추가

합니다. 구조 창에는 UI, Menu 및 Other와 같은 폴더가 있는 컴포넌트 트리가 들어 있습니다.



2 this 레이아웃을 Inspector의 BorderLayout으로 변경합니다.

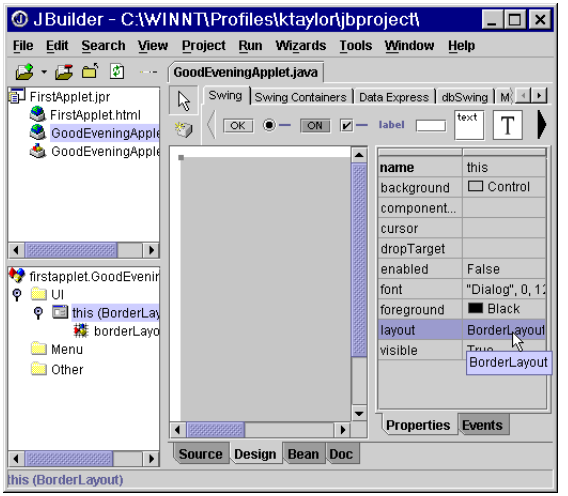
1 구조 창의 컴포넌트 트리에서 this를 선택합니다.

2 Inspector의 Properties 페이지에서 layout 속성의 오른쪽을 클릭합니다. 레이아웃의 드롭다운 리스트에서 BorderLayout을 선택합니다.

BorderLayout은 North, South, East, West, Center 영역에 컨테이너의 컴포넌트를 정렬합니다. 컨테이너의 하나 이상의 모서리에 컴포넌트를 채우거나 컴포넌트로 컨테이너의 가운데를 채울 때 BorderLayout을 사용합니다. 이 레이아웃은 단일 컴포넌트가 컨테이너를 완전히 채우는 데 사용하는 레이아웃이기도 합니다.

주의 Inspector의 드롭다운 리스트에서 XYLayout(사용자 지정 Borland 레이아웃)을 선택하는 경우 JBuilder는 다음과 같은 import 문 ,com.borland.jbcl.layout.*을 소스 코드에 추가합니다. 배포 전에 더 이식성이 뛰어난 레이아웃으로 변경할 경우 이 import 문은 제거되지 **않습니다**. 이 import 문을 제거하지 않으면 배포된 애플릿은 가져올 jbcl 레이아웃을 찾기 때문에 실행되지 않습니다. 배포하기 전에 import 문을 수동으로 제거해야 합니다.

JBuilder의 UI 디자이너는 각 컨테이너에 대해 보통 AMT 부모 컨테이너의 레이아웃과 같은 기본 레이아웃 매니저를 사용합니다. Java AWT에서 모든 패널은 기본적으로 FlowLayout을 사용합니다. 패널의 레이아웃 매니저를 보려면 컴포넌트 트리의 확장 아이콘을 클릭하여 선택 사항을 확장합니다. 레이아웃 매니저는 부모 컨테이너 바로 아래의 트리에 항목으로 표시됩니다.

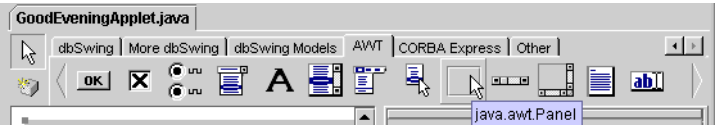


참조 사항

- *JBuilder를 이용한 애플리케이션 구축*의 "레이아웃 매니저 사용"
- 자습서: "중첩 레이아웃으로 UI 만들기"

3 UI 디자이너의 상단에 있는 컴포넌트 팔레트에서 AWT 탭을 클릭합니다. 컴포넌트 팔레트를 오른쪽으로 스크롤하여 AWT 탭을 찾습니다. Swing 컴포넌트와 달리 AWT 컴포넌트는 대부분의 브라우저에서 지원됩니다.

주의 UI 디자이너는 항상 컴포넌트 팔레트의 첫 페이지인 Swing 컴포넌트와 함께 열립니다. 실수로 이 Swing 컴포넌트를 선택하면 브라우저에서 애플릿이 실행되지 않습니다. 팔레트의 AWT 페이지에서 컴포넌트를 선택할 때는 주의하십시오.



4 this 패널에 두 개의 패널을 추가합니다. 상단 패널에는 선택할 수 있는 언어의 드롭다운 리스트와 리스트를 식별하는 레이블이 포함됩니다. 하단 패널에는 다양한 언어의 "Good Evening"이 포함됩니다.

1 컴포넌트 팔레트에서 AWT 탭을 선택합니다.

2 컴포넌트 팔레트에서 AWT Panel 컴포넌트를 *Shift+Click*합니다. *Shift+Click*을 사용하면 컴포넌트 팔레트에서 매번 선택하지 않고 같은 컴포넌트를 여러 번 가져다 놓을 수 있습니다. 이제 여러 패널을 this에 추가할 수 있습니다.

팁 커서를 컴포넌트 위에 놓고 툴팁으로 나타나는 이름을 확인합니다.

3 구조 창의 컴포넌트 트리에서 this를 더블 클릭하여 두 패널을 추가합니다. panel1과 panel2가 컴포넌트 트리의 this 아래에 추가됩니다.

4 컴포넌트 팔레트의 왼쪽에 있는 Selection Tool을 클릭하여 컴포넌트를 선택할 수 없게 합니다.

5 Inspector에서 패널의 constraints 속성을 확인합니다. 디자인의 상단 패널은 North이고 하단 패널은 Center입니다. constraints 속성을 확인하려면 다음과 같이 합니다.

1 컴포넌트 트리나 디자이너에서 패널을 선택합니다 .

2 Inspector 의 constraints 속성이 상단 패널은 North 로 , 하단 패널은 Center 로 설정되어 있는지 확인합니다 . 그렇지 않으면 constraints 속성 오른쪽에 있는 열을 클릭하고 드롭다운 리스트에서 올바른 제약 조건을 선택합니다 .

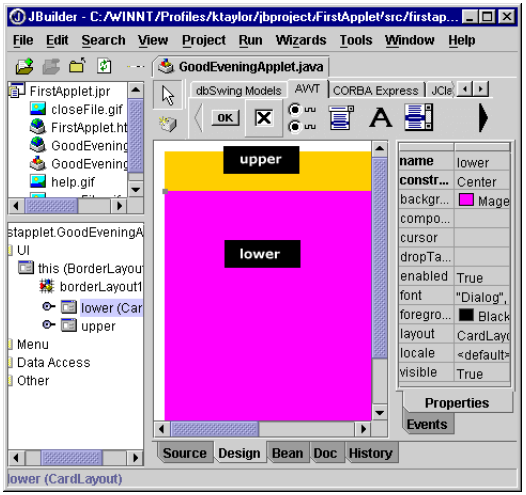
6 상단 패널의 이름을 upper로 재지정합니다.

1 컴포넌트 트리 또는 디자이너의 상단 패널을 선택하고 Inspector 의 name 속성 오른쪽에 있는 열을 더블 클릭합니다 .

2 upper 를 입력하고 *Enter* 를 누릅니다 .

팁 컴포넌트 트리에서 컴포넌트를 마우스 오른쪽 버튼으로 클릭하고 메뉴에서 Rename을 선택하여 컴포넌트 이름을 재정할 수도 있습니다.

7 하단 패널의 이름을 lower로 재지정합니다.



5 Inspector에서 upper의 배경색을 Orange로 변경합니다.

1 컴포넌트 트리 또는 디자이너에서 upper를 선택합니다.

2 Inspector에서 background 속성 오른쪽에 있는 열을 클릭합니다.

3 *아래쪽 화살표*를 클릭하여 색 드롭다운 리스트를 열고 목록에서 Orange를 선택합니다.

6 lower의 배경색을 Magenta로 변경합니다.

7 lower의 레이아웃을 CardLayout으로 변경합니다. CardLayout 패널에는 다섯 개의 패널이 포함되며 각 패널에는 다른 언어의 "Good Evening"이 있습니다.

CardLayout은 컴포넌트(일반적으로 패널)를 카드를 쌓는 것처럼 겹겹이 쌓아 놓습니다.한 번에 하나의 패널만 볼 수 있으며 다른 컨트롤을 사용하여 패널을 플립(flip)하여 위에 올 패널을 선택할 수 있습니다. CardLayout은 대체로 체크 박스나 리스트 같은 컴포넌트 제어와 관련이 있습니다.컴포넌트 제어 상태는 CardLayout이 표시하는 컴포넌트를 결정합니다. 사용자는 UI에서 골라서 선택합니다.

8 Panel 컴포넌트를 선택할 때 *Shift+Click*을 사용하여 lower 패널에 다섯 개의 패널(패널 1에서 5)을 추가합니다. 각 패널에는 다른 언어의 "Good Evening"이 있습니다. Selection Tool을 선택하여 Panel의 다중 선택을 비활성화합니다.

참고 잘못된 위치에 컴포넌트를 가져다 놓은 경우 컴포넌트 트리에서 컴포넌트를 선택하고 *Delete*를 누릅니다. 그런 다음 다시 추가합니다.

9 패널 1에서 5의 레이아웃을 BorderLayout으로 변경합니다.

1 *Shift+Click*이나 *Ctrl+Click*을 사용하여 컴포넌트 트리의 패널 1에서 5를 선택합니다.

2 Inspector에서 layout 속성을 BorderLayout으로 변경합니다. 다섯 개의 패널 모두 BorderLayout을 가집니다.

3 컴포넌트 트리나 UI 디자이너에서 다른 컴포넌트를 선택하고 모든 패널의 선택을 취소합니다.

10 Inspector에서 다섯 개 패널 각각의 배경색을 다른 색으로 변경합니다.

팁 Inspector에서 background 속성 오른쪽에 있는 생략 버튼을 클릭하고 색 슬라이더를 사용하여 색을 만듭니다.

11 파일과 프로젝트를 저장합니다.

12 GoodEveningApplet.html을 마우스 오른쪽 버튼으로 클릭하고 Run을 선택합니다. 애플릿이 Sun의 **appletviewer**에서 실행되면 CardLayout의 상단 패널 및 upper만 볼 수 있습니다. 다른 언어 패널은 드롭다운 리스트를 추가하고 이벤트를 목록 선택 사항에 추가한 후에 표시됩니다.

13 애플릿을 종료합니다.

14 GoodEveningApplet 탭을 마우스 오른쪽 버튼으로 클릭하고 Remove "GoodEveningApplet" Tab을 선택하여 메시지 창을 닫습니다.

5 단계: 애플릿에 AWT 컴포넌트 추가

이제 컴포넌트 팔레트를 사용하여 Label 및 Choice 컴포넌트를 디자인의 상단 패널, upper에 추가합니다.

1 컴포넌트 팔레트에서 AWT 탭을 선택하고 Choice 컴포넌트를 클릭합니다.

2 컴포넌트를 디자인의 상단 오렌지색 패널, upper로 가져다 놓습니다. 다음 두 가지 방법 중 하나를 사용합니다.

- 컴포넌트 트리에서 upper를 클릭합니다.
- UI 디자이너에서 upper를 클릭합니다.

컴포넌트 트리의 upper에 choice1이 추가됩니다.

3 컴포넌트 팔레트의 AWT 페이지에서 Label 컴포넌트를 선택하고 Choice 컴포넌트 왼쪽의 upper에 놓습니다. 컴포넌트 트리의 upper에 label1이 추가됩니다.

4 컴포넌트 트리에서 label1을 선택하고 다음 단계를 수행합니다.

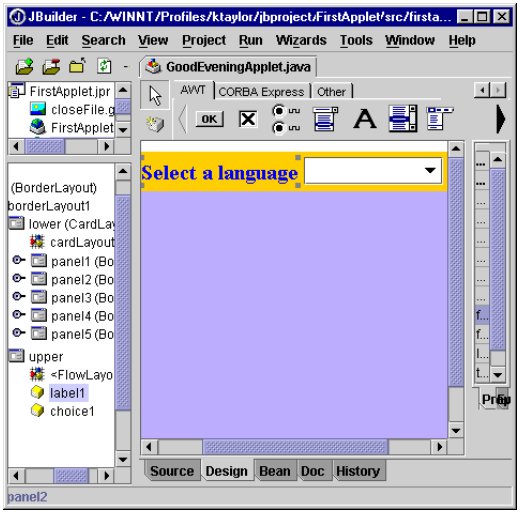
1 Inspector에서 text 속성 오른쪽의 열을 더블 클릭하여 기존 텍스트가 선택 표시됩니다. Select a language를 입력하고 *Enter*를 누릅니다. "Select a language"가 Choice 컴포넌트 옆에 있는 레이블에 표시됩니다.

2 글꼴을 설정하려면 font 속성 오른쪽에 있는 열을 클릭합니다. 생략 버튼을 클릭하여 Font 대화 상자를 엽니다.

3 글꼴 목록에서 Serif를 선택하고 Bold를 선택 표시합니다. Size 상자에 20을 입력한 다음 OK를 클릭합니다.

4 Inspector에서 foreground 속성 오른쪽에 있는 열을 클릭하고 텍스트 색을 설정합니다. *아래쪽 화살표*를 클릭하고 색상의 드롭 다운 리스트에서 Blue를 선택합니다.

디자인이 다음과 같이 나타납니다.



5 하단 CardLayout 패널의 각 패널(패널 1에서 5)에 AWT 레이블을 추가합니다. 각 레이블은 다른 언어의 "Good Evening"을 가지게 됩니다.

6 각 레이블을 다른 언어의 "Good Evening"으로 변경합니다. 먼저 컴포넌트 트리에서 각 패널 아래의 레이블을 선택하고 Inspector의 text 속성에 해당 언어로 "Good Evening"을 입력합니다. 레이블에 다음 언어를 사용하거나 직접 선택합니다.

- label2: Good Evening (English)
- label3: Guten Abend (German)
- label4: Oodgay vening eay (Pig Latin)
- label5: God Kväll (Swedish)
- label6: Gudday, Mate (Australian)

7 *Ctrl+Click*을 사용하여 label2에서 label6까지 선택하고 모든 레이블의 font 속성을 Bold로 변경하고 글꼴 크기는 24로 변경합니다. 레이블의

foreground 속성은 Black으로 변경합니다. 컴포넌트 트리 또는 UI 디자이너에서 컴포넌트를 클릭하여 레이블 선택을 해제합니다.

8 다음과 같이 Inspector에서 constraints 속성을 North, South, East, West 또는 Center로 변경하여 각 레이블의 위치를 변경합니다.

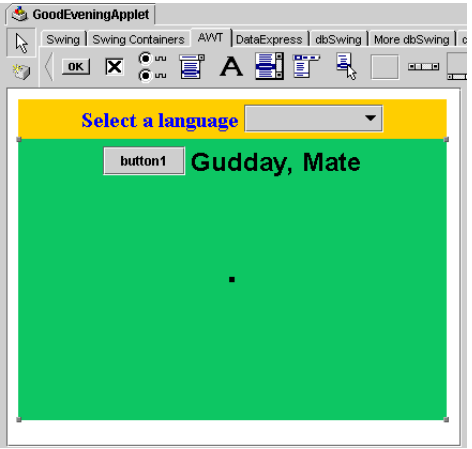
- label2: North
- label3: South
- label4: East
- label5: West
- label6: Center

참고 BorderLayout 패널에 있는 레이블의 위치를 기억해 두십시오. Center는 전체 패널을 채우고 North, South, East 및 West는 패널의 일부분만 채웁니다.

9 panel5의 레이아웃 매니저를 upper 패널에 사용된 것과 같은 레이아웃인 FlowLayout으로 변경합니다. "Gudday, Mate" 레이블이 패널의 상단 가운데로 옮겨 집니다. FlowLayout은 컴포넌트를 연속적으로 배열할 때 적합한 레이아웃입니다. 그 다음 FlowLayout에 의해 레이블과 동일한 행에 놓일 버튼을 추가합니다.

FlowLayout은 행의 왼쪽에서 오른쪽으로 컴포넌트를 정렬합니다. 행이 가득 차면 나머지 컴포넌트는 새 행으로 이동합니다.

10 AWT Button 컴포넌트를 클릭하고 "Gudday, Mate" 레이블 왼쪽으로 가져다 놓습니다. 디자이너에서 버튼을 다른 위치로 이동해 보십시오. FlowLayout에 의해 레이블이 있는 행 뒤로 오는 것을 알 수 있습니다. panel5의 애플릿 디자인은 다음과 같습니다.



11 File|Save All을 선택하여 프로젝트를 저장합니다.

레이아웃 및 해당 동작에 대한 자세한 내용은 *Building Applications with JBuilder*의 "레이아웃 매니저 사용"과 "자습서: 중첩 레이아웃"을 참조하십시오.

6 단계:소스 코드 편집

이 단계에서는 드롭다운 리스트에 언어를 추가한 다음 각 언어 패널을 Choice 컴포넌트에 연결하는 이벤트를 추가합니다.

1 다음과 같이 드롭다운 리스트를 위한 언어를 init() 메소드에 추가합니다.

1 컨텐트 창에서 Source 탭을 클릭하여 에디터에서 소스 코드를 변경합니다.

2 구조 창에서 init() 메소드를 선택합니다. 에디터에서 init() 메소드 코드가 구문 강조됩니다.

팁 구조 창을 클릭하고 메소드 이름을 입력하여 구조 창에서 메소드를 검색합니다.

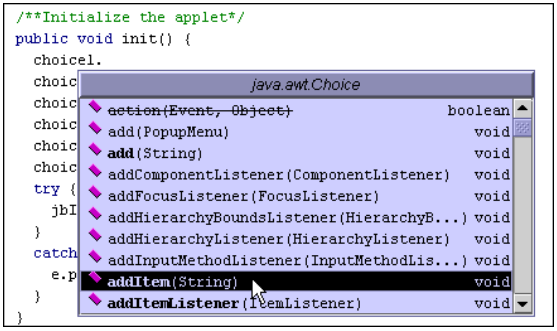
3 여는 중괄호 뒤와 try/catch 문 앞 사이에 커서를 두고 *Enter*를 눌러 빈 줄을 추가로 만듭니다.

팁 에디터를 확장하고 프로젝트 및 구조 창을 숨기려면 View|Toggle Curtain을 선택합니다.

4 굵게 표시된 다음 코드 블록을 init() 메소드에 추가합니다.

```
//initialize the applet
 public void init() {
   choice1.addItem("English");
   choice1.addItem("German");
   choice1.addItem("Pig Latin");
   choice1.addItem("Swedish");
   choice1.addItem("Australian");

  try {
    jbinit();
   }
    catch(Exception e) {
      e.printStackTrace();
    }
  }
```

팁 CodeInsight를 사용하여 코드를 완성할 수 있습니다. choice1을 입력합니다.그리고 팝업 창을 기다리거나 *Ctrl+spacebar*를 눌러 창을 호출합니다. choice 다음에 반드시 점(.)을 찍어야 합니다. 화살표 키를 사용하여 팝업 창에서 addItem(String)을 선택합니다. *Enter* 키를 누릅니다. Editor Options 대화 상자(Tools|Editor Options|CodeInsight)에서 CodeInsight를 구성할 수 있습니다.



코드에 구문 오류가 있으면 에디터에 입력할 때 구조 창에 Errors 폴더가 표시됩니다. 폴더를 열고 오류 메시지를 선택하면 소스 코드에서 오류가 구문 강조 표시됩니다.

참조 사항 *Building Applications with JBuilder*의 "오류 및 경고 메시지"

그 다음 언어 선택에 이벤트를 연결합니다. 드롭다운 리스트 Choice 컴포넌트에서 언어를 선택하면 선택한 언어로 "Good Evening"이 cardLayout 패널에 나타납니다.

2 다음과 같이 Choice 목록 이벤트를 연결합니다.

1 UI 디자이너로 돌아갑니다.

2 컴포넌트 트리에서 upper 아래에 있는 choice1을 선택합니다.

3 Inspector에서 Events 탭을 선택합니다.

4 itemStateChanged 이벤트 오른쪽을 더블 클릭합니다. JBuilder는 메소드 코드를 생성하고 메소드에 커서가 삽입되어 있는 소스 코드로 안내합니다.

```
void choice1_itemStateChanged(ItemEvent e) {

}
```

5 굵게 표시된 다음 코드를 추가하여 적절한 언어 패널을 언어 선택 사항에 연결합니다.

```
void choice1_itemStateChanged(ItemEvent e) {
  if ("English".equals(choice1.getSelectedItem())){
    cardLayout1.show(lower, "panel1");
  }
  else if ("German".equals(choice1.getSelectedItem())){
    cardLayout1.show(lower, "panel2");
  }
  else if ("Pig Latin".equals(choice1.getSelectedItem())){
    cardLayout1.show(lower, "panel3");
  }
  else if ("Swedish".equals(choice1.getSelectedItem())){
    cardLayout1.show(lower, "panel4");
  }
  else if ("Australian".equals(choice1.getSelectedItem())){
    cardLayout1.show(lower, "panel5");
  }
}
```

팁 코드 템플릿을 사용하여 코드를 생성할 수 있습니다. if를 입력하고 *Ctrl+J*를 눌러 코드 템플릿 팝업 창에 액세스합니다. 화살표 키를 사용하여 선택 사항을 탐색합니다. if-else if 템플릿을 선택하고 *Enter*를 누릅니다.

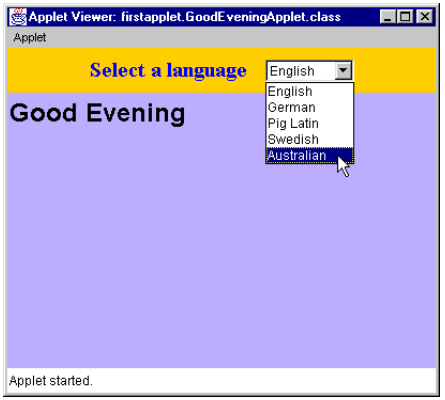
다음과 같이 코드가 생성됩니다.

```
if () {

}
else if{

}
```

3 File|Save All을 선택합니다.

4 프로젝트 창에서 GoodEveningApplet.html을 마우스 오른쪽 버튼으로 클릭하고 Run을 선택하여 애플릿을 실행합니다.

"Good Evening" 애플릿이 Sun의 **appletviewer**에서 실행됩니다.



오류가 있으면 AppBrowser 하단의 메시지 창에 나타납니다.도움말을 보려면 오류 메시지를 선택하고 *F1*을 누릅니다. 오류 메시지를 선택하면 에디터에서 코드가 구문 강조 표시됩니다. 경우에 따라 구문 강조 표시된 코드 라인의 앞이나 뒤에 오류가 있을 수도 있습니다. 오류를 수정하고, 프로젝트를 저장한 다음 애플릿을 다시 실행합니다.

5 드롭다운 리스트를 테스트합니다. 목록에서 선택한 언어는 그 아래에 있는 패널의 언어와 일치해야 합니다.

6 애플릿을 종료합니다.

panel5의 button1에 button 이벤트를 추가합니다. 버튼을 누르면 label6의 "Gudday, Mate" 텍스트가 빨간색으로 변경됩니다.

7 다음과 같이 버튼 이벤트를 추가합니다.

1 UI 디자이너로 전환합니다.

2 panel5의 button1을 선택합니다. Inspector의 Properties 페이지에서 버튼의 Label 속성을 button1에서 Push Me로 변경합니다. *Enter* 키를 누릅니다. 버튼이 텍스트에 맞추어 자동으로 크기가 조정됩니다.

3 Inspector의 Events 탭을 클릭하여 button1을 누를 때 발생하는 동작을 정의합니다.

4 ActionPerformed 이벤트의 오른쪽에 있는 열을 더블 클릭합니다. JBuilder는 if-else if 문 바로 아래의 ActionPerformed 이벤트에 다음과 같은 뼈대 코드가 추가된 에디터로 전환합니다.

```
void button1_actionPerformed(ActionEvent e) {

}
```

팁 디자이너에서 버튼을 더블 클릭해도 마찬가지입니다.

"Gudday, Mate"를 빨간색으로 변경하는 버튼 이벤트를 정의하는 코드를 입력합니다.

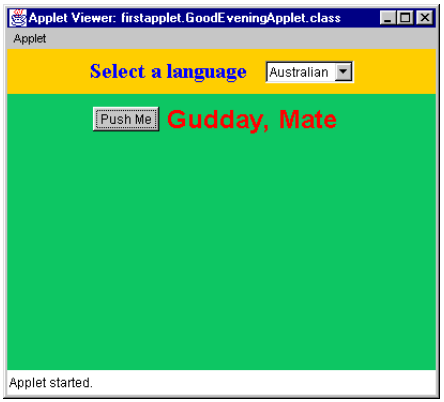
5 굵은 자체로 표시된 다음 코드를 입력합니다.

```
void button1_actionPerformed(ActionEvent e) {
  label6.setForeground(new Color(255,0,0));
}
```

8 프로젝트를 저장합니다.

9 애플릿을 실행하고 드롭다운 리스트에서 "Australian"을 선택합니다 ."Push Me" 버튼을 클릭합니다. "Gudday, Mate"가 빨간색이 됩니다.

애플릿은 다음과 같이 나타납니다.



10 애플릿을 종료합니다.

7 단계: 애플릿 배포

Java 애플릿 배포는 애플릿에 필요한 여러 가지 Java 클래스 파일, 이미지 파일 및 그 밖의 파일을 함께 묶어서 이러한 파일과 애플릿 HTML 파일이 실행될 수 있는 서버나 클라이언트 컴퓨터의 특정 위치로 복사하는 과정으로 구성됩니다.파일을 별도로 배포하거나 압축된 아카이브 파일 또는 압축되지 않은 아카이브 파일로 배포할 수 있습니다. Java 아카이브 파일인 JAR 파일이 가장 일반적으로 사용됩니다. JAR 파일은 보다 작은 파일 크기 및 보다 빠른 다운로드 시간의 이점을 제공합니다.

애플릿 배포 시 다음 사항을 기억해야 합니다.

- 기존 디렉토리 구조를 유지합니다. 다음 예제에서 GoodEveningApplet.class는 패키지 구조를 반영하도록 firstapplet 디렉토리에 있어야 합니다.firstapplet/GoodEveningApplet.class. JAR 파일을 배포하려는 경우 파일의 디렉토리 구조를 확인하고 일치하는지 확인합니다.
- 필요한 모든 클래스를 전달합니다. 클래스 파일은 HTML 파일에 상대적인 적절한 위치에 있어야 하고 codebase 속성과 일치해야 합니다.
- 애플릿 HTML 파일을 전달합니다.

주의 이전 JDK 1.02 호환 브라우저용 애플릿을 만드는 경우 이전 브라우저에서는 JAR 파일을 지원하지 않는다는 사실을 기억해 두십시오. 그 대신 ZIP 아카이브 파일을 만드십시오.

참조 사항
- *JBuilder를 이용한 애플리케이션 구축*의 "Java 프로그램 배포"
- JBuilder 자습서 "Java 텍스트 에디터 구축 - 텍스트 에디터 애플리케이션 배포"의 단계 16

JBuilder의 에디션에 따라 애플릿 배포를 위한 다음과 같은 여러 도구가 있습니다.

- JDK에 사용할 수 있는 Java **jar** 툴
- 전문가용 및 기업용에서 사용할 수 있는 JBuilder의 Archive Builder

중요 JAR 파일을 만들기 전에 다음의 검사 목록을 검토합니다.

- 배포하기 전에 프로젝트를 저장하고 컴파일합니다.
- GoodEveningApplet.html의 code 속성에 패키지 이름을 포함하는 전체 클래스 이름 firstapplet.GoodEveningApplet이 있는지 확인합니다.
- GoodEveningApplet.html의 codebase 속성이 HTML 파일에 상대적인 클래스 파일의 위치를 올바르게 지정하는지 확인합니다. 다음 예제에서는 클래스 파일을 포함하는 JAR 파일이 HTML 파일과 동일한 디렉토리에 있으므로 codebase는 "."입니다.

jar 툴을 이용한 애플릿 배포

JBuilder 개인용

JDK에서는 bin 디렉토리에 있는 **jar** 툴을 사용하여 배포용 JAR 파일을 만듭니다. 아카이브 및 압축 도구인 **jar** 툴은 여러 파일을 단일 JAR 아카이브로 결합합니다.

기본 **jar** 명령은 다음과 같은 형식을 갖습니다.

```
jar {ctxu}[vfm0M] [jar-file] [manifest-file] [-C dir] files ...
```

참고 추가 JAR 옵션에 대한 도움말을 보려면 명령 줄에서 jar -help를 입력하거나 http://java.sun.com/j2se/1.3/docs/tooldocs/tools.html#basic의 **jar** 설명을 참조하십시오.

다음 단계로 JAR 파일을 만듭니다.

1. 프로젝트를 저장하고 컴파일합니다.
2. /<home>/jbproject/ 디렉토리에 애플릿을 위한 applets 디렉토리를 만듭니다.이 디렉토리는 애플릿 HTML 파일과 JAR 파일을 포함시키는 테스트 디렉토리가 됩니다.
3. 명령줄 셸이나 DOS 창을 엽니다.
4. 프로젝트 디렉토리인 /<home>/jbproject/FirstApplet/으로 이동합니다.
5. 다음과 같이 JAR 명령을 입력합니다.

   ```
   jar cvf GoodEvening.jar -C classes firstapplet
   ```

   여기에서

jar	-JAR 파일을 생성하는 명령입니다.
c	-새 보관을 만듭니다.
v	-JAR 파일에 추가된 사항을 지정하는 출력을 생성합니다.
f	-생성할 JAR 파일의 이름을 지정합니다.
-C	-classes 디렉토리의 경우 jar 명령을 실행하는 동안 지정한 디렉토리로 임시 변경합니다.
클래스	-jar 명령이 실행되는 디렉토리입니다.
firstapplet	-JAR 파일에 추가될 클래스의 패키지입니다.

참고 JDK는 경로에 있어야 합니다. 그렇지 않은 경우 다음 명령을 사용하십시오.

```
<jbuilder>/jdk1.3/bin/jar cvf GoodEvening.jar -C classes firstapplet
```

여기서 <jbuilder>는 예를 들어 jbuilder5/처럼 실행 중인 JBuilder의 버전을 나타냅니다. JBuilder가 다른 드라이브에 있으면 드라이브 이름을 포함합니다.

참고 Windows에서는 백슬래시(₩)를 사용합니다.

6. JAR 파일은 프로젝트 FirstApplet 디렉토리에 생성됩니다. WinZip과 같은 JAR 유틸리티로 JAR 파일을 열고 클래스의 디렉토리 구조가 올바른지 확인합니다. 예를 들어, firstapplet/GoodEveningApplet.class와 같은 형식입니다. 또한 **jar** 툴이 목록 파일을 생성했는지 기억해 두십시오. 기록 파일은 아카이브에 대한 모든 메타 정보가 저장되는 파일입니다.
7. GoodEvening.jar를 applets 디렉토리에 복사하여 테스트합니다.

JAR 파일을 생성한 후 24- 27페이지 "8 단계: HTML 파일 수정"을 계속합니다.

참조 사항

- http://java.sun.com/docs/books/tutorial/jar/basics/index.html의 "Using JAR Files: The Basics"
- http://java.sun.com/j2se/1.3/docs/tooldocs/tools.html#basic의 "**jar**–The Jave Archive Tool"
- *Building Applications with JBuilder*의 "Java 프로그램 배포"

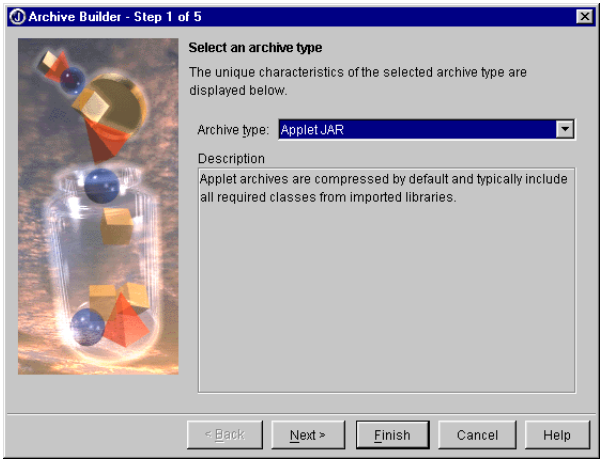
Archive Builder 를 이용한 애플릿 배포

이것은 JBuilder 전문가용 및 기업용 버전의 기능입니다.

JBuilder의 Archive Builder는 애플릿 배포에 필요한 모든 파일을 모으고 JAR 파일로 보관할 수 있습니다.

JBuilder 전문가용 및 기업용을 이용하여 애플릿을 배포하려면 다음과 같이 합니다.

1. 프로젝트를 저장하고 컴파일합니다.
2. /<home>/jbproject/ 디렉토리에 애플릿을 위한 applets 디렉토리를 만듭니다.이 디렉토리는 애플릿 HTML 파일과 JAR 파일을 포함시키는 테스트 디렉토리가 됩니다.
3. Wizards|Archive Builder를 선택하여 Archive Builder를 엽니다.

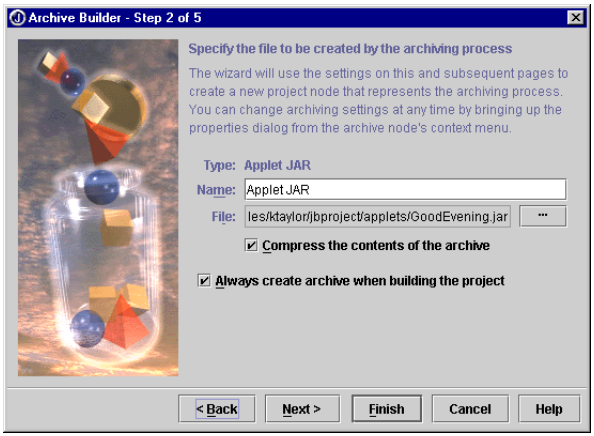


4. 1 단계의 Archive Type 드롭다운 리스트에서 Applet JAR를 선택합니다. Next를 클릭하여 2 단계로 이동합니다.

참고 JAR 파일을 지원하지 않는 이전 브라우저용으로 애플릿을 배포하는 경우 Applet ZIP 아카이브 타입을 선택합니다.

5. 이름 필드에 Applet JAR의 기본 이름을 사용합니다.
6. File 필드 옆에 있는 생략 버튼을 클릭하고 /<home>/jbproject/applets/ 디렉토리를 찾아 이동합니다.

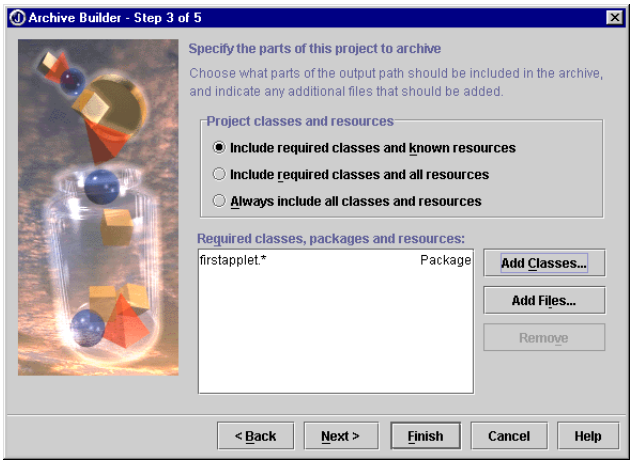
7 JAR 파일 이름을 GoodEvening.jar로 변경하고 OK를 선택합니다.



8 3 단계로 이동하려면 Next를 선택합니다.

9 Include Required Classes And Known Resources를 선택합니다. 그런 다음 추가할 클래스를 수동으로 선택합니다. 이 옵션을 선택하면 클래스와 함께 배포할 필요가 없는 HTML 파일을 제외할 수 있습니다.

10 Add Classes 버튼을 클릭하고 firstapplet 패키지를 선택합니다. 패키지는 Required Classes, Packages And Resources 목록에 표시됩니다.



11 4 단계로 가려면 Next를 클릭합니다.

12 4 단계와 5 단계에서 기본 설정을 사용합니다. 5 단계에서는 아카이브에 대한 목록 파일을 생성하도록 옵션을 설정합니다.

참고 목록 파일에 대한 내용은 *Building Applications with JBuilder*의 "Java 프로그램 배포"에서 "목록 파일"을 참조하십시오.

13 Finish를 클릭하여 Archive Builder를 종료합니다. Applet JAR이라는 아카이브 노드가 프로젝트 창에 나타납니다.마우스 오른쪽 버튼을 클릭하고 Properties를 선택하여 이 파일을 수정할 수 있습니다.

14 Project|Make Project를 선택하여 프로젝트를 컴파일합니다. Archive Builder는 마법사의 3 단계에서 추가한 firstapplet 패키지를 JAR 파일에 모읍니다.

15 Applet JAR 아카이브 노드 옆에 있는 확장 아이콘을 클릭하여 GoodEvening.jar 아카이브 파일을 봅니다.프로젝트 창에서 JAR 파일을 더블 클릭합니다. 목록 파일은 컨텐트 창에 나타나고 JAR 파일의 컨텐트는 구조 창에 나타납니다. 구조 창에서 파일을 선택하여 컨텐트 창에서 봅니다.

참고 여러 프로그램을 같은 위치에 전달하기 위해 JAR 파일의 각각에 포함하지 않고 재배포 가능한 파일을 개별적으로 전달할 수 있습니다.

참조 사항
- *Building Applications with JBuilder*의 "Archive Builder 사용"
- *Building Applications with JBuilder*의 "Java 프로그램 배포"

8 단계: HTML 파일 수정

애플릿이 JAR 파일로 배포되었으므로 archive 속성을 사용하여 HTML 파일을 수정하고 JAR 파일 이름을 포함시켜야 합니다. 브라우저에서 Java를 지원하지 않거나 브라우저를 업그레이드하지 않으면 애플릿을 볼 수 없음을 Java 지원 브라우저가 없는 사용자에게 알려 주는 메시지를 <applet> 태그 내에 추가할 수 있습니다.

다음과 같은 방법으로 HTML 파일을 수정합니다.

1 JBuilder에서 GoodEveningApplet.html을 열고 다음과 같이 archive 속성을 추가합니다.

   1 Source 탭을 선택하여 HTML 소스 코드를 봅니다.

   2 다음과 같은 HTML 코드를 <applet> 태그 내에 추가합니다.

```
archive  = "GoodEvening.jar"
```

<applet> 태그가 다음과 같이 나타납니다.

```
<applet
  codebase = "."
  code     = "firstapplet.GoodEveningApplet.class"
archive  = "GoodEvening.jar"
  name     = "TestApplet"
  width    = 400
  height   = 300
  hspace   = 0
  vspace   = 0
  align    = top
>
</applet>
```

팁 애플릿에 여러 개의 JAR 파일이 있으면 다음에 표시된 대로 쉼표로 구분하여 목록을 만듭니다.

```
archive="file1.jar, file2.jar"
```

중요 이전 브라우저는 JAR 파일과 아카이브 파일의 여러 목록을 지원하지 않지만 archive 속성에서 단일 ZIP 파일은 지원합니다.

다음으로 Java를 지원하는 브라우저가 없는 사용자에게 브라우저에서 Java를 지원하지 않으므로 애플릿을 볼 수 없다는 것을 알려 줍니다.

2 열고 닫는 <applet> 태그 사이에 다음과 같은 메시지를 입력합니다.

```
이 애플릿을 보려면 JDK 1.1.x 이상을 실행하는 Java 를 지원하는 브라우저가 필요합니다 .
```

<applet> 태그는 다음과 같이 나타납니다.

```
<applet
  codebase = "."
  code     = "firstapplet.GoodEveningApplet.class"
archive  = "GoodEvening.jar"
  name     = "TestApplet"
  width    = 400
  height   = 300
  hspace   = 0
  vspace   = 0
  align    = top
>
이 애플릿을 보려면 JDK 1.1.x 이상을 실행하는 Java 를 지원하는 브라우저가
필요합니다 .
</applet>
```

Java를 지원하지 않는 모든 브라우저는 <applet> 태그를 무시하고 태그 사이에 있는 모든 내용을 표시합니다. Java를 지원하는 브라우저에서는 <applet> 태그를 인식하기 때문에 Java를 지원하는 브라우저를 사용하는 모든 사람은 메시지가 아닌 애플릿을 봅니다.

중요 HTML 파일을 저장하기 전에 codebase 및 code 값을 다시 확인합니다. 이 값이 잘못되면 애플릿이 실행되지 않습니다. codebase 값은 HTML 파일에 상대적인 애플릿 코드(클래스 또는 JAR 파일)의 위치입니다. "." 값은 클래스 파일이 HTML 파일과 동일한 디렉토리에 있다는 의미입니다. code 값은 패키지 이름을 포함하는 애플릿의 전체 클래스 이름이어야 합니다.

3 파일을 저장하고 닫습니다.

4 수정된 GoodEveningApplet.html을 프로젝트의 classes 디렉토리에서 applets 디렉토리로 복사합니다. applets 디렉토리는 두 파일, GoodEveningApplet.html과 GoodEvening.jar을 포함해야 합니다.

주의 JBuilder는 두 가지 파일 즉 프로젝트 디렉토리의 루트에 있는 프로젝트 노트 HTML 파일인 FirstApplet.html과 프로젝트 src 디렉토리에 있는 <applet> 태그를 포함하는 애플릿 HTML 파일인 GoodEveningApplet.html을 생성합니다. GoodEveningApplet.html 대

신 FirstApplet.html을 applets 디렉토리로 복사하지 마십시오. 그러면 애플릿이 실행되지 않습니다.

9 단계: 배포된 애플릿을 명령 줄에서 실행

배포된 애플릿은 웹에서 테스트하기 전에 로컬로 테스트하는 것이 좋습니다. Sun의 **appletviewer**를 사용하여 명령 줄에서 테스트할 수 있습니다. 테스트를 수행하면 애플릿을 실행하는 데 필요한 모든 요소가 브라우저에 있는지 알려 줍니다. 파일이 없거나 HTML 파일에서 오류가 발생하면 애플릿은 실행되지 않습니다. 그러면 웹에 게시하기 전에 오류를 수정할 수 있습니다.

다음과 같은 방법으로 명령 줄에서 애플릿을 실행합니다.

1. GoodEveningApplet.html과 GoodEvening.jar의 복사본이 applets 디렉토리에 있는지 확인합니다.
2. 명령줄 창을 엽니다.
3. 다음과 같이 모든 CLASSPATH 변수를 지워 이 세션의 클래스 경로 설정을 제거합니다.
   - Windows 95, 98, NT 및 2000:set CLASSPATH=
   - UNIX:
     - csh 셸의 경우 :unsetenv CLASSPATH
     - sh 셸의 경우 :unset CLASSPATH
4. applets/ 디렉토리로 이동합니다.
5. 다음 명령을 입력하여 **appletviewer**를 실행합니다.

   <jbuilder>/jdk1.3/bin/appletviewer GoodEveningApplet.html

   여기에서 <jbuilder>는 예를 들어 jbuilder5/처럼 실행 중인 JBuilder의 버전을 나타냅니다.

중요 JBuilder가 다른 드라이브에 있으면 드라이브 문자를 포함시킵니다.

참고 Windows에서는 역슬래시(₩)를 사용합니다.

6. "Good Evening" 애플릿이 **appletviewer**에 로드되고 실행되면 배포는 성공적이고 모든 클래스를 찾아 포함한 경우입니다. 애플릿이 실행되지 않으면 오류 메시지를 확인하고 수정한 다음 다시 컴파일하고 배포하여 다시 테스트합니다.

"Good Evening" 애플릿은 JBuilder 설치 디렉토리의 samples/Tutorials/FirstApplet/에 있는 예제로 사용할 수 있습니다.

애플릿을 실행하는 데 문제가 있으면 24- 31페이지 "애플릿 소스 코드"를 확인하고 일반 오류에 대한 다음과 같은 항목을 참조하십시오.

- http://www.java.sun.com/docs/books/tutorial/applet/problems/index.html의 "Solving common applet problems"

- *웹 애플리케이션 개발자 안내서*의 "애플릿 작업" 장에서 "<applet> 태그를 사용할 때의 일반적인 실수" 및 "애플릿 작업을 위한 추가 팁"

10 단계: 배포된 애플릿을 웹 상에서 테스트

이 단계에서는 배포된 애플릿을 웹에서 테스트하는 것에 대한 포괄적인 개요만 제공하고 자세한 내용은 다루지 않습니다. 필요한 파일이 아카이브에 포함되어 있는지 확인하려면 다양한 브라우저에서 애플릿을 테스트하는 것이 중요합니다.

애플릿을 테스트하는 마지막 단계는 웹에서 실행하는 것입니다. 웹에서 실행하면 필요한 모든 파일이 실제로 있는지를 알 수 있습니다.

이러한 단계를 완료한 다음 웹에서 애플릿을 테스트합니다.

1. 애플릿의 HTML 파일과 JAR 파일을 인터넷 서버로 전송하거나 Windows NT 서버로 복사합니다.
   1. FTP(file transfer protocol) 유틸리티를 사용하여 파일을 서버로 전송합니다. 파일은 바이너리 파일로 전송해야 합니다.
   2. 서버에서 HTML 파일과 JAR 파일의 위치가 HTML 파일의 codebase 속성과 일치하고 code 속성에 패키지 이름을 포함하는 전체 클래스 이름이 있는지 확인합니다.
2. 다양한 브라우저에서 애플릿을 테스트합니다. 애플릿을 로드하지 못하면 브라우저에서 Java를 지원하는지 확인합니다. 또한 브라우저의 Java Console에서 오류 메시지를 확인합니다.

   다음과 같은 방법으로 Java Console을 엽니다.

   - Netscape의 경우 Communicator|Tools|Java Console을 선택합니다.
   - Internet Explorer의 경우 View|Java Console을 선택합니다.
3. 오류를 수정하고 애플릿을 재배포한 다음 브라우저에서 다시 테스트합니다.

축하합니다! JBuilder를 이용하여 처음으로 애플릿을 만들었습니다.이제 JBuilder의 개발 환경에 익숙해졌으므로 JBuilder의 여러 가지 시간 절약(time-saving) 기능을 통해 프로그램을 더 쉽게 만들 수 있을 것입니다.

본 자습서의 개선에 관한 추가적인 제안 사항은 전자 우편을 통해 jpgpubs@borland.com으로 보내 주십시오.

다른 애플릿 자습서에 대해서는 다음을 참조하십시오.

- http://java.sun.com/docs/books/tutorial/index.html의 "Java™ Tutorial"
- http://homepages.borland.com/ccalvert/JavaCourse/index.htm의 Charlie Calvert's Part II: Applet

- http://formlessvoid.com/jc/applets/의 Rich Wilkman's Curmudgeon 웹 사이트
- http://www.microps.com/mps/p_appletdesign.html의 John Moore's "Applet design and deployment"

애플릿 소스 코드

애플릿 HTML 소스 코드

GoodEveningApplet.html의 소스 코드

```
<html>
<head>
<meta http="Content-Type" content="text/html; charset=windows-1252">
<title>
Good Evening HTML Page
</title>
</head>
<body>
Java 를 지원하는 브라우저의 하단에 firstapplet.GoodEveningApplet 이 나타납
니다 .


<applet
  codebase = "."
  code     = "firstapplet.GoodEveningApplet.class"
archive  = "GoodEvening.jar"
  name     = "TestApplet"
  width    = 400
  height   = 300
  hspace   = 0
  vspace   = 0
  align    = top
>
이 애플릿을 보려면 JDK 1.1.x 이상을 실행하는 Java 를 지원하는 브라우저가 필
요합니다 .
</applet>
</body>
</html>
```

애플릿 클래스 소스 코드

GoodEveningApplet.java의 소스 코드

```
package firstapplet;

import java.awt.*;
import java.awt.event.*;
import java.applet.*;
```

```
public class GoodEveningApplet extends Applet {
  boolean isStandalone = false;
  BorderLayout borderLayout1 = new BorderLayout();
  Panel lower = new Panel();
  Panel upper = new Panel();
  CardLayout cardLayout1 = new CardLayout();
  Panel panel1 = new Panel();
  Panel panel2 = new Panel();
  Panel panel3 = new Panel();
  Panel panel4 = new Panel();
  Panel panel5 = new Panel();
  BorderLayout borderLayout2 = new BorderLayout();
  BorderLayout borderLayout3 = new BorderLayout();
  BorderLayout borderLayout4 = new BorderLayout();
  BorderLayout borderLayout5 = new BorderLayout();
  Choice choice1 = new Choice();
  Label label1 = new Label();
  Label label2 = new Label();
  Label label3 = new Label();
  Label label4 = new Label();
  Label label5 = new Label();
  Label label6 = new Label();
  FlowLayout flowLayout1 = new FlowLayout();
  Button button1 = new Button();
  /**Get a parameter value*/
  public String getParameter(String key, String def) {
    return isStandalone ? System.getProperty(key, def) :
      (getParameter(key) != null ? getParameter(key) : def);
  }

  /**Construct the applet*/
  public GoodEveningApplet() {
  }
  /**Initialize the applet*/
 public void init() {
   choice1.addItem("English");
   choice1.addItem("German");
   choice1.addItem("Pig Latin");
   choice1.addItem("Swedish");
    choice1.addItem("Australian");

  try {
    jbInit();
    }
    catch(Exception e) {
      e.printStackTrace();
    }
  }
  /**Component initialization*/
  private void jbInit() throws Exception {
    this.setLayout(borderLayout1);
    upper.setBackground(Color.orange);
    lower.setBackground(Color.magenta);
```

```
lower.setLayout(cardLayout1);
panel1.setLayout(borderLayout2);
panel2.setLayout(borderLayout3);
panel3.setLayout(borderLayout4);
panel4.setLayout(borderLayout5);
panel5.setLayout(flowLayout1);
panel1.setBackground(new Color(190, 173, 255));
panel2.setBackground(new Color(83, 182, 255));
panel3.setBackground(new Color(255, 149, 66));

panel4.setBackground(new Color(239, 107, 140));
panel5.setBackground(new Color(17, 198, 99));
label1.setFont(new java.awt.Font("Serif", 1, 20));
label1.setForeground(Color.blue);
label1.setText("Select a language");
label2.setFont(new java.awt.Font("Dialog", 1, 24));
label2.setForeground(Color.black);
label2.setText("Good Evening");
label3.setFont(new java.awt.Font("Dialog", 1, 24));
label3.setForeground(Color.black);
label3.setText("Guten Abend");
label4.setFont(new java.awt.Font("Dialog", 1, 24));
label4.setForeground(Color.black);
label4.setText("Oodgay vening eay");
label5.setFont(new java.awt.Font("Dialog", 1, 24));
label5.setForeground(Color.black);
label5.setText("God Kvall");
choice1.addItemListener(new java.awt.event.ItemListener() {
  public void itemStateChanged(ItemEvent e) {
    choice1_itemStateChanged(e);
  }
});
label6.setFont(new java.awt.Font("Dialog", 1, 24));
label6.setForeground(Color.black);
label6.setText("Gudday, Mate");
button1.setLabel("Push Me");
button1.addActionListener(new java.awt.event.ActionListener() {
  public void actionPerformed(ActionEvent e) {
    button1_actionPerformed(e);
  }
});
this.add(lower, BorderLayout.CENTER);
lower.add(panel1, "panel1");
panel1.add(label2, BorderLayout.NORTH);
lower.add(panel2, "panel2");
panel2.add(label3, BorderLayout.SOUTH);
lower.add(panel3, "panel3");
panel3.add(label4, BorderLayout.EAST);
lower.add(panel4, "panel4");
panel4.add(label5, BorderLayout.WEST);
lower.add(panel5, "panel5");
panel5.add(button1, null);
panel5.add(label6, null);
this.add(upper, BorderLayout.NORTH);
```

```
    upper.add(label1, null);
    upper.add(choice1, null);
  }
  /**Start the applet*/
  public void start() {
  }
  /**Stop the applet*/
  public void stop() {
  }

  /**Destroy the applet*/
  public void destroy() {
  }
  /**Get Applet information*/
  public String getAppletInfo() {
    return "Applet Information";
  }
  /**Get parameter info*/
  public String[][] getParameterInfo() {
  return null;
  }

void choice1_itemStateChanged(ItemEvent e) {
  if ("English".equals(choice1.getSelectedItem())){
    cardLayout1.show(lower, "panel1");
    }
  else if ("German".equals(choice1.getSelectedItem())){
    cardLayout1.show(lower, "panel2");
    }
  else if ("Pig Latin".equals(choice1.getSelectedItem())){
    cardLayout1.show(lower, "panel3");
    }
  else if ("Swedish".equals(choice1.getSelectedItem())){
    cardLayout1.show(lower, "panel4");
    }
  else if ("Australian".equals(choice1.getSelectedItem())){
    cardLayout1.show(lower, "panel5");
    }
  }

  void button1_actionPerformed(ActionEvent e) {
  label6.setForeground(new Color(255,0,0));
  }
}
```