IBM VisualAge® for Java™

시작하기 전에

버전 3.0

IBM VisualAge® for Java™

시작하기 전에

버전 3.0

주!

이 책과 이 책이 지원하는 제품을 사용하기 전에, 반드시 vii 페이지의 "주의사항"에 있는 일반 정보를 읽으십시오.

개정판

이 개정판은 새로운 개정판에서 특별히 언급하지 않는 한 IBM VisualAge for Java 릴리스 3.0 및 후속 릴리스 및 수정판에 적용됩니다.

목차

주의사항

이 책은 미국에서 제공되는 제품 및 서비스에 맞게 개발되었습니다. IBM은 이 책에서 명시한 제품, 서비스 또는 기능을 다른 국가에서 제공하지 않을 수 있습니다. 사용자의 지역에서 현재 사용할 수 있는 제품 및 서비스에 대해서는 해당 IBM 영업대표에게 문의하십시오. 이 책에서 IBM 제품, 프로그램 또는 서비스를 참조하고 있다고 해서 오로지 IBM 제품, 프로그램, 서비스만을 사용해야 한다는 의미는 아닙니다. IBM의 지적 소유권을 침해하지 않는 기능상으로 동등한 제품, 프로그램 또는 서비스를 대신 사용할 수 있습니다. 그러나 비 IBM 제품, 프로그램 또는 서비스와 관련된 작동의 평가 및 검증은 사용자의 책임입니다.

IBM은 이 책의 주제를 다루는 응용프로그램을 보유하거나 출원 중일 수 있습니다. 이 책을 제공한다고 해서 그 특허에 대한 사용권을 부여하는 것은 아닙니다. 다음 주소로 우편을 보내면 사용권을 조회하실 수 있습니다.

150-510
서울특별시 영등포구 여의도동 25-11, 한진빌딩
한국 아이.비.엠 주식회사
지적재산권부

2 바이트(DBCS) 정보와 관련된 사용권을 조회하려면 해당 국가의 IBM 지적재산권부로 문의하거나 다음으로 조회 문의서를 보내십시오.

IBM World Trade Asia Corporation
Licensing
2-31 Roppongi 3-chome, Minato-ku
Tokyo 106, Japan

다음 문단은 영국 또는 다음 규정이 국내법과 모순되는 국가에는 적용되지 않습니다. IBM은 특정 목적에의 적합성 또는 판매 가능성에 대한 암시적 보증을 포함하여 어떤 종류의 명시적 또는 묵시적 보증 없이 이 책을 '현상대로' 제공합니다. 일부 국가에서는 특정 거래에서의 명시적인 또는 암시적인 보증의 거부를 허용하지 않으므로 이 문장이 사용자에게 적용되지 않을 수 있습니다.

이 책에는 기술상의 부정확성이나 인쇄상의 오류가 있을 수 있습니다. 이 책의 정보는 정기적으로 변경되며, 변경사항은 개정판에 통합됩니다. IBM은 언제라도 이 책에서 설명하는 프로그램 및 제품을 통지없이 변경하거나 개선할 수 있습니다.

이 책에서 언급한 IBM 이외의 웹사이트는 오직 편의를 위한 것이므로 어떤 방식으로든 이 웹사이트를 보증하지 않습니다. 그리고 이 웹사이트의 데이터는 IBM 제품용 데이터가 아니므로 이 웹사이트를 사용하는 것은 사용자의 책임입니다.

IBM은 독자가 제공한 정보가 타당한 경우 적절한 방식으로 이를 사용하거나 배포할 수 있으며, 제공한 독자는 이에 대해 책임을 지지 않습니다.

(i) 개별적으로 작성된 프로그램과 기타 프로그램(이 프로그램 포함) 간의 정보 교환 (ii) 교환된 정보의 상호 사용을 목적으로 이 프로그램에 대한 정보가 필요한 프로그램 사용권자는 다음으로 문의하십시오.

151-010
서울특별시 영등포구 여의도동 25-11 한진빌딩
한국 아이.비.엠 주식회사
소프트웨어 사업본부(02-781-7777)

이런 정보는 적절한 조항과 조건에 따라 사용가능하며 경우에 따라서는 비용을 지불해야 합니다.

이 책에서 언급한 사용권 프로그램 및 이에 사용가능한 모든 사용권 자료는 IBM에서 IBM 고객 계약, 국제 프로그래밍 사용권 계약 또는 그에 해당하는 고객과의 계약에 따라 제공한 것입니다.

타사 제품과 관련된 정보는 해당 제품의 공급자, 공개 발표 또는 기타 공개적으로 사용가능한 소스에서 확보한 것입니다. IBM은 해당 제품을 검사하지 않았으므로 성능의 정확성, 호환성 또는 타사 제품과 관련된 기타 주장을 확인할 수 없습니다. 타사 제품의 성능에 관한 문제는 해당 제품의 공급자에게 제기해야 합니다.

이 책에는 일상적인 업무에 사용되는 자료와 보고서 예제가 있습니다. 가능한 완벽하게 설명하기 위해 예제에는 개인, 회사, 상호 및 상품명이 포함됩니다. 이러한 이름들은 모두가 허구이며 실제 비지니스 기업에서 사용하는 이름 및 주소와 유사하다면 이는 우연입니다.

저작권:

이 책에는 다양한 운영 플랫폼에서의 프로그래밍 기술을 나타내는 예제 응용프로그램이 소스 언어로 있습니다. 사용자는 예제 프로그램이 쓰여진 운영 플랫폼의 응용프로그램 프로그래밍 인터페이스를 따르는 응용프로그램의 개발, 사용, 마케팅 또는 분배의 목적으로, IBM에 비용을 지불하지 않고 예제 프로그램을 복사하고, 수정하고, 분배할 수 있습니다. 이런 예제는 모든 조건에서 철저히 검사하지는 않았습니다. 따라서 IBM은 이런 프로그램의 신뢰성, 유용성 또는 기능을 보증할 수 없습니다. 사용자는 IBM의 프로그래밍 인터페이스를 따르는 응용프로그램의 개발, 사용, 마케팅 또는 분배의 목적으로, IBM에 비용을 지불하지 않고 예제 프로그램을 복사하고, 수정하고, 분배할 수 있습니다.

이런 샘플 프로그램의 사본이나 일부 또는 파생 작업에는 다음과 같은 저작권 정보가 있어야 합니다.

(회사명) (연도). Portions of this code are derived from IBM Corp. Sample Programs. Copyright IBM Corp. 1997, 1999. All rights reserved.

프로그래밍 인터페이스 정보

프로그래밍 인터페이스 정보의 목적은 이 프로그램을 사용하여 응용프로그램 소프트웨어 작성을 돕는 데 있습니다.

일반 프로그래밍 인터페이스를 통해 사용자는 이 프로그램 도구의 서비스를 확보하는 응용프로그램 소프트웨어를 작성할 수 있습니다.

그런데 이 정보에는 진단, 수정 및 조정 정보도 있을 수 있습니다. 진단, 수정 및 조정 정보는 응용프로그램 소프트웨어 디버그를 돕기 위해 제공됩니다.

경고: 이 진단, 수정 및 조정 정보는 변하기 쉬우므로 이런 정보를 프로그래밍 인터페이스로 사용하지 마십시오.

등록 상표 및 서비스 마크

다음 용어는 미국 및 기타 국가에서 사용되는 IBM사의 등록상표입니다.

AIX
AS/400
DB2@@
CICS
LG-IBM
Network Station
OS/2
OS/390
OS/400
Operating System/400
S/390
VisualAge
WebSphere

Lotus, Lotus Notes 및 Domino는 미국 및 기타 국가에서 사용되는 Lotus Development사의 독점 등록된 등록상표입니다.

Tivoli Management Environment, TME 10 및 Tivoli Module Designer는 미국 및 기타 국가에서 사용되는 Tivoli Systems사의 등록상표입니다.

Encina 및 DCE Encina Lightweight Client는 미국 및 기타 국가에서 사용되는 Transarc사의 등록상표입니다.

Microsoft, Windows, Windows NT 및 Windows 로고는 미국 및 기타 국가에서 사용되는 Microsoft사의 등록상표 또는 독점 등록된 등록상표입니다.

Java 및 모든 Java 관련 등록상표는 미국 및 기타 국가에서 사용되는 Sun Microsystems사의 등록상표 또는 독점 등록된 등록상표입니다.

UNIX는 미국 및 기타 국가에서 X/Open Company Limited가 독점적으로 사용권을 체결한 등록상표입니다.

ActionMedia, LANDesk, MMX, Pentium 및 ProShare는 미국 및 기타 국가에서 사용되는 Intel사의 등록상표 또는 독점 등록된 등록상표입니다.

기타 이중 별표(**)로 표시되는 회사, 상품 및 서비스명은 해당 회사의 등록상표 또는 서비스 상표입니다.

제1장 소개

이 문서의 목적은 다음을 소개하는 데 있습니다.

- VisualAge for Java 사용시 필요한 기본 개념 및 항목
- VisualAge for Java를 사용하여 응용프로그램 작성시 알아야 하는 기본 내용

이런 목적을 쉽게 달성하기 위해 여기에서는 간단한 Java 애플릿을 작성해 봅니다. 그런 다음 이 애플릿에 기능을 추가해 봅니다.

이 책의 내용

이 책은 다음 절로 구성되어 있습니다.

- 5 페이지의 『제2장 이 릴리스의 새로운 내용』에서는 마지막 릴리스 이후 VisualAge for Java의 변경된 내용을 간략하게 소개합니다.
- 7 페이지의 『제3장 VisualAge for Java란 무엇인가?』에서는 VisualAge for Java의 전반적인 기능을 소개하고 알아 두어야 할 개념을 설명합니다.
- 15 페이지의 『제4장 첫번째 애플릿 빌드 소개』에서는 간단한 애플릿 작성 프로세스를 통해 VisualAge for Java의 비주얼 프로그래밍 기능을 소개합니다.
- 37 페이지의 『제5장 애플릿 상태 점검 추가 소개』에서는 VisualAge for Java의 비주얼 프로그래밍 기능에 대해 자세하게 설명하고 앞에서 작성한 애플릿을 개선하는 방법을 보여 줍니다. 또한 이 절에서는 코드 관리에 대한 VisualAge for Java의 접근 방법을 소개합니다.
- 47 페이지의 『제6장 To-Do List 프로그램 향상 소개』에서는 작성한 애플릿에 기능을 추가하는 방법을 보여 주고 VisualAge for Java의 전반적인 코딩 환경에 대해 자세하게 설명합니다.
- 63 페이지의 『제7장 비주얼 컴포지션 편집기에서 할 수 있는 기타 작업 소개』에서는 VisualAge for Java의 강력한 비주얼 프로그래밍 기능을 자세하게 설명합니다.
- 91 페이지의 『제8장 개정판 관리 소개』에서는 VisualAge for Java의 개정판 제어 기능에 대해 자세하게 설명합니다.

- 105 페이지의 『제9장 IDE에서 할 수 있는 기타 작업』에서는 조정과 디버깅같은 VisualAge for Java의 다른 기능에 대한 내용을 설명합니다.
- 147 페이지의 『제10장 관계형 데이터 작업 소개』에서는 데이터 액세스 빈에서 작업하는 방법을 보여 줍니다.
- 163 페이지의 『제11장 Domino AgentRunner 사용』에서는 VisualAge for Java에서 Domino 에이전트를 빌드, 실행 및 디버깅하는 과정을 설명합니다.
- 169 페이지의 『제12장 JSP/서블릿 개발 환경』에서는 VisualAge for Java에서 JavaServer 페이지를 지원하는 기능을 소개합니다.

5 페이지의 『VisualAge for Java에 대한 자세한 정보』에서는 VisualAge for Java와 같이 제공되는 온라인 도움말을 설명합니다. 또한 이 절에서는 온라인 도움말에서 자료를 인쇄하는 방법을 자세하게 설명합니다.

이 책의 샘플 프로그램

15 페이지의 『제4장 첫번째 애플릿 빌드 소개』에서 시작하는 이 문서의 예제를 수행하여 *To-Do List* 샘플 프로그램을 작성할 수 있습니다. 이 예제의 완성된 버전은 VisualAge for Java와 같이 제공되는 IBM Java Examples 프로젝트의 com.ibm.ivj.examples.vc.todolist입니다. 이 예제의 완성된 버전을 검토하는 방법에 대한 자세한 설명은 103 페이지의 『저장소에서 예제 점검』을 참조하십시오.

150 페이지의 『실제에서의 데이터 액세스빈 개요』를 시작하면 *StaffList*라 불리는 간단한 조회 프로그램을 빌드하기 위해 DB2 Universal Database에 들어 있는 간단한 데이터베이스를 사용합니다.

이 책의 대상

이 책의 대상은 VisualAge for Java의 기본 사용에 익숙한 프로그램 작성자와 제품에 대해 이해하려는 사람입니다. 이 책에서는 VisualAge for Java를 사용한 프로그램 빌드 이면의 기본 개념을 소개하고, VisualAge for Java의 일반 비주얼 프로그래밍 프로세스를 설명하고, 샘플 프로그램을 작성해 봅니다. 이 책을 최대로 활용하려면 Java 언어의 기본 내용에 익숙해야 합니다.

이 제품에 관하여

VisualAge for Java는 Java 응용프로그램과 애플릿 작성을 위한 완전히 통합된 환경입니다.

VisualAge for Java는 대화식 비주얼 프로그래밍 도구와 공통 인터페이스 컴포넌트를 표시하는 JavaBeans의 세트를 제공합니다. 빈을 모으고 연결하여 프로그램을 작성합니다. 대부분의 경우 코드를 작성할 필요도 없을 수 있습니다. 코드를 작성해야 할 경우 VisualAge for Java에서 코딩을 실행할 수 있는 최신 통합 개발 환경을 제공해줍니다.

이 책에서 사용된 규칙

텍스트에 다음과 같은 규칙을 사용합니다.

강조 표시 양식	사용
굵은꼴	단추나 메뉴 선택사항과 같이 선택할 수 있는 항목
기울임꼴	• 특별한 강조 • 일반 논의시 메소드명. VisualAge for Java 환경에서 선택한 메소드 이름은 굵은꼴이지만 코드 샘플의 메소드 이름은 타자체로 표시됩니다. • 특성 및 이벤트명 • 처음으로 사용하는 새로운 용어
타자체	• Java 코드의 예제 • 입력할 수 있는 텍스트

제2장 이 릴리스의 새로운 내용

VisualAge(R) for Java(TM)의 이번 버전은 다음 변경사항을 포함합니다.

- 새로운 도구
  - JavaBeans(TM) (EJB) 및 JavaServer(TM) Pages(TM) (JSP) 컴포넌트의 테스트 및 디버깅에 대한 런타임 서버를 제공하는 WebSphere(TM) 테스트 환경
  - JSP 실행 모니터와 서블릿 실행기가 포함된 JSP/서블릿 개발 환경
- 매크로 지원을 포함한 조정가능한 검색 및 IDEC에서 확장된 코드 보조
- 성능에 대해 향상된 코드 생성
- 개정된 통합 디버거
- 비주얼 컴포지션 편집기에서 지원하는 추가 특성 편집기
- Domino(TM) 5.0의 지원
- SQLJ 지원
- 추가 데이터 액세스 빈
- 프로그램 구성요소 및 Java 키워드에 대한 도움말 참조
- 공용 커넥터 프레임워크를 지원하는 새로운 e-business 커넥터 빈
- 유로 통화 지원 기능이 있는 버전 1.1.7A로 JDK(TM) 갱신
- 버전 1.0.3으로 JFC 갱신
- 비주얼 컴포지션 편집기에 Null에서 GridBag 배치 변환기
- Windows(R) 98 지원

VisualAge for Java에 대한 자세한 정보

시작하기 전에 서적에서는 VisualAge for Java에서 할 수 있는 작업을 간단하게 설명합니다. 자세한 정보는 VisualAge for Java의 임의 창의 **도움말** 메뉴에서 볼 수 있는 전체 온라인 도움말 세트를 참조하십시오.

온라인 도움말은 몇 개의 범주로 구성되어 있으며 도움말 홈 페이지나 내용 페이지에서 전체 범주에 직접 액세스할 수 있습니다.

- **개념**에서는 VisualAge for Java를 사용할 때 알아두어야 할 개념의 기초와 정의를 설명합니다.
- **참조**에는 필요한 정보를 쉽게 검색할 수 있도록 구성되어 있는 참조 정보와 작동에 대해 자세히 설명되어 있습니다.
- **타스크**에서는 타스크 수행 방법과 특정 작업을 하는 데 필요한 단계별 지침이 있습니다.
- **PDF 인덱스**는 지정된 주제에 대한 모든 정보를 인쇄할 수 있습니다.
- **샘플**은 제품과 함께 제공된 샘플을 설명합니다. 몇몇 샘플에는 비주얼 컴포지션 편집기로 빌드하는 방법에 대한 지침이 들어 있습니다.
- **용어집**에서는 VisualAge for Java에서 자주 사용하는 용어를 정의합니다.

자료 인쇄

VisualAge for Java의 도움말에서 모든 주제를 인쇄할 수 있습니다. 주제를 인쇄하려면 다음을 실행하십시오.

1. 인쇄하려는 도움말 주제를 표시하십시오.
2. 프레임을 마우스 왼쪽 단추로 클릭하여 내용 프레임(하단 오른쪽 프레임)을 선택하십시오.
3. **파일** 메뉴에서 **인쇄 프레임**을 선택하십시오.

주제에 대한 모든 내용을 인쇄하려면 PDF 인덱스에서 찾은 파일을 사용하십시오.

최신 VisualAge for Java 정보를 볼 수 있는 곳

이 설명서가 발간된 이후에 추가되거나 변경된 제품 정보를 보려면 제품에 동봉된 README 파일을 참조하십시오.

최신 정보를 구하려면 다음 웹 사이트를 책갈피에 추가하십시오. www.software.ibm.com/ad/vajava

이 사이트의 **Library** 영역에는 Java 프로그래밍 관련 서적과 논문 및 링크가 제공됩니다.

제3장 VisualAge for Java란 무엇인가?

VisualAge for Java는 Java 프로그램 개발 전체 과정을 지원하는 통합적인 비주얼 환경입니다. 특히 VisualAge for Java는 이 절에 설명된 개발 타스크 수행에 필요한 모든 것을 제공합니다.

빠른 응용프로그램 개발

VisualAge for Java의 비주얼 프로그래밍 기능을 사용하여 Java 애플릿 및 응용프로그램을 빠르게 개발할 수 있습니다. *비주얼 컴포지션 편집기*에서 다음을 선택한 다음 클릭하십시오.

- 사용자 프로그램용 인터페이스 디자인
- 사용자 인터페이스 구성요소의 동작 지정
- 사용자 인터페이스와 다른 프로그램 사용자간의 관계 정의

VisualAge for Java는 비주얼 컴포지션 편집기에서 사용자가 시각적으로 지정한 것을 구현하는 Java 코드를 생성합니다. 대부분의 경우, 사용자는 Java 코드를 직접 작성하지 않고도 완전한 프로그램을 디자인하고 수행할 수 있습니다.

비주얼 프로그래밍 기능 외에도, VisualAge for Java는 다음을 포함한 많은 타스크를 신속하게 실행할 수 있도록 안내하는 SmartGuides를 제공합니다.

- 새로운 애플릿 작성.
- 새로운 *프로그램 구성요소* 작성. VisualAge for Java에서 프로그램 구성요소는 다음 중 하나입니다.
  - **프로젝트**: VisualAge for Java에서 최상위 프로그램 구성요소 프로젝트는 패키지를 포함합니다.
  - **패키지**: Java 언어 생성. 패키지는 클래스와 인터페이스를 포함합니다.
  - **클래스**: Java 언어 생성. 클래스는 메소드와 필드를 포함합니다.
  - **인터페이스**: Java 언어 생성. 인터페이스는 메소드와 필드를 포함합니다. 인터페이스내의 필드는 static final 필드여야 합니다.

  - **메소드**: Java 언어 생성.
- JavaBeans용 기능 작성.
- 파일 시스템에서 코드 반입 및 코드 반출.

업계 강화 Java 프로그램 작성

VisualAge for Java는 업계 강화(industrial-strength) 코드 개발에 필요한 프로그래밍 도구를 제공합니다. 특히 다음을 실행할 수 있습니다.

- 완전히 통합된 비주얼 디버거를 사용해서 코드를 실행하는 중에 코드를 점검 및 갱신합니다.
- JavaBeans를 작성, 수정, 사용합니다.
- 프로젝트, 패키지, 클래스 또는 메소드 레벨에서 코드를 검색합니다.

프로그램의 복수 개정판 유지 관리

VisualAge for Java에는 사용자가 쉽게 프로그램의 복수 개정판을 유지 관리할 수 있도록 허용하는 정교한 코드 관리 시스템이 있습니다. 개정판을 *버전화*하면 언제든지 원할 때 코드 상태를 볼 수 있습니다. 버전화한 그 개정판이 읽기 전용으로 표시되어 사용자가 원하는 이름을 지정할 수 있습니다. 이런 방법으로 개발 주기에서 중요한 점검 시기의 스냅샷을 보존할 수 있습니다.

주요 개념

이 절에서는 시작하는 데 필요한 기본적인 개념 정의를 제공합니다.

저장소를 사용한 개발

VisualAge for Java 환경에서는 Java 코드 파일을 조작할 필요가 없습니다. 대신 VisualAge for Java는 *저장소*라는 구조화된 객체의 데이터베이스로 코드를 관리하고 이 코드를 프로그램 구성요소의 계층구조로 사용자에게 보여 줍니다.

 프로젝트

 패키지

 클래스 또는  인터페이스

 public  default  protected  private 메소드

파일이 아닌 프로그램 구성요소를 조작하기 때문에, 파일명이나 디렉토리 구조에 대해 크게 신경쓰지 않고 코드의 논리적 구성에만 집중할 수 있습니다.

작업영역과 저장소

VisualAge for Java의 모든 작업은 하나의 *작업영역*을 중심으로 구성됩니다. 이 작업영역에는 현재 작업하고 있는 Java 프로그램의 소스 코드가 들어 있습니다. 또한 작업영역에는 표준 Java 라이브러리와 기타 필요한 클래스 라이브러리에 있는 모든 패키지, 인터페이스, 클래스가 들어 있습니다.

작업영역에서 작성하고 저장한 모든 코드는 저장소에도 저장됩니다. VisualAge for Java에서 생성한 코드는 자동으로 저장소에 저장됩니다. 저장소는 작업영역 내 모든 코드를 저장하는 것 외에도 필요할 때 작업영역에 추가할 수 있는 다른 패키지도 보유합니다.

VisualAge for Java에서 프로그램 구성요소의 *개정판*을 작성하여 프로그램 구성요소에 변경한 내용을 관리할 수 있습니다. 작업영역에는 프로그램 구성요소의 개정판을 하나만 포함할 수 있습니다. 그러나 저장소에는 모든 프로그램 구성요소의 모든 개정판이 들어 있습니다.

Workbench

VisualAge for Java는 다른 창을 사용하여 사용자의 코드를 검사 및 조작하는 여러가지 방법을 제공합니다. VisualAge for Java에서 사용하는 기본 창을 *Workbench*라고 합니다. 이 창은 작업영역의 프로그램 구성요소를 모두 표시합니다.

[엔터프라이즈] Enterprise Edition에서는 현재 사용자가 제목 막대에 표시됩니다.

도구 막대

Workbench 도구 막대는 메뉴 막대 아래에 있으며, Workbench에서 가장 자주 수행하는 타스크에 쉽게 액세스할 수 있도록 작업을 모은 것입니다. 이러한 타스크에는 표준 편집 작업, 프로그램 구성요소 실행, 디버그, 탐색 및 조작이 포함됩니다. 특별히 프로젝트 페이지에서 도구 막대에 있는 도구는 왼쪽에서 오른쪽으로 프로젝트 및 패키지같은 프로그램 구성요소를 수행, 디버그, 탐색, 작성하고 1.0 또는 1.1과 같은 개정판 정보를 보여줍니다.

주: VisualAge for Java의 도구 막대에 있는 도구를 식별하려면 도구 위에 마우스 포인터를 놓으면 됩니다. 도구를 식별하는 레이블이 나타납니다.

Workbench 창의 페이지

각 페이지에서 작업영역의 코드를 특정 측면에서 볼 수 있습니다.

- **프로젝트** 페이지는 작업영역의 모든 프로젝트를 표시합니다. 프로젝트를 확장하여 프로젝트 안에 있는 프로그램 구성요소를 볼 수 있습니다.
- **패키지** 페이지는 작업영역의 모든 패키지를 표시합니다. 패키지를 확장하여 패키지 안에 있는 프로그램 구성요소를 볼 수 있습니다.
- **클래스** 페이지는 작업영역의 모든 클래스를 java.lang.Object를 루트로 하는 계층구조로 표시합니다. 계층을 목록으로 또는 그래픽형 열람으로 표시할 수 있습니다. 클래스를 확장하여 상속되는 클래스를 볼 수 있습니다.
- **인터페이스** 페이지는 작업영역의 모든 인터페이스를 표시합니다.
- **모든 문제점** 페이지는 미해결 문제가 있는 클래스와 메소드를 모두 표시합니다. 코드를 저장하면, VisualAge for Java가 자동으로 컴파일합니다.

코드 반입 및 반출

파일 시스템과 VisualAge for Java 사이에서 코드를 쉽게 이동할 수 있습니다. 기존 Java 코드를 VisualAge for Java로 가져오려면, 반입 SmartGuide를 사용해서 가져올 파일(또는 전체 디렉토리 구조)을 지정하십시오. VisualAge for Java는 가져온 코드를 컴파일하여 오류가 있는지 표시하고, 작업영역에 적절한 프로그램 구성요소를 추가합니다.

VisualAge for Java 외부에서 프로그램을 실행하려면 반출 SmartGuide를 사용하여 반출할 수 있습니다. VisualAge for Java는 반출한 각 클래스에 대해 Java 소스(*.java) 파일이나 컴파일된 (*.class) 파일을 만듭니다.

비주얼 컴포지션 편집기를 사용한 비주얼 프로그래밍

비주얼 컴포지션 편집기는 JavaBeans 또는 간단하게 *빈*이라고 하는 소프트웨어 객체를 시각적으로 배열하고 연결하여 프로그램을 개발할 수 있는 VisualAge for Java의 한 영역입니다. 그래픽으로 표현된 컴포넌트를 조작하여 객체 지향 프로그램을

작성하는 이런 과정을 *비주얼 프로그래밍*이라고 합니다. 사용자가 직접 만든 코드를 사용하여 단추 클릭하기 같은 비주얼 이벤트를 링크할 수도 있습니다.

빈

VisualAge for Java에서 *빈*은 시각적으로 프로그래밍할 때 조작하는 컴포넌트입니다. 이러한 빈은 JavaBeans 스펙을 따르는 Java 클래스입니다. 비주얼 컴포지션 편집기에서, 팔레트에서 빈을 선택하여 그 특성을 지정하고, 연결을 작성합니다. 빈은 기타 빈과 빈에 대한 연결을 포함합니다. VisualAge for Java에서 빈의 역할에 대한 자세한 내용은 139 페이지의 『JavaBeans 컴포넌트 지원』을 참조하십시오.

비주얼 컴포지션 편집기에서는 두 가지 유형의 빈을 사용합니다.

- *비주얼* 빈은 실행시 프로그램에서 볼 수 있습니다. 창, 단추, 텍스트 필드와 같은 비주얼 빈이 프로그램의 그래픽 사용자 인터페이스(GUI)를 구성합니다.
- *넌-비주얼* 빈은 실행시 프로그램에 나타나지 않습니다. 넌-비주얼 빈은 보통 프로그램 내에서 데이터를 캡슐화하고 행동을 구현하는 객체를 나타냅니다.

빈의 *public 인터페이스*는 다른 빈과 상호작용하는 방식을 결정합니다. 빈의 public 인터페이스는 다음 기능으로 구성됩니다.

- *특성*은 다른 빈에서 액세스할 수 있는 데이터입니다. 이 데이터는 계좌 정산(balance of an account), 선적 규모 또는 단추의 레이블과 같은 빈의 논리적 특성을 표시할 수 있습니다.
- *이벤트*는 무언가 발생했음을 나타내는 신호입니다. 예를 들어, 창을 열거나 특성 값을 변경하는 작업이 이벤트를 트리거합니다.
- *메소드*는 빈이 수행할 수 있는 작동입니다. 메소드는 기타 빈으로부터의 연결에 의해 트리거될 수 있습니다.

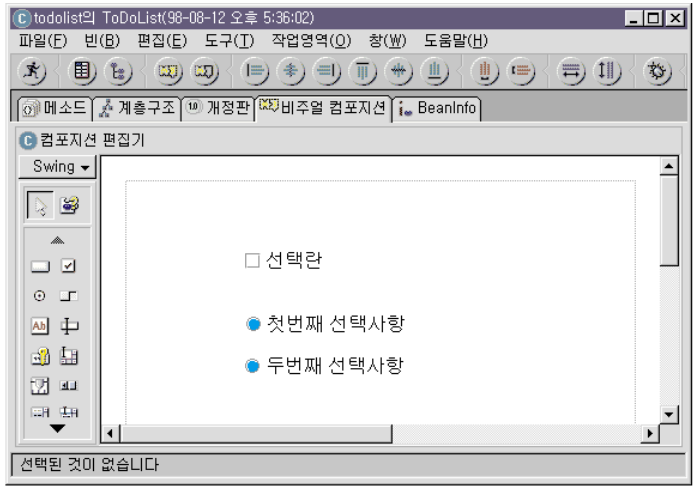
연결

비주얼 컴포지션 편집기에서 *연결*은 빈이 서로 상호작용하는 방식을 정의합니다. 빈과 빈을 연결하거나 서로 다른 연결을 연결할 수 있습니다. 연결은 *소스*와 *대상*을 가집니다. 연결을 시작하는 지점이 *소스*이고 연결이 끝나는 지점이 *대상*입니다.

비주얼 컴포지션 편집기

비주얼 컴포지션 편집기는 VisualAge for Java에 통합되어 있는 비주얼 프로그래밍 툴입니다. 이것은 사용자가 클래스를 검색할 때 나타나는 창의 한 페이지입니다.

비주얼 컴포지션 편집기는 여러 개의 컴포넌트로 구성됩니다. 왼쪽에는 *빈 팔레트*, 아래에는 *상태 영역*, 상단에는 *도구 막대*가 있고, 빈을 배치하는 *빈 작업 영역*이 있습니다. 아래 그림에는 세 개의 빈, 즉 하나의 선택란 빈과 두 개의 라디오 단추 빈이 있습니다.



비주얼 컴포지션 편집기를 사용해서 새로운 빈을 만들 수 있습니다. 이러한 새로운 빈은 다른 빈을 포함할 수도 있고, 다른 빈간의 연결을 포함할 수도 있습니다. 비주얼 컴포지션 편집기에서 사용자가 작성한 빈은 다른 빈을 포함하기 때문에, 이를 컴포지트 빈으로 생각할 수 있습니다. 사용자가 작성한 컴포지트 빈이 프로그램을 구성합니다.

빈 팔레트

비주얼 컴포지션 편집기의 왼쪽에 있는 빈 팔레트에는 가장 자주 사용하는 빈 세트가 미리 만들어진 형태로 나와 있습니다. 빈 팔레트는 범주별로 빈을 조직합니다.

비주얼 컴포지션 편집기 아래에 있는 상태 영역은 현재 빈 팔레트에서 선택한 범주와 빈을 표시하거나 빈 작업 영역에서 현재 선택한 빈 또는 연결을 표시합니다.

주: 빈 아이콘 위에 마우스 포인터를 놓아 빈을 식별할 수 있습니다. 그러면 아이콘을 식별하는 레이블이 나타납니다.

도구 막대

비주얼 컴포지션 편집기의 메뉴 막대 아래에 있는 도구 막대를 이용하면 빈을 조작하는 동안 많이 사용하는 도구에 쉽게 액세스할 수 있습니다. 이러한 도구를 사용하면 빈 위치 지정, 빈 크기조정, 빈 사이의 연결 표시하기 또는 숨기기, 사용자 프로그램 테스트와 같은 작업을 하는 데 도움이 됩니다. 특히 왼쪽에서 오른쪽으로 배치되어 있는 도구들은 각각 실행, 특성 지정, 빈 목록 제공, 연결 표시 및 숨기기, 여러가지 방법으로 작업영역에 빈 배치, 디버그를 수행하는 도구입니다.

도구 막대에 있는 대부분의 도구는 빈 작업 영역에서 현재 선택되어 있는 빈에 작용합니다. 도구를 사용할 빈을 선택하지 않으면, 도구를 사용할 수 없습니다.

주: 도구 메뉴를 사용하여 이런 도구에 액세스할 수도 있습니다.

빈 작업 영역

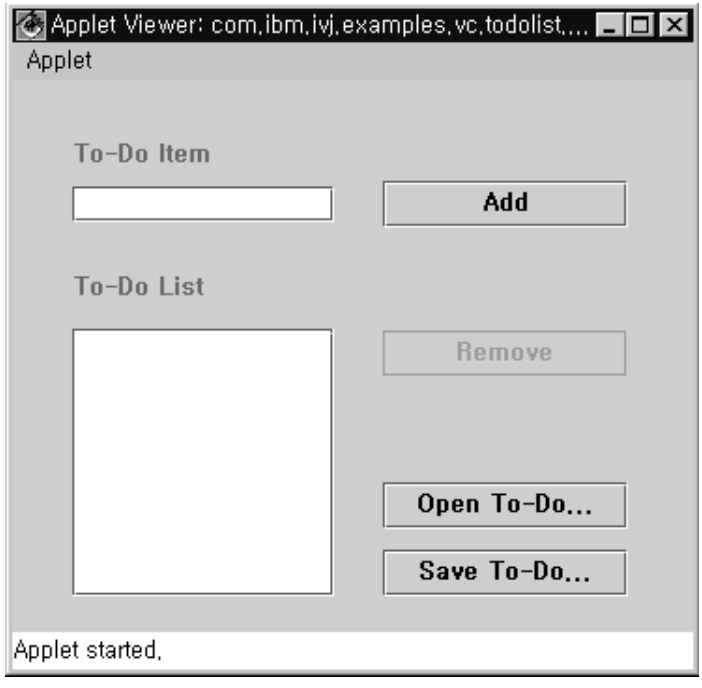
비주얼 컴포지션 편집기의 오른쪽에 있는 빈 작업 영역은 프로그램을 구성하는 영역입니다. 넌-비주얼 빈을 비주얼 빈 위에 드롭할 수 없습니다.

빈의 유형에 상관없이 모든 빈에는 그 빈을 수정하거나 그에 대해 작업하는 데 사용할 수 있는 옵션을 표시하는 팝업 메뉴가 있습니다.

제4장 첫번째 애플릿 빌드 소개

이 절에서는 VisualAge for Java에서 첫번째 애플릿인 To-Do List의 빌드를 안내해 줍니다.

기타 많은 빈으로 이루어진 빈(*컴포지트* 빈)으로 구성된 To-Do List 애플릿을 작성합니다. 이 애플릿은 To-Do Item 입력용 JTextField 빈과 To-Do Item 표시용 JList 빈을 포함합니다. 또한 목록으로부터 항목 추가 및 제거를 위한 두 개의 JButton 빈이 있습니다. 완성된 To-Do List 애플릿의 사용자 인터페이스는 다음과 같습니다.



완성된 애플릿에서, 사용자는 **To-Do Item** 필드에 항목을 입력하고 **Add**를 선택합니다. 그러면 항목이 To-Do List에 추가됩니다. To-Do List에서 항목을 선택하고 **Remove**를 선택하면 To-Do List에서 항목이 제거됩니다.

첫번째 애플릿을 시작하기 전에

VisualAge for Java를 아직 설치하지 않았으면 제품 CD에 있는 readme.txt 파일에서 제품 설치 방법을 참고하십시오. VisualAge for Java 설치 프로그램은 사용자 개발 환경에 필요한 모든 파일을 설치합니다.

VisualAge for Java 시작

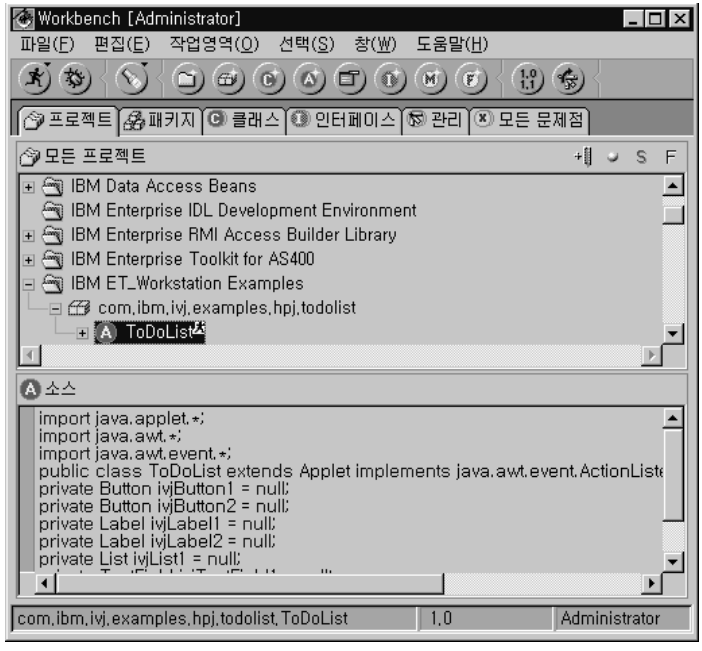
다음 중 하나를 실행하여 VisualAge for Java를 시작할 수 있습니다.

- OS/2의 경우 VisualAge for Java 폴더에서 IDE 아이콘  을 두 번 클릭하십시오.
- Windows 95 및 Windows NT의 경우, **VisualAge for Java**를 선택하고 **시작 - 프로그램** 메뉴에서 **IDE**를 선택하십시오.

[엔터프라이즈] 처음으로 VisualAge for Java, Enterprise Edition을 시작하는 경우 사용자 작업영역의 소유자와 Administrator라고 부르는 사용자에 대한 네트워크 이름을 선택하는 프롬프트가 표시됩니다. 이 예제는 연습용이므로 작업영역 소유자로 **Administrator**를 선택할 수 있습니다. 네트워크 이름은 네트워크에서 사용자 컴퓨터를 확인하는 이름을 사용합니다. 팀 개발 환경에 대한 자세한 정보는 VisualAge for Java Enterprise Edition의 *시작하기 전에* 서적을 참조하십시오.

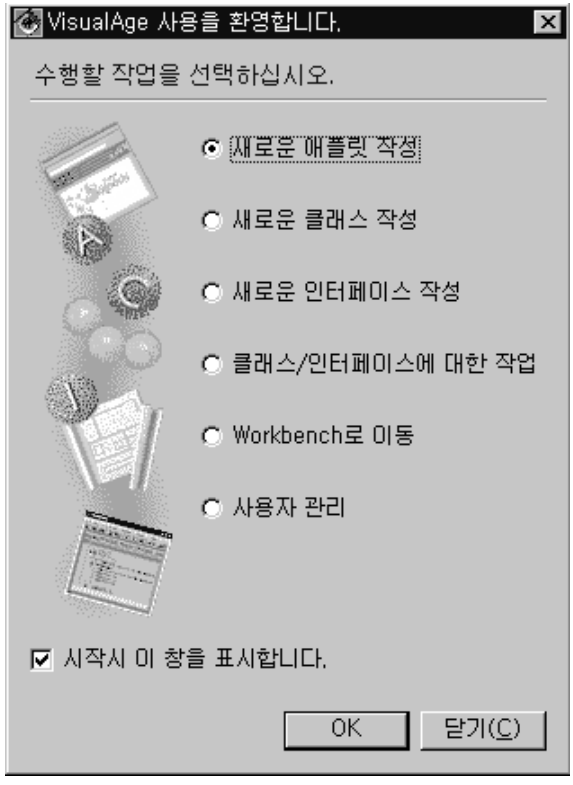
이제 수행하고 실행하기

VisualAge for Java를 시작하면 Workbench 창이 나타납니다.



Workbench 창은 기타 창으로의 액세스, 프로그램 구성요소 작성 및 프로그램 구성요소 내용 열람에 사용됩니다.

다음으로 VisualAge for Java 환영 창이 나타납니다.



VisualAge for Java 환영 창은 애플릿, 클래스, 인터페이스를 작성하는 빠른 경로를 제공합니다.

Workbench로 이동을 선택하고 **OK**를 클릭하십시오.

팁: 빠른 시작 창을 사용해서 IDE에서 일반적인 타스크를 쉽고 빠르게 수행할 수도 있습니다. 프로그램 구성요소 작성, 스크랩북 사용법 학습, 저장소 관리, 샘플 및 유용한 빈을 작업영역에 추가하는 등의 작업을 할 수 있습니다. 이 창은 언제든지 F2 키를 누르거나 IDE 검색기에서 **파일** 메뉴의 **빠른 시작**을 선택하여 시작할 수 있습니다. IDE에서 사용자 소유의 프로그램을 작성 및 관리할 때 이를 염두에 두십시오. 자세한 정보는 131 페이지의 『빠른 시작 창 사용』을 참조하십시오.

SmartGuide 사용

To-Do List 애플릿에 대해 애플릿은 물론 사용자 작업을 포함하는 프로젝트와 패키지도 작성합니다. SmartGuide를 사용하여 작성할 수 있습니다. SmartGuide는 VisualAge for Java 빠른 시작 창에서 액세스할 수 있습니다.

비주얼 컴포지션 편집기에서 새로운 애플릿을 조작할 때 JavaBeans를 시각적으로 조작하게 됩니다. 이러한 JavaBeans(또는 단순히, *빈*)는 Workbench에서 애플릿 점검시 클래스로 표시됩니다.

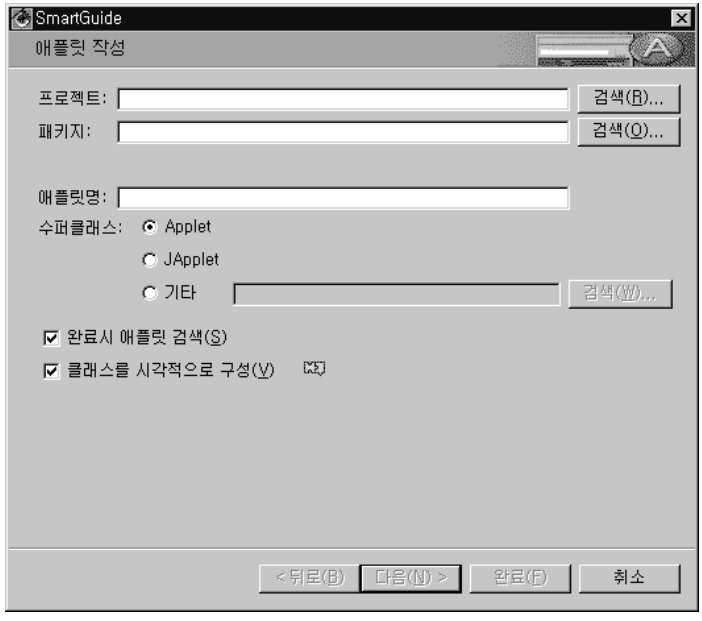
VisualAge for Java는 사용자에게 대문자로 시작하는 애플릿(및 기타 모든 클래스) 이름을 제공하도록 제안합니다. 클래스명은 대소문자를 구분해야 하며 공백을 포함할 수 없습니다. 클래스명이 여러 단어로 구성된 경우, 단어 사이에 공백을 입력하지 말고 대신 각 단어의 첫글자를 대문자로 입력하십시오(예: *ToDoList*).

애플릿, 프로젝트 및 패키지 작성

SmartGuide를 열려면, Workbench **파일** 메뉴에서 **빠른 시작**을 선택하십시오. 빠른 시작 창에서 다음을 수행하십시오.

1. **기본**을 선택한 다음 **애플릿 작성**을 선택하십시오.

2. **OK**를 선택하십시오. 애플릿 작성 SmartGuide가 열립니다.



애플릿 작성 SmartGuide에서 애플릿을 작성하려면 다음을 수행하십시오.

1. **프로젝트** 필드에 *My ToDoList Project*와 같은 프로젝트명을 입력하십시오.
2. **패키지** 필드에 *todolist*와 같은 패키지명을 입력하십시오.
3. **애플릿명** 필드에 *ToDoList*와 같은 이름을 입력하십시오.
4. **최상위 클래스** 필드에서 검색을 사용하여 **JApplet**을 선택하십시오. 여기서는 `com.sun.java.swing.JApplet`이 됩니다.

   애플릿(java.applet.Applet)을 AWT(Abstract Windowing Toolkit)와 같이 사용하지 마십시오. 빌드중인 애플릿은 Swing 컴포넌트를 사용합니다. 결과적으로 Swing 빈은 JApplet 최상위 클래스와 함께 사용해야 하고, AWT 빈은 애플릿 최상위 클래스와 함께 사용해야 합니다.
5. **클래스를 시각적으로 구성**을 선택했는지 확인하십시오.

   비주얼 컴포지션 편집기를 사용하면 이와 같은 프로그램 구성요소를 빨리 작성할 수 있지만 원하면 직접 코드를 작성할 수도 있습니다. 데모에 대해서는 128 페이지의 『직접 간단한 빈 작성』을 참조하십시오.
6. **완료**를 선택하십시오.

VisualAge for Java는 프로젝트, 패키지 및 애플릿을 작성한 후 그 애플릿에서 비주얼 컴포지션 편집기를 여십시오.

비주얼 컴포지션 편집기 사용

비주얼 컴포지션 편집기가 열릴 때, 사용자의 To-Do List 애플릿을 시각적으로 작성할 수 있습니다. To-Do List 애플릿은 몇몇 비주얼 빈을 포함하는 하나의 빈으로 구성됩니다.

빈에 대한 작업

VisualAge for Java에서 빈에 대해 작업할 때 빈 끌기 조작, 복수 빈 선택, 팝업 메뉴 표시 등의 기본적인 기술을 사용합니다.

빈 끌기

빈을 끌어서 놓으려면 적절한 마우스 단추로 클릭한 채로 이동하십시오. (OS/2에서는 마우스 오른쪽 단추를 사용하여 빈을 끌고, UNIX 플랫에서는 가운데 단추 그리고 Windows에서는 왼쪽 단추를 사용하십시오.) 십자형을 원하는 위치로 이동하고 단추에서 손을 떼십시오.

여러 빈 선택

여러 개의 빈을 선택하려면, Ctrl 키를 누른 채 선택하려는 항목을 마우스 왼쪽 단추로 클릭하십시오. 이것을 *선택 세트*라고 합니다.

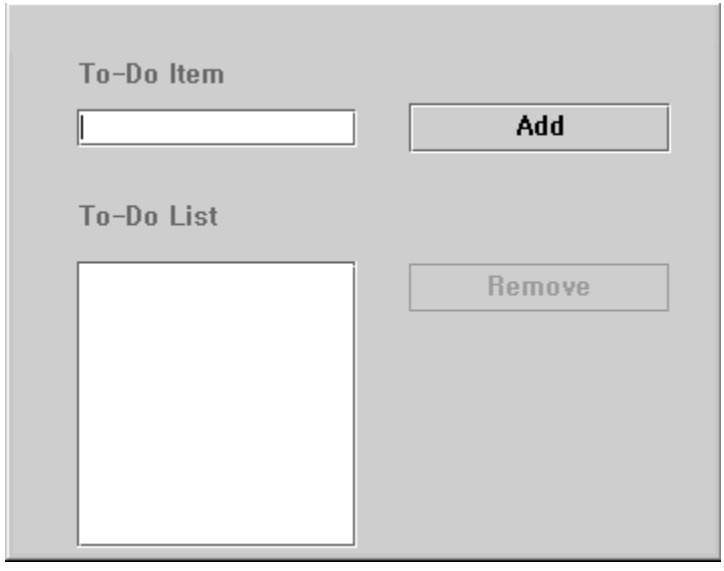
주: 세트로 작동할 수 있는 빈만 선택 세트에 포함될 수 있습니다. 예를 들어, 창 내에 지정된 빈 세트는 *크기조정* 또는 정렬의 목적으로 함께 선택될 수 있습니다. 그러나 창 빈과 창에 포함되는 빈들 중 하나를 함께 선택할 수 없습니다.

빈 팝업 메뉴 표시

팝업 메뉴를 표시하려면, 빈을 마우스 오른쪽 단추로 클릭하십시오.

To-Do List 애플릿 빌드

비주얼 컴포지션 편집기를 열고 새로운 ToDoList 애플릿을 열면 기본 JApplet 빈이 빈 작업 영역에서 회색 사각형으로 표시됩니다. 사용자 인터페이스의 나머지를 작성하려면, 기타 비주얼 빈을 몇개 추가해야 합니다. To-Do List 애플릿용 사용자 인터페이스 작성 완료시, 비주얼 컴포지션 편집기의 빈 작업 영역은 다음과 같이 나타납니다.



다음과 같은 사용자 인터페이스를 작성하려면, 나머지 빈을 추가, *크기조정* 및 정렬해야 합니다.

주: 빈을 애플릿에 추가할 때 기본 JApplet 빈이 다른 빈을 모두 수용하기에 작은 경우가 있습니다. 이럴 때는 마우스 왼쪽 단추로 선택 핸들을 끌어 JApplet 빈의 *크기*를 조정할 수 있습니다.

이 빈은 <null> 배치 관리 프로그램을 사용합니다. 이 빈의 사용자 인터페이스를 GridBagLayout 배치로 작성하는 예제를 보려면 온라인 도움말을 참고하십시오. <null>에서 다른 배치 관리 프로그램으로 변환할 수도 있습니다.

텍스트 필드와 레이블 추가

애플릿 작성 중 이 단계에서는 To-Do Item을 입력하는 데 사용되는 JTextField 빈과 필드를 식별하는 JLabel 빈을 추가합니다. 이러한 빈들은 빈 팔레트의 Swing 범주에 있습니다.

팁: 팔레트에서 작업할 때 *크기*가 너무 작아 사용하기에 불편한 아이콘이 있기도 합니다. 이러한 아이콘을 *크게* 만들려면, 팔레트의 회색 영역을 마우스 오른쪽 단추로 클릭하고 팝업 메뉴 분할영역에서 **큰 아이콘 보기**를 선택하십시오.

1. 빈 팔레드에서 JTextField 빈  을 선택하십시오. 비주얼 컴포지션 편집기 아래의 정보 영역에는 현재 빈 팔레트에서 선택한 사항이 범주: Swing 빈: JTextField와 같이 표시됩니다.
2. JApplet 빈(빈 작업 영역 위의 사각형) 위로 마우스 포인터를 이동하십시오. 이 때 포인터는 사용자가 선택한 빈과 함께 로드되었음을 나타내기 위해 십자형으로 변경됩니다. JTextField를 추가하려는 위치에서 마우스 왼쪽 단추를 클릭하십시오.

   빈을 잘못 선택하고 아직 JApplet 빈에 드롭하지 않았으면 정확한 빈을 선택하거나, 빈 팔레트에서 선택 도구  를 선택하여 마우스 포인터를 로드 해제하십시오.

   JTextField 빈을 JApplet에 추가한 후 이를 마우스로 끌어서 새로운 위치로 이동할 수 있습니다,

   - OS/2에서는 마우스 오른쪽 단추로 빈을 클릭한 다음 누른 채로 끌기하십시오.
   - UNIX 플랫폼에서는 마우스 가운데 단추를 사용하십시오.
   - Windows에서는 마우스 왼쪽 단추를 사용하십시오.

   마우스 왼쪽 단추로 사각형의 면을 끌어 빈 *크기*를 조정할 수 있습니다.

3. JLabel 빈  을 선택하고 JTextField 바로 위에 추가하십시오.

   위치는 정확하지 않아도 됩니다. 도구 막대에서 도구를 사용한 빈의 규모조정 및 정렬 방법은 뒤에서 배웁니다.

레이블 텍스트 변경 및 다른 레이블 추가

다음 설명대로 텍스트를 편집하여 JLabel의 텍스트를 To-Do Item으로 변경하십시오.

1. JLabel 빈을 선택하십시오. 도구 막대에서 특성  을 선택하십시오.
2. 특성 창에서 텍스트 필드를 클릭한 다음 JLabel1 대신 To-Do Item을 입력하십시오. 레이블 텍스트가 **To-Do Item**으로 표시됩니다. 레이블 텍스트를 보려면 레이블을 확장해야 합니다.
3. JLabel을 복사하여 JTextField 아래에 또다른 레이블을 추가할 수 있습니다. 빈을 복사하려면, 다음을 실행하십시오.
   a. 빈을 선택하십시오.
   b. **편집** 메뉴에서 **복사**를 선택하십시오.
   c. **편집** 메뉴에서 **붙여넣기**를 선택하십시오. 이때 포인터는 사용자가 선택한 빈과 함께 로드되었음을 나타내기 위해 십자형으로 변경됩니다.
   d. JTextField 아래를 마우스 왼쪽 단추로 클릭하여 새로운 레이블을 추가하십시오.
4. JLabel1에서 한 것처럼 텍스트를 편집하여 새로운 레이블의 텍스트를 To-Do List로 변경하십시오.

팁: Ctrl 키를 사용하여 빈을 복사할 수도 있습니다. 마우스 포인터를 JLabel 빈 위에 놓고 Ctrl 키를 누른 채 빈의 사본을 JTextField 빈 아래로 끌기 조작하십시오.

사용자 목록에 화면 이동 분할영역 추가

항목 목록을 화면 이동할 수 있도록 화면 이동 분할영역을 추가하십시오.

1. JScrollPane 빈  을 선택하십시오(이 빈은 팔레트에 있는 컨테이너 빈의 그룹에서 선택할 수 있습니다.)
2. JScrollPane을 텍스트 필드 아래에 추가하십시오.

목록 추가

To-Do Item을 표시하는 목록을 작성하려면, JList를 추가해야 합니다.

1. JList 빈  을 선택하여 화면이동 분할영역 내에 놓으십시오. JList 빈은 JScrollPane을 채우도록 조정됩니다.
2. 특성 창에서 *selectionMode* 특성을 SINGLE_SELECTION으로 변경하십시오. 이는 목록에서 선택을 처리하는 데 필요한 코드를 단순화시킵니다.
3. 사용자가 수행한 작업을 정기적으로 저장하는 것이 좋습니다. 사용자 작업을 저장하려면, **빈** 메뉴에서 **빈 저장**을 선택하십시오.

단추 추가

To-Do List에서 항목을 추가 및 제거하려면, 두 단추를 추가해야 합니다.

1. 한번에 둘 이상의 빈 인스턴스를 추가하려면, Ctrl 키를 누른 채로 빈을 선택하십시오.

2. JButton 빈  을 선택하고 텍스트 필드 오른쪽에 단추를 추가하십시오.

   마우스 포인터는 여전히 십자형으로 남아 있는데, 이는 마우스 포인터가 단추 빈과 함께 로드되었음을 나타냅니다. 또다른 JButton을 추가하려면 JButton1 아래를 마우스 왼쪽 단추로 클릭하십시오.

3. 빈 팔레트에서 선택 도구  를 선택하여 마우스 포인터를 로드 해제하십시오.
4. Label 빈에서처럼 JButton1의 텍스트를 `Add`로 변경하십시오. 특성 시트에서 빈 이름을 AddButton으로 명명하십시오.
5. JButton2의 텍스트를 `Remove`로 변경하십시오. 특성 시트에서도 빈 이름을 RemoveButton으로 명명하십시오.
6. **빈** 메뉴에서 **빈 저장**을 선택하여 수행한 작업을 저장하십시오.

축하합니다! VisualAge for Java를 사용하여 첫번째 사용자 인터페이스를 작성했습니다. 다음은, To-Do List 애플릿에서 빈을 규모조정 및 정렬해야 합니다.

비주얼 빈 크기조정 및 정렬

이 빈은 배치 관리 프로그램을 사용하지 않으므로, 사용자 인터페이스의 외관을 정리하려면 비주얼 컴포지션 편집기의 도구 막대에서 *크기조정 및 정렬* 도구를 사용해야 합니다. 도구 막대는 빈 *크기조정 및 정렬*을 위해 몇가지 도구를 제공합니다. 여러가지 도구로 실험해서 그 사용 방법에 대해 알아보십시오.

다음의 단계는 두 빈의 *크기* 일치, 빈과 기타 빈 정렬, 다른 빈 내에서의 빈 균일 분배 방법을 설명합니다. 빈 *크기조정*과 변경에 대해서는 63 페이지의 『빈 조작』에서 자세하게 배웁니다.

빈 크기조정, 정렬 및 분배

빈을 *크기조정*하고, 정렬하고, 분배하는 순서가 항상 중요한 것은 아닙니다. 보통 상단 왼쪽 모서리에서 시작하여 창 내 모든 빈을 통과하며 작업합니다.

텍스트 필드의 폭과 일치하도록 목록을 *크기조정*하려면, 다음을 수행하십시오.

1. 도구 막대에서 빈 목록  을 선택하십시오.
2. 목록에서 JScrollPane1을 선택하십시오.

   목록을 담을 컨테이너 *크기*를 조정하려고 한다는 점을 기억하십시오. 컨테이터는 화면이동 분할영역입니다. JList 빈은 거의 완전히 JScrollPane 빈을 덮어써서 빈 작업 영역에서 선택하기 어렵기 때문에 빈 목록을 사용하게 됩니다.
3. Ctrl 키를 누른 채로 여러 항목을 선택한 다음 *빈 작업 영역에서* 마우스 왼쪽 단추로 텍스트 필드를 선택하십시오.
4. 도구 막대에서 폭 일치 도구  를 선택하십시오.

   텍스트 필드가 마지막으로 선택되었기 때문에 이것이 폭 일치 작동의 앵커 빈(anchor bean)이 됩니다. 목록의 폭은 텍스트 필드의 폭과 일치하도록 변경됩니다.

주: 앵커 빈에는 단색 선택 핸들이 있습니다. 기타 선택된 항목은 윤곽선을 가진 선택 조정을 포함합니다.

Add 단추와 **Remove** 단추를 *크기조정* 및 정렬하려면, 다음을 실행하십시오.

1. 애플릿에 맞게 적절한 *크기*가 되도록 **Remove** 단추의 *크기*를 변경하십시오.
2. **Add** 단추를 선택하고 Ctrl 키를 누른 채로 마우스 왼쪽 단추로 **Remove** 단추를 선택하십시오. 도구 막대에서 폭 일치 도구  를 선택하십시오.
3. 단추가 선택된 상태이기 때문에 도구 막대에서 왼쪽 정렬 도구  를 선택하여 단추 왼쪽 가장자리를 정렬하십시오.

텍스트 필드, 목록 및 레이블의 왼쪽을 정렬하려면, 다음을 수행하십시오.

1. 애플릿에서 원하는 위치로 텍스트 필드의 레이블(**To-Do Item**)을 이동하십시오.
2. 화면 이동 분할영역(목록이 아니라 화면 이동 분할영역), 텍스트 필드 및 관련 레이블을 선택하십시오.(텍스트 필드의 레이블을 마지막에 선택해야 합니다.)

   텍스트 필드 레이블을 마지막으로 선택하여, 이것을 정렬 작동의 앵커 빈으로 만드십시오.
3. 도구 막대에서 왼쪽 정렬 도구  를 선택하십시오.
4. 텍스트 필드, 목록, 레이블이 선택되어 있으므로 도구 막대에서 수직 분배 도구  를 선택하여 창에서 나란히 분배하십시오.
5. 수행한 작업을 저장하십시오.

이제 To-Do List 애플릿의 사용자 인터페이스를 완전히 완료했습니다.

주: 작성하고 있는 전체 애플릿이 빈입니다. 메뉴에서 **빈**을 선택한 다음 **빈 저장**을 선택하면, 모든 애플릿이 저장됩니다.

실수 정정

비주얼 컴포지션 편집기에서 변경했다가 원래대로 두고 싶다면, **편집** 메뉴에서 **실행 취소**를 선택하여 작업을 이전 상태로 복원하십시오. 현재 빈에 대한 비주얼 컴포지션 편집기를 열었을 때로 돌아갈 때까지 얼마든지 작업을 실행 취소할 수 있습니다.

작업을 실행 취소했다가 먼저 바로 전의 작업을 재실행하기로 결정했다면, **편집** 메뉴에서 **재실행**을 선택하십시오. **재실행**은 뷰를 마지막 **실행 취소** 이전의 상태로 복원합니다. 사용자 빈용 비주얼 컴포지션 편집기를 닫자마자, 변경의 실행 취소 또는 다시 실행 능력을 잃게 됩니다.

빈 연결

사용자 인터페이스를 만들기 위해 비주얼 빈을 추가했으면, 다음 단계는 이들을 연결하는 것입니다.

이벤트와 메소드 및 특성에서 매개변수 연결

To-Do List 애플릿은 **Add** 단추를 선택할 때 텍스트 필드에 입력한 텍스트를 목록에 추가해야 합니다. 이 예제에서는 DefaultListModel이라는 모델을 포함하도록 사용자 목록을 확장합니다. Swing으로 알려져 있는 Java Foundation클래스는 데이터를 데이터의 열람으로부터 분리합니다. 실제 목록 항목은 목록 모델에 저장되어 있습니다. *이벤트와 메*소드 연결은 애플릿의 목록으로 모델의 데이터를 전송합니다.

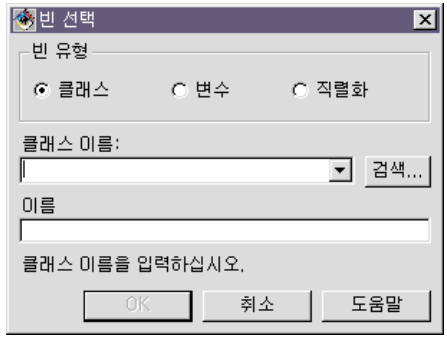
단추를 선택하면 *actionPerformed(java.awt.event.ActionEvent)* 이벤트에 신호를 보내고 목록 모델에 항목을 추가하는 것은 *addElement(java.lang.Object)* 메소드가 수행하므로, 목록 모델에 항목을 추가하는 이벤트와 메소드 연결은 AddButton의 *actionPerformed(java.awt.event.ActionEvent)* 이벤트와 DefaultListModel의 *addElement(java.lang.Object)* 메소드 사이에서 연결됩니다.

단순히 이벤트와 메소드 연결을 추가하면 실제로는 목록 모델에 아무것도 추가되지 않습니다. 그 이유는 *addElement(java.lang.Object)* 메소드가 목록에 추가할 개

체를 지정하는 매개변수를 필요로 하기 때문입니다. *특성에서 매개변수* 연결을 작성하여 매개변수를 지정합니다. 텍스트 필드의 내용은 목록 모델의 *addElement (java.lang.Object)* 메소드에 대한 매개변수로서 제공됩니다.

목록 모델 추가 및 연결 작성

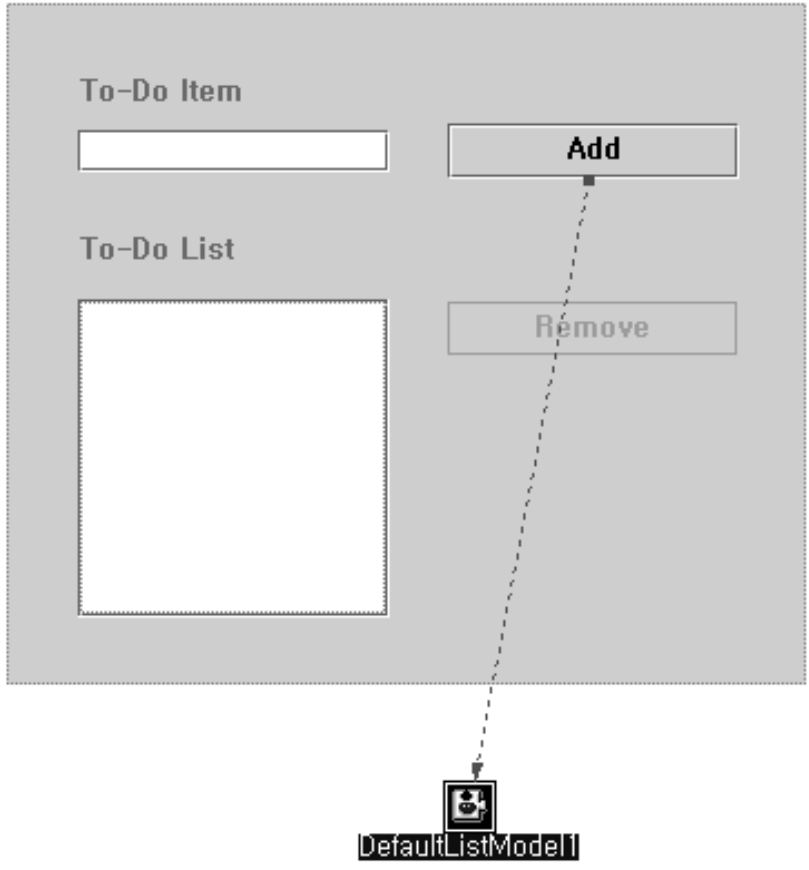
1. 팔레트에서 빈 선택 도구  를 선택하십시오.
2. 빈 선택 창에서 빈 유형으로 **클래스**를 선택하십시오. **검색**할 때 de 패턴을 사용하여, DefaultListModel 클래스를 선택하십시오.



   OK를 클릭하고 DefaultListModel 빈을 빈 작업 영역의 애플릿 아래(회색으로 표시된 영역)에 놓으십시오. 기본값으로 새로운 빈 이름은 DefaultListModel1입니다.
3. 이제 항목을 목록으로 이동하기 위한 연결을 시작할 수 있습니다. AddButton을 선택하고 마우스 오른쪽 단추를 클릭한 다음 팝업 메뉴에서 **연결**, **actionPerformed**를 선택하십시오.

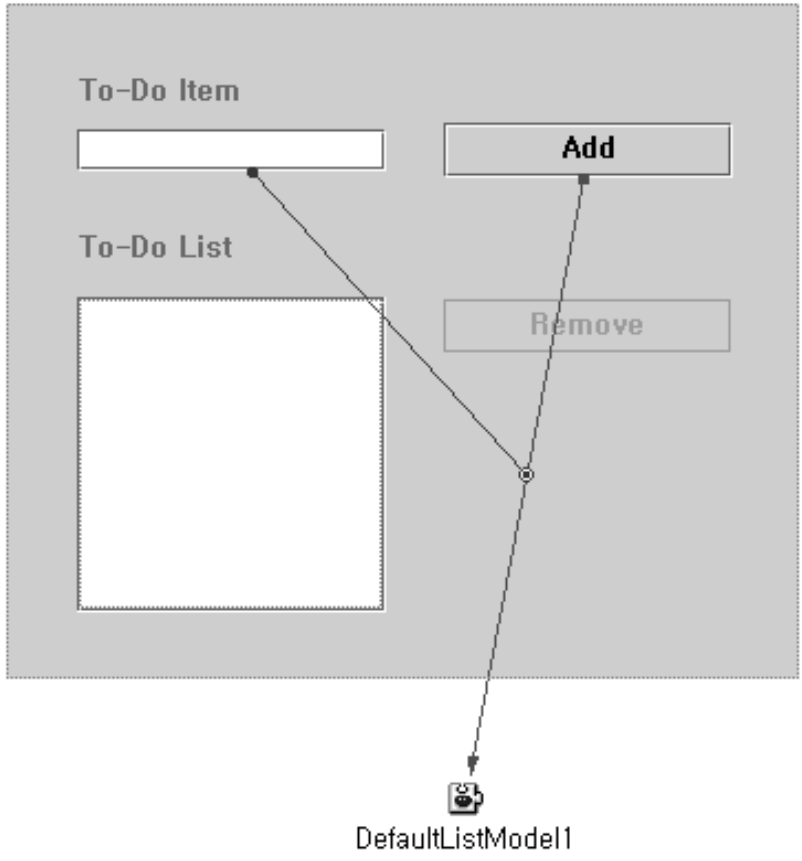
   마우스 포인터가 연결중임을 나타내는 모양으로 바뀝니다. 실수로 잘못된 연결을 시작한 경우, Esc 키를 눌러 취소하십시오.
4. 연결을 완성하려면 DefaultListModel1을 마우스 왼쪽 단추로 클릭한 다음 **연결가능 기능**을 선택하십시오. 이때 나타나는 팝업 메뉴에서 **addElement(java.lang.Object)**를 클릭하십시오. 점선이 나타나 더 많은 정보가 필요함을 표시합니다. 이런 경우에 *addElement(java.lang.Object)* 메소드에 대한 매개변수 값

이 없습니다.



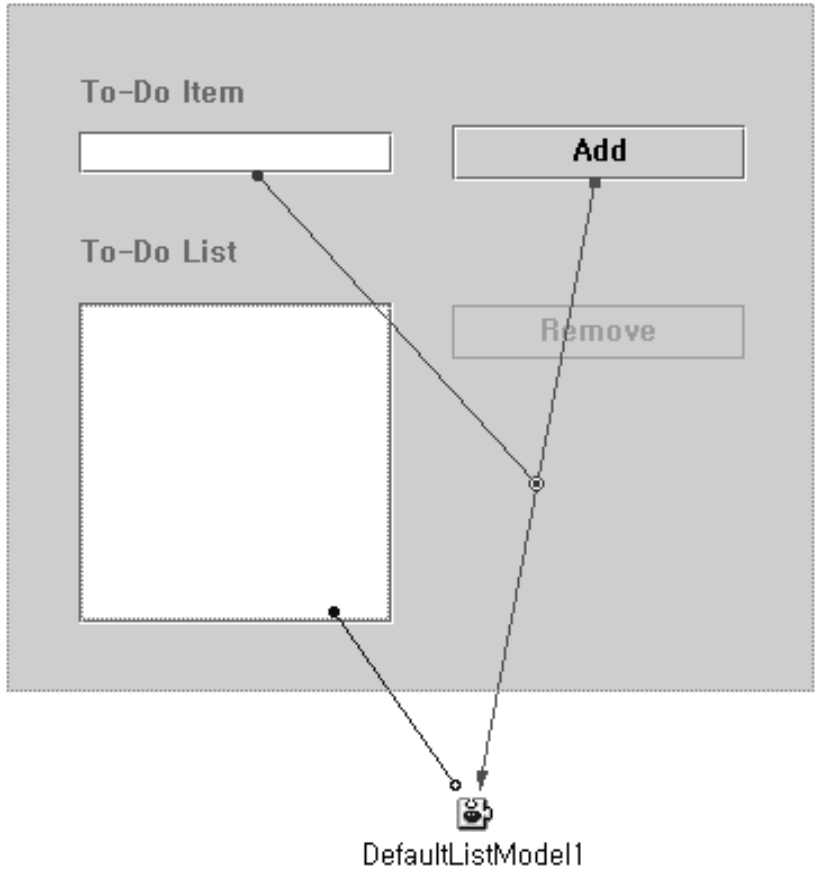
5. 목록에 추가할 것을 지정하는 특성에서 매개변수 연결을 작성하려면, 다음 단계를 따르십시오.
   - 이벤트와 메소드 연결의 점선 링크 위로 마우스 포인터를 이동하십시오.
   - 마우스 오른쪽 단추를 클릭하십시오. 팝업 메뉴에서 **연결**을 선택한 다음 **obj**를 선택하십시오.

- 텍스트 필드를 마우스 왼쪽 단추로 클릭한 다음 팝업 메뉴에서 **텍스트**를 선택하십시오.



6. 마지막으로 사용자가 애플릿에서 볼 수 있도록 To-Do Item을 DefaultListModel1에서 애플릿의 목록으로 가져와야 합니다. 이것은 소스(모델)에서 열람(목록)으로 데이터를 전송하는 연결입니다. 마우스 오른쪽 단추로 DefaultListModel을 클릭하십시오. 팝업 창에서 **연결**을 선택한 다음 **this**를 선택하십시오. 목록을 마우스 왼쪽 단추로 클릭한 후 팝업 메뉴에서 **모델**을 선

택하십시오. 그림처럼 특성간 연결을 의미하는 파란 선이 나타납니다.



7. 수행한 작업을 저장하십시오.

이벤트와 코드 연결

이벤트와 코드 연결 프로그래밍은 직접 작성한 메소드를 이벤트와 연관시키는 데 사용됩니다. 작성한 첫번째 애플릿에서, VisualAge for Java가 비주얼 구성요소와 연결을 기초로 메소드를 생성했기 때문에 어떤 메소드도 작성할 필요가 없었습니다. 그러나 때때로 추가 논리가 요구됩니다.

이 예제에서는 이벤트와 코드 연결을 사용하여 **Remove** 단추를 활성화합니다. 이벤트와 코드 연결을 사용하여 한 메소드에 여러 개의 동작을 지정할 수 있습니다. 직접 코드를 쓰지 않고 이렇게 할 수 있지만 여기서는 직접 코드를 작성하는 것이 더 능률적입니다.

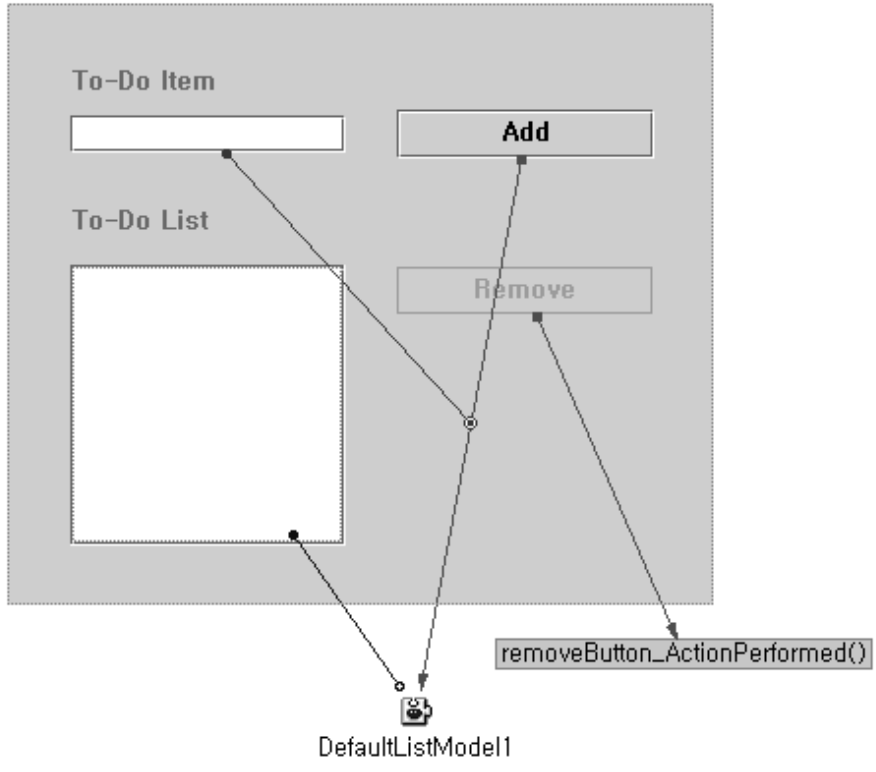
1. RemoveButton 빈을 마우스 오른쪽 단추로 클릭하십시오. 팝업 메뉴에서 **연결**을 선택한 다음 **actionPerformed**를 선택하십시오.
2. 빈 작업 영역에서 마우스 오른쪽 단추를 클릭하십시오. 팝업 메뉴에서 **이벤트와 코드 연결**을 선택하십시오. 이벤트와 코드 연결 창이 나타납니다.

   VisualAge for Java에서는 기본적으로 removeButton_ActionPerformed라는 새로운 메소드 스텁을 작성합니다.(어느 메소드가 어느 단추에 속하는지를 한눈에 알아볼 수 있도록 이 예제에서는 단추 이름을 바꾸었습니다.) 이 메소드는 컴포지트 빈 클래스(ToDoList)에 속합니다.
3. 메소드 스텁에 다음 코드를 복사하십시오. 하드카피 책에서 이 설명을 읽고 있다면 *시작하기 전에* 웹 버전에서 이 부분을 잘라 붙여넣기 해야 합니다.

```
public void removeButton_ActionPerformed(java.awt.event.ActionEvent actionEvent) {
        // Grab selected item
        Object itemToBeRemoved = getJList1().getSelectedValue();
        // Remove item from model
        getDefaultListModel1().removeElement(itemToBeRemoved);
        // Echo item to text field
        getJTextField1().setText(itemToBeRemoved.toString());
        // Refresh list from model
        getJList1().setModel(getDefaultListModel1());
        return;
}
```

   이 메소드는 VisualAge for Java에서 생성한 get 메소드를 호출하여 ToDoList에 삽입된 빈에 액세스합니다.
4. **OK**를 선택하여 창을 닫고 새로운 메소드를 저장하십시오.

연결은 이제 다음과 같습니다.



빠르고 간단하게 코드를 작성하도록 도와주는 VisualAge for Java IDE 기능에 대한 소개를 보려면 120 페이지의 『직접 코드 작성』으로 가십시오.

사용자 인터페이스를 완료했고 비주얼 빈 사이의 연결에 의해 정의된 애플릿 동작을 알아보았으므로 이제 작업을 저장 및 테스트할 준비가 되었습니다.

To-Do List 애플릿 저장 및 테스트

이미 작업중인 사용자 애플릿의 파트를 저장했습니다. 변경을 빈에 저장하면, 이전 빈 스펙을 새로운 것으로 대체하는 것입니다. 이를 실행할 때 VisualAge for Java는 빈의 새로운 활용 모두에 대해 새로운 스펙을 사용합니다. 빈에 대해 작업할 때 정기적으로, 그리고 편집을 완료했을 때 빈에 대한 변경사항을 저장하는 것이 바람직합니다.

비주얼 빈 저장

비주얼 컴포지션 편집기에서 다음을 실행하십시오.

- 빈을 저장하려면 **빈** 메뉴에서 **빈 저장**을 선택하십시오.

  사용자 빈이 저장되고 있으며 *런타임 코드*가 작성되고 있음을 알려주는 메시지 박스가 나타납니다. 생성된 이 런타임 코드는 사용자가 응용프로그램을 실행할 때 빈을 작성하는 데 사용합니다.

애플릿 테스트

작업을 저장했으므로, 이제 To-Do List 애플릿을 테스트할 수 있습니다.

1. 비주얼 컴포지션 편집기에서 애플릿을 테스트하려면 도구 막대에서 수행  을 선택하십시오. 애플릿 열람기는 그안에 있는 사용자 애플릿으로 시작합니다.
2. 애플릿이 나타나면, 사용자가 기대하는 방식으로 수행하는지 확인하면서 이를 실험해 보십시오. To-Do List 애플릿의 경우, 입력한 항목을 목록에 추가하고 목록에서 선택한 항목을 제거할 수 있는지 확인해야 합니다.

주: 애플릿을 설계할 때 Swing 빈은 Sun의 Metal 룩으로 표시됩니다. 예를 들어 JButton은 Windows JButton과 대조되는 Metal JButton으로 보입니다. 사용자 코드를 추가하여 빈이 런타임시 다른 시스템의 룩앤필(look and feel)로 표시되게 할 수 있습니다. 예를 들어 빈이 Windows 룩앤필을 가지도록 코드를 추가할 수 있습니다. 이러한 사용자 코드 추가에 대해서는 이 문서에서 다루지 않습니다.

테스트 완료시 애플릿 창을 닫았는지 확인하십시오.

언제든지 비주얼 컴포지션 편집기로 복귀해 변경할 수 있으며 변경을 저장하고, 애플릿을 다시 테스트할 수 있습니다.

축하합니다! 사용자의 To-Do List 애플릿이 완료되었습니다.

작업영역 저장

더 진행하기 전에 작업을 저장하십시오. 작업영역을 저장하면, 작업중인 모든 코드의 현재 상태와 현재 사용자가 연 창의 상태를 저장하는 것입니다. 작업영역을 저장하려면 **파일** 메뉴에서 **작업영역 저장**을 선택하십시오.

제5장 애플릿 상태 점검 추가 소개

사용자가 작성한 To-Do List 애플릿으로부터 미완성 비즈니스가 하나 남아있습니다. 되도록 애플릿을 단순하게 유지하려면, 모든 종류의 상태 점검을 포함하지 마십시오. **Add** 및 **Remove** 단추는 항상 사용할 수 있습니다. 이것은 애플릿의 이상적인 수행이 아닙니다. 예를 들어 To-Do List에서 선택한 항목이 있는 경우에는 **Remove** 단추만을 선택할 수 있어야 합니다.

이 절에서는 사용자 애플릿에 상태 점검을 추가하는 단계를 안내해 줍니다. 따라서 To-Do List 애플릿을 작성했을 때 배운 내용을 검토하고 비주얼 컴포지션 편집기가 작업하는 방식에 대해 약간 더 배울 기회입니다. 이 절에서는 **Remove** 단추에 대해서만 다루지만, 실험해 보고 싶다면 **Add** 단추의 몇가지 상태 점검을 추가해 볼 수 있습니다.

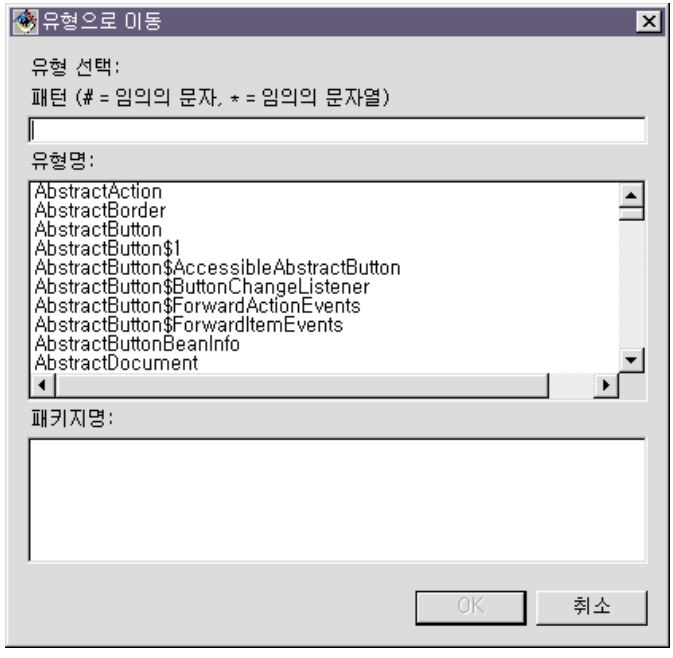
상태 점검을 포함하도록 애플릿을 갱신하기 전에, 애플릿을 찾아 그의 버전화된 개정판을 작성하는 단계를 거칩니다.

주: 이 절에서는 사용자가 15 페이지의 『제4장 첫번째 애플릿 빌드 소개』에서 설명한 단계를 완료했다고 가정합니다. 이제 완료된, To-Do List 애플릿을 가지게 됩니다. 아직 완료하지 않았다면, 기본 To-Do List 애플릿을 작성하는 단계를 완료하십시오.

Workbench에서 To-Do List 애플릿 찾기

상태 점검을 To-Do List 애플릿에 추가하려면, 먼저 이를 찾아야 할 수 있습니다. 비주얼 컴포지션 편집기에서 이를 작성할 때 사용자는 VisualAge for Java가 애플릿을 구현하기 위해 작성한 코드를 넣고 있는 트랙을 유지하지 않을 수 있습니다. 하지만 VisualAge for Java에서는 프로그램 구성요소를 찾는 강력한 검색 기능이 있으므로 크게 염려하지 않아도 됩니다. 이러한 기능은 탐색에 더 상세히 기술되어 있습니다. 지금으로선, 다음이 To-Do List 애플릿을 찾는 신속한 방법입니다.

1. Workbench 창에서 **프로젝트** 페이지를 선택하십시오.
2. **선택** 메뉴에서 **이동**을 선택한 다음 **유형**을 선택하십시오. 유형으로 이동 창이 나타납니다.



3. **패턴** 필드에서 `ToDoList` 같이 To-Do List 애플릿 이름을 입력하십시오. 이름을 입력하면, **유형명** 목록이 사용자가 지금까지 입력한 것과 일치하는 유형(즉, 클래스 및 인터페이스)만을 포함하도록 갱신됩니다.
4. **유형명** 목록에서 To-Do List 애플릿의 이름을 선택하십시오. 패키지가 **패키지명**에 나열된 경우, 둘 이상의 패키지에 사용자가 지정한 이름을 가진 클래스가 있음을 의미합니다. To-Do List 애플릿을 작성한 패키지를 선택하고 **OK**를 선택하십시오.
5. 프로젝트의 목록이 갱신됩니다. 사용자의 애플릿을 포함한 프로젝트와 패키지가 확장되며 사용자 애플릿의 클래스가 선택됩니다.

사용자 애플릿의 클래스를 찾았으므로, 이제 이를 버전화할 수 있습니다.

애플릿 개정판 버전화

프로그램 구성요소의 개정판을 *버전화*할 때, 이름을 제공하고 명시적으로 현재 상태를 저장하십시오. 코드를 더 변경하고 이러한 변경을 저장하면, 새로운 개정판이 버전화된 개정판에 기초하여 작성됩니다. 변경을 되돌리거나 다른 변경 세트를 시도할 경우, 단지 버전화된 개정판으로 리턴할 수 있습니다. 개정판 및 버전화된 개정판에 대해 더 자세한 내용은 사용자 코드의 개정판 관리를 참조하십시오.

To-Do List 애플릿 클래스를 변경하기 전에 필요하다면 리턴할 수 있도록 버전화하십시오. 사용자의 애플릿 클래스를 개정하려면, 다음을 실행하십시오.

1. 사용자 애플릿의 클래스가 선택되었는지 확인하십시오. **선택** 메뉴에서 **관리**를 선택한 다음 **버전**을 선택하십시오. 선택된 항목 개정 SmartGuide가 나타납니다.

2. **자동**이 선택되었는지 확인하고 **OK**를 클릭하십시오. **개정판 이름 보기** 

가 선택되어 있으면 버전 이름이 클래스 옆에 나타납니다.

다음에 이 클래스를 수정하고 이를 저장하면, VisualAge for Java가 이 버전화된 개정판의 코드에 기초한 새로운 개정판을 작성합니다. 애플릿으로 갱신하는 중에 문제에 부딪힐 경우, 방금 작성한 To-Do List 애플릿의 작업중인 개정판으로 리턴할 수 있습니다.

[엔터프라이즈] Enterprise edition에서는 개정판을 개발자 팀이 관리합니다. 개정판 소유자만이 버전화할 수 있습니다. 또한 버전화 다음에는 릴리스화가 자주 수행됩니다. 팀 개발 환경에 대해 자세히 알려면, VisualAge for Java, Enterprise Edition에 대한 온라인 도움말이나 *시작하기 전에* 서적을 참조하십시오.

애플릿 상태 점검 추가

사용자의 애플릿 개정판을 개정했으므로, 이제 상태 점검을 추가할 준비가 되었습니다.

Remove 단추의 이상적인 동작

현재 목록에 항목이 없는 경우에도, **Remove** 단추는 항상 사용가능합니다. 다음은 **Remove** 단추의 작동 방식입니다.

- 애플릿 시작시 **Remove** 단추는 사용불가능해야 합니다.
- To-Do List에서 항목을 선택하면 **Remove** 단추가 사용가능해야 합니다.
- To-Do List에서 선택한 항목을 삭제하면 **Remove** 단추가 사용불가능으로 되어야 합니다.

Remove 단추의 원하는 동작을 얻고 싶으면 다음을 수행해야 합니다.

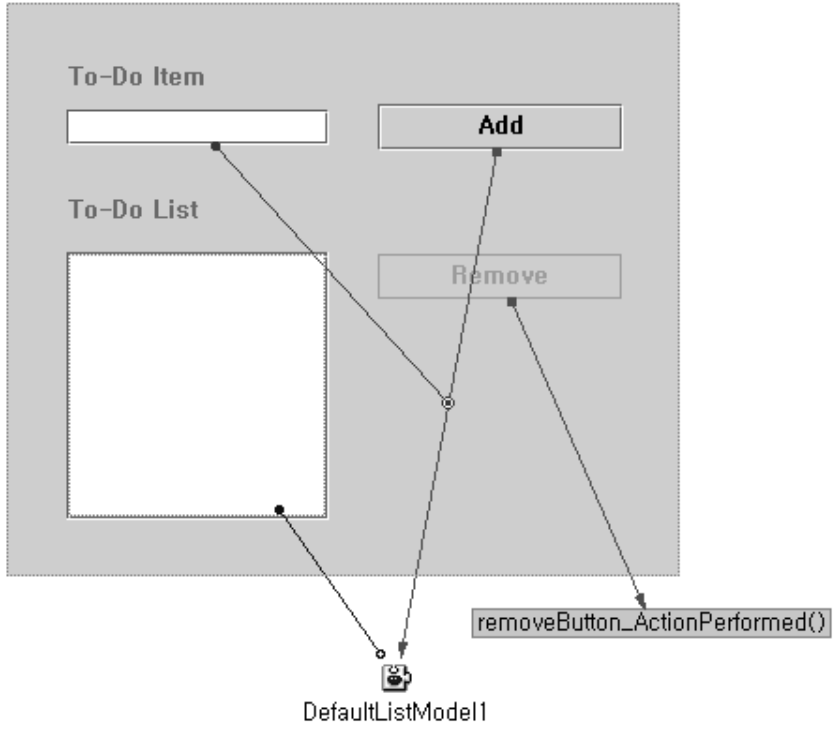
- 비주얼 컴포지션 편집기에서 To-Do List 애플릿을 여십시오.
- 애플릿을 처음 시작하면 **Remove** 단추를 사용할 수 없도록 단추 특성을 설정하십시오.
- To-Do List에서 항목을 선택하면 단추를 사용할 수 있도록 이벤트와 코드 연결을 추가하십시오.

비주얼 컴포지션 편집기에서 To-Do List 애플릿 열기

먼저 비주얼 컴포지션 편집기에서 다음과 같이 To-Do List 애플릿 클래스를 여십시오.

1. Workbench에서 사용자의 To-Do List 애플릿의 클래스를 선택하십시오.
2. **선택** 메뉴에서 **선택 열기**를 선택한 다음 **비주얼 컴포지션**을 선택하십시오.

3. 빈 작업 영역이 나타납니다. 빈 작업 영역은 다음과 같이 나타나야 합니다.



Remove 단추 특성 설정

이제 애플릿 시작시 사용불가능하게 **Remove** 단추의 특성을 설정하십시오.

1. RemoveButton을 선택하고 마우스 오른쪽 단추를 클릭하십시오. 나타나는 팝업 메뉴에서 **특성**을 선택하십시오. 특성 창이 나타납니다.

2. **enabled**의 오른쪽 필드를 선택하십시오. 필드의 드롭다운 목록에서 **False**를 선택한 다음 특성 창을 닫으십시오. **Remove** 단추가 사용불가능으로 표시됩니다.

Remove 단추 사용가능 및 사용불가능화하는 연결 추가

To-Do List에서 항목을 선택하면 **Remove** 단추를 사용가능으로 만드는 연결을 추가하십시오.

1. 비주얼 컴포지션 페이지에서 JList 빈을 선택한 다음 마우스 오른쪽 단추를 클릭하십시오.
2. 팝업 메뉴에서 **연결, 연결가능 기능**을 선택하십시오. 연결 창이 열립니다.
3. **이벤트** 목록에서 **listSelectionEvents**를 선택하십시오. 마우스 포인터가 변하여 연결 중임을 나타냅니다.
4. 빈 작업 영역을 마우스 왼쪽 단추로 클릭하고 **코딩할 이벤트**를 선택하여 연결을 완성하십시오.

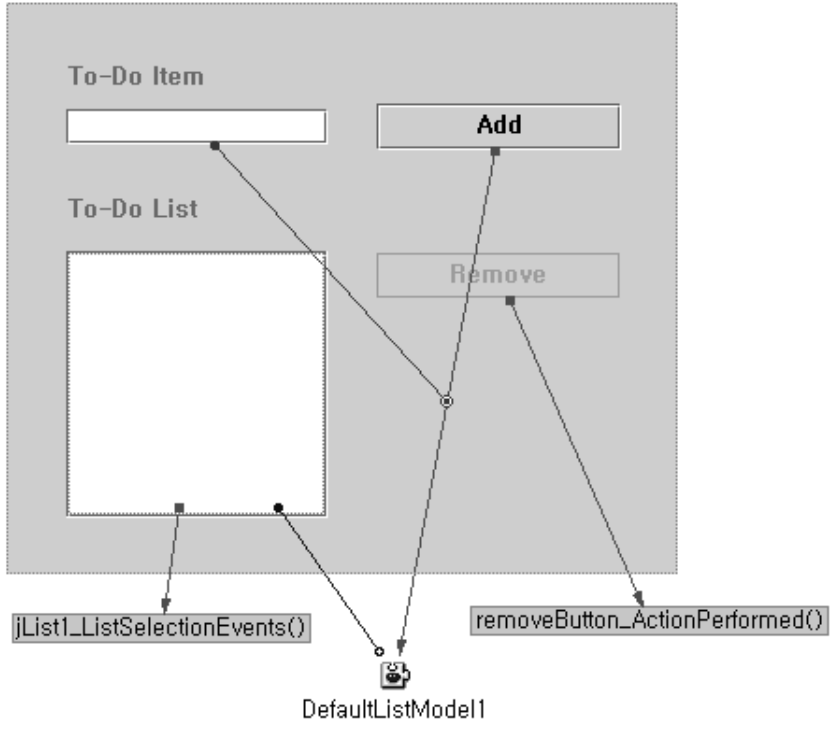
이벤트와 코드 연결 창이 나타납니다. VisualAge for Java에서는 기본적으로 jList1_ListSelectionEvents()라는 새로운 메소드 스텁을 작성합니다.

5. 메소드 스텁에 다음 코드를 복사하십시오.

```
public void jList1_ListSelectionEvents() {
        if (getJList1().getSelectedIndex() < 0)
                getRemoveButton().setEnabled(false);
        else
                getRemoveButton().setEnabled(true);
        return;
}
```

이 메소드는 JList1의 getSelectedIndex() 메소드를 호출합니다. 메소드가 -1을 리턴하면 목록에서 선택한 항목이 없고 RemoveButton의 *enabled* 특성이 false로 설정되어 있습니다. -1 외 다른 값을 리턴하면 *enabled* 특성이 true로 설정되어 있습니다.

6. **OK**를 클릭하여 창을 닫으십시오. To-Do List는 다음과 같이 나타나야 합니다.



변경 사항 저장 및 테스트

더 진행하기 전에, 작업을 저장하고 테스트하십시오.

1. 비주얼 컴포지션 편집기에서 수행한 작업의 현재 상태를 저장하려면 **빈** 메뉴에서 **빈 저장**을 선택하십시오.

2. 변경 사항을 테스트하려면 도구 막대에서 수행  을 선택하십시오.
3. 사용자 애플릿에 대한 애플릿 열람기가 나타납니다.
4. 애플릿에 여러 가지 실험을 해서 **Remove** 단추가 올바르게 동작하는지 확인하십시오. 애플릿 시작시 **Remove** 단추가 사용불가능해지고 To-Do List에서

항목이 선택된 즉시 사용가능해지는지 확인하십시오. To-Do List에서 항목이 선택되지 않은 경우 **Remove** 단추가 다시 사용불가능해지는지 확인하십시오.

축하합니다! To-Do List 애플릿에 상태 점검을 성공적으로 추가했습니다.

이제 사용자가 작성한 코드의 새로운 레벨이 작동하므로 39 페이지의 『애플릿 개정판 버전화』에서 설명하는 단계를 따라 다른 버전화된 개정판을 작성하십시오.

제6장 To-Do List 프로그램 향상 소개

앞의 절에서 사용자의 간단한 To-Do List 애플릿에 상태 점검을 추가했습니다. 이 절에서는 To-Do List를 명명된 파일에 저장하고 To-Do List를 포함한 파일을 열 수 있도록 간단한 To-Do List 애플릿을 수정해 봅니다.

애플릿을 수정할 때 다음을 배우게 됩니다.

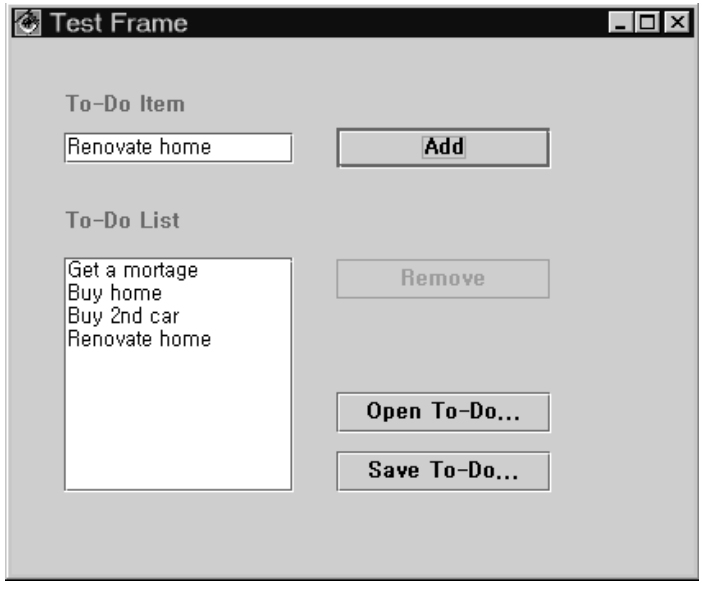
- 새로운 메소드 작성
- 비주얼 컴포지션 편집기에서 비즈니스용 논리 코드 추가
- 사용자 인터페이스 갱신
- 애플릿 또는 응용프로그램으로서 코드 수행

주: 이 절에서는 사용자가 애플릿에 상태 점검 추가 절에서 설명한 단계를 완료했다고 가정합니다. 이제 간단한 상태 점검을 포함한 완료된 To-Do List 애플릿을 가지게 됩니다. 아직 완료하지 않았다면, 사용자의 To-Do List 애플릿에 상태 점검 추가 단계를 완료하십시오.

향상된 To-Do List 프로그램 작동 방식

사용자의 애플릿에 가한 수정으로 건너뛰어 갱신된 To-Do List 프로그램을 작성하기 전에, 완성된 프로그램의 작동 방식을 검토해봅시다.

다음은 To-Do List 프로그램을 설명한 것입니다.



기존 애플릿과 마찬가지로, 갱신된 To-Do List 프로그램은 **Add** 단추 선택시 **To-Do Item** 필드 내의 텍스트를 **To-Do List**에 추가합니다. **Remove** 단추 선택시, 프로그램이 To-Do List로부터 선택된 항목을 제거합니다.

새로운 단추는 어떻게 됩니까? 다음은 그 수행의 개요입니다.

- **Open To-Do File**을 선택하면 목록 파일 내용이 To-Do List 프로그램으로 로드됩니다.
- **Save To-Do File**을 선택하면 To-Do List 내용이 목록 파일에 복사됩니다.

인터페이스와 동작에서의 차이점 외에, To-Do List 애플릿과 To-Do List 프로그램 사이의 중요한 차이점이 하나 더 있습니다. 파일 시스템에 액세스하여 파일을 읽고 써야하기 때문에, To-Do List 프로그램이 애플릿보다는 응용프로그램으로서 수행되어야 합니다. Java 애플릿에는 파일 시스템에 대한 액세스가 허용되지 않습니다.

향상된 To-Do List 프로그램을 작성하려면 다음 단계를 따르십시오.

1. 49 페이지의『새로운 메소드 작성』
2. 55 페이지의『To-Do List 프로그램 사용자 인터페이스에 단추 추가』

3. 57 페이지의『Open To-Do File 단추 연결』
4. 58 페이지의『Open To-Do File 단추 테스트』
5. 60 페이지의『Save To-Do File 단추 연결』
6. 61 페이지의『완료된 To-Do List 프로그램 저장 및 테스트』

다음 절에서 이에 대해 상세히 기술합니다.

새로운 메소드 작성

To-Do List 프로그램을 향상시키는 다음 단계는 To-Do List 프로그램에 대한 논리를 포함하는 몇몇 메소드를 추가하여 To-Do File을 읽고 쓰는 것입니다.

다음은 새로운 클래스 작성을 위해 완료해야 하는 개별 타스크입니다.

- 새로운 메소드에 필요한 클래스를 반입하십시오.
- 새로운 메소드에 필요한 static 변수를 추가하십시오.
- 50 페이지의『파일 읽기용 메소드 추가』
- 52 페이지의『파일 쓰기용 메소드 추가』

다음 절에서 이러한 타스크에 대해 더 상세히 설명합니다.

새로운 메소드에 필요한 클래스 반입

새로운 메소드에는 com.sun.java.swing.* 클래스와 java.io.* 클래스를 반입해야 합니다. 이 클래스를 반입하려면 다음을 수행하십시오.

1. Workbench에서 ToDoList 클래스를 선택하십시오.
2. 소스 분할영역에서 기존 반입 명령문 아래 java.applet.*과 java.awt.* 클래스를 반입하는 명령문을 추가하십시오.

```
import com.sun.java.swing.*;
import java.io.*;
```

새로운 메소드에 필요한 static 변수 추가

새로운 메소드에는 To-Do List가 저장될 파일 이름을 보유하는 static 변수가 필요합니다. 다음과 같이 이 변수를 추가하십시오.

1. Workbench에서 ToDoList 클래스를 선택하십시오.
2. 소스 분할영역에서 클래스에 대해 정의된 필드 아래 다음 명령문을 추가하십시오.

```
static String FILE_NAME = "todo.lst";
```

변경 사항을 저장하십시오. 소스 분할영역에서 마우스 오른쪽 단추를 클릭하십시오. **저장**을 클릭하십시오.

파일 읽기용 메소드 추가

이 절에서는 입력 파일을 읽는 *readToDoFile*이라는 메소드를 작성하는 방법에 대해 설명합니다. 이 메소드는 목록 파일 내용을 행 단위로 읽어 각 행을 DefaultListModel1 인스턴스에 추가합니다.

1. ToDoList 클래스를 선택하십시오.
2. **선택** 메뉴에서 **추가**를 선택한 다음 **메소드**를 선택하십시오. 메소드 작성 SmartGuide가 나타나면 메소드 이름에서 다음을 입력하십시오.

```
void readToDoFile()
```

3. **완료**를 선택하여 메소드를 생성하십시오.
4. 새로운 메소드를 선택하고 메소드를 구현하는 코드를 추가하십시오. 검색기에서 이 문서를 열람하고 있다면, 다음 코드를 선택하고 이를 복사하여 **소스** 분할영역에 붙이십시오. 종료된 메소드는 다음과 같습니다.

```
public void readToDoFile() {
FileReader fileInStream;
BufferedReader dataInStream;
String result;
try {
   // read the file and fill the list
   fileInStream = new FileReader(FILE_NAME);
   dataInStream = new BufferedReader(fileInStream);
// clear the existing entries from the list
getDefaultListModel1().removeAllElements();
// for each line in the file create an item in the list
while ((result = dataInStream.readLine()) != null) {
   if (result.length() != 0)
      getDefaultListModel1().addElement(result);
      }
fileInStream.close();
```

```
        dataInStream.close();
    } catch (Throwable exc) {
    handleException(exc);
    }
    return;
}
```

5. **편집** 메뉴에서 **저장**을 선택하여 변경을 저장하고 재컴파일하십시오.

다음 타스크를 계속하기 전에, 이 메소드 내의 코드를 검토해 보겠습니다.

1. 메소드 시작 부분에는 파일과 그 내용을 조작하는 데 사용하는 필드 선언이 있습니다.

```
FileReader fileInStream;
BufferedReader dataInStream;
String result;
```

2. 다음으로 명령문을 FileReader가 있는 파일과 연관시키고 FileReader를 BufferedReader에 연관시키십시오. BufferedReader를 사용하면 한번에 한 행씩 파일을 읽을 수 있습니다. static FILE_NAME 변수는 앞에서 정의했습니다.

```
try {
   // read the file and fill the list
   fileInStream = new FileReader(FILE_NAME);
   dataInStream = new BufferedReader(fileInStream);
```

3. 다음으로 목록을 지웁니다. 그러면 루프가 파일을 한 번에 한 행씩 문자열 result로 읽어 들입니다. result가 0 길이 문자열이 아니면 result 값이 목록 모델에 추가됩니다.

```
// clear the existing entries from the list
getDefaultListModel1().removeAllElements();
// for each line in the file create an item in the list
while ((result = dataInStream.readLine()) != null) {
   if (result.length() != 0)
      getDefaultListModel1().addElement(result);
}
```

4. 시행 블록 끝에서 명령문은 파일에 연관된 스트림을 닫습니다.

```
fileInStream.close();
dataInStream.close();}
```

5. 캐치 블록은 예외를 *handleException()*으로 전달합니다. VisualAge for Java는 시각적으로 구성된 모든 클래스에 대해 이 메소드를 생성합니다. 주석 분리자가 제거됩니다.

```
catch (Throwable exc) {
   handleException(exc);
   }
}
...
handleException(Throwable exc) {
/* Uncomment the following lines to print uncaught exceptions to stdout */
        System.out.println("———- UNCAUGHT EXCEPTION ———-");
        exception.printStackTrace(System.out);
}
```

파일 쓰기용 메소드 추가

ToDoList 클래스, *writeToDoFile()*에 추가할 메소드가 두 개 이상 있습니다. 이 메소드는 목록 모델 내용을 목록 파일에 하나씩 기록합니다.

1. ToDoList 클래스를 선택하십시오.
2. **선택** 메뉴에서 **추가**를 선택한 다음 **메소드**를 선택하십시오. 메소드 작성 SmartGuide가 나타나면 메소드 이름에서 다음을 입력하십시오.

   ```
   void writeToDoFile()
   ```

3. **완료**를 선택하여 메소드를 생성하십시오.
4. 새로운 *writeToDoFile()* 메소드를 선택하고 이를 구현할 코드를 추가하십시오. 검색기에서 이 문서를 열람하고 있다면, 다음 코드를 선택하고 이를 복사하여 소스 분할영역에 붙여넣으십시오. 종료된 메소드는 다음과 같습니다.

   ```
   public void writeToDoFile() {
      FileWriter fileOutStream;
      PrintWriter dataOutStream;
      // carriage return and line feed constant
   String crlf = System.getProperties().getProperty("line.separator");
      // write the file from the list
      try {
         fileOutStream = new FileWriter(FILE_NAME);
         dataOutStream = new PrintWriter(fileOutStream);
   // for every item in the list, write a line to the output file
   for (int i = 0; i < getDefaultListModel1().size(); i++)
      dataOutStream.write(getDefaultListModel1().getElementAt(i) + crlf);
      fileOutStream.close();
      dataOutStream.close();
      } catch (Throwable exc) {
          handleException(exc);
      }
      return;
   }
   ```

5. **편집** 메뉴에서 **저장**을 선택하여 변경을 저장하고 재컴파일하십시오.

이 코드는 *readToDoFile()*의 코드와 비슷합니다. 다음 단계로 나아가기 전에, 실제로 행을 파일에 작성하는 루프를 다음에서 검토해봅시다.

```
// for every item in the list, write a line to the output file
for (int i = 0; i < getDefaultListModel1().size(); i++)
   dataOutStream.write(getDefaultListModel1().getElementAt(i) + crlf);
```

이 루프는 목록 모델의 각 항목을 통과합니다. 각 항목에는 `crlf`(행 구분 기호로 구성되는 문자열)가 추가되고 파일에 기록됩니다. 행 분리자는 각 항목이 파일 내 독립 행에 쓰여지도록 강제실행합니다.

스크랩북을 사용하여 코드 테스트

계속하기 전에 행 분리자에 대해 잠깐 알아봅시다. 사용자가 전에 이를 본 적이 없어서 그 작동 방식을 알고 싶어한다고 가정해 봅시다. 스크랩북 창을 사용하여 사용자 클래스의 이 파트를 연습하는 코드 일부를 테스트할 수 있습니다.

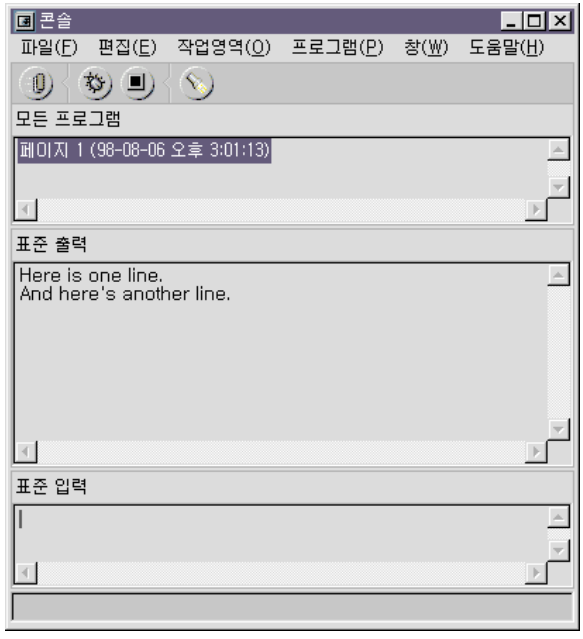
행 분리자 코드를 테스트하려면, 다음을 실행하십시오.

1. **창** 메뉴에서 **스크랩북**을 선택하십시오. 그러면 스크랩북 창이 나타납니다.
2. 다음 코드를 스크랩북 창 내 페이지에 입력하십시오.

   ```
   String crlf = System.getProperties().getProperty("line.separator");
   System.out.println("Here is one line."+crlf+"And here's another line.");
   ```

3. 코드 행을 둘다 선택하고 스크랩북 창 도구 막대에서 수행  을 선택하십시오.

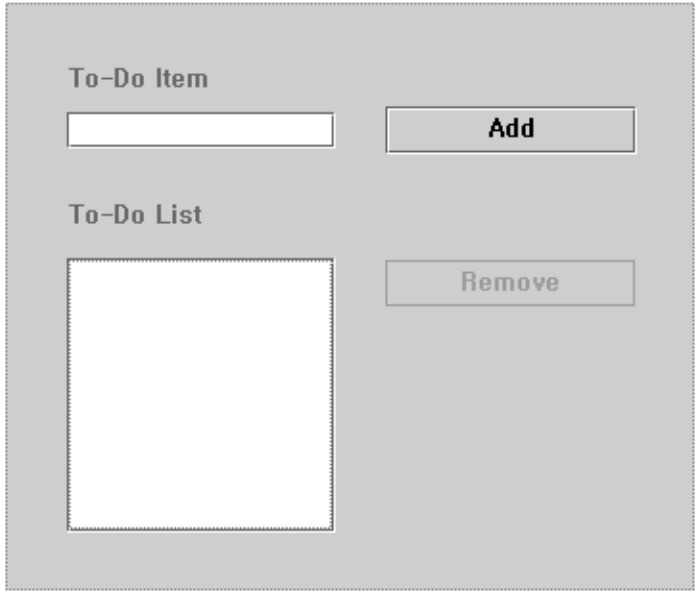
4. **창** 메뉴에서 **콘솔**을 선택하십시오. 콘솔 창은 다음과 같이 나타나야 합니다.



출력이 독립 행에 나타나도록 행 분리자가 출력을 나누는 것에 주의하십시오. 이 간단한 예제는 스크랩북 창을 사용하여 즉시 그리고 편리하게 코드 조각을 시험해 볼 수 있는 방식을 보여 줍니다.

To-Do List 프로그램 사용자 인터페이스에 단추 추가

ToDoList 클래스에 논리를 추가하는 모든 단계를 완료했습니다. 이제 To-Do List 애플릿의 사용자 인터페이스를 수정할 준비가 되었습니다. 현재 To-Do List 애플릿은 다음과 같이 나타나야 합니다.



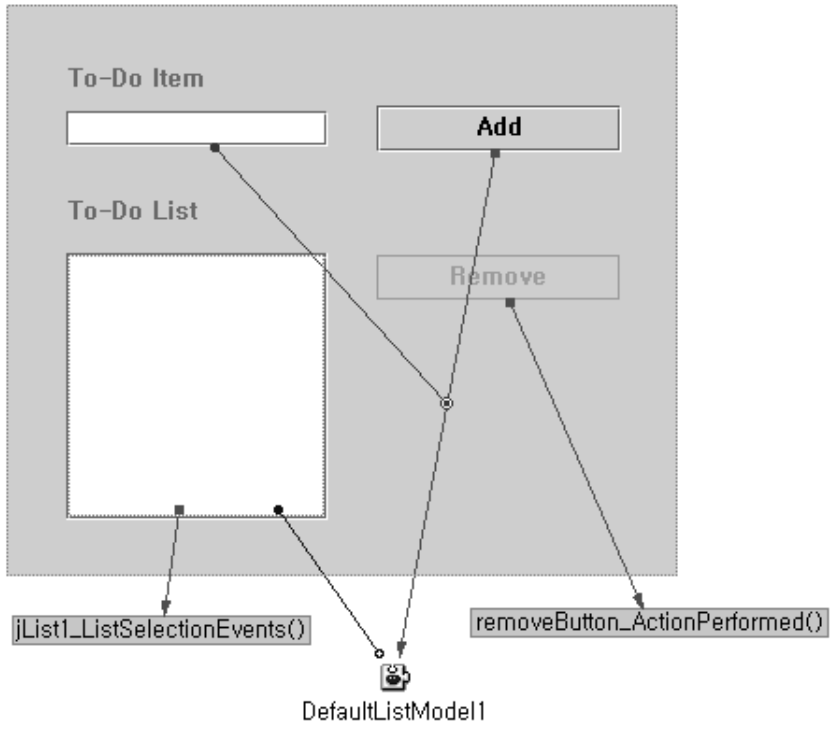
이 사용자 인터페이스에 새로운 두 단추를 추가해야 합니다.

- To-Do List에 읽기 위해 파일 열기를 시작하는 **Open To-Do File** 단추
- 파일에 To-Do List의 내용 저장을 시작하는 **Save To-Do File** 단추

이러한 두 단추를 추가하려면, 다음을 실행하십시오.

1. Workbench에서 To-Do List 애플릿에 대한 ToDoList 클래스를 선택하십시오.
2. **선택** 메뉴에서 **선택 열기**를 선택한 다음 **비주얼 컴포지션**을 선택하십시오.

3. 빈 작업 영역이 나타납니다. 빈 작업 영역은 다음과 같이 나타나야 합니다.



4. JButton 빈을 선택하고 단추를 기존 **Remove** 단추 아래에 추가한 다음 ReadButton으로 명명하십시오. **Add** 단추와 **Remove** 단추를 옮기거나 빈 작업 영역의 길이를 늘여 새로 만든 단추를 위한 공간을 만드십시오.
5. 방금 추가한 단추를 선택한 다음 단추의 텍스트를 **Open To-Do File**로 변경하십시오. 텍스트를 변경하려면 다음을 수행하십시오.
   - ReadButton에 대한 **특성**을 여십시오.
   - *text*의 값을 `Open To-Do File`로 변경하십시오.
6. 같은 절차에 따라 ReadButton 아래 다른 단추를 추가하십시오. 단추 이름을 SaveButton으로 명명하고 단추의 텍스트는 `Save To-Do File`로 변경하십시오.
7. 새로운 단추의 가로가 기존 단추의 가로와 일치하도록 *크기*를 조절하십시오.

- AddButton을 선택하십시오. Ctrl 키를 누른 채로 나머지 단추를 선택하여 단추 네 개를 모두 선택하십시오. 선택한 마지막 빈에는 *앵커 빈* 임을 나타내는 단일 선택 핸들이 있습니다. 앵커 빈은 *크기 재조정*의 안내자로 역할하는 빈 또는 다른 선택된 빈과 일치하는 빈입니다.

- 도구 막대에서 **폭 일치**  를 선택하십시오.

8. 새로운 두 단추를 기존 단추와 정렬하십시오.
   - SaveButton을 선택하십시오. Ctrl 키를 누른 채로 ReadButton, RemoveButton 및 AddButton을 선택한 단추 네 개를 모두 선택하십시오. AddButton이 앵커 빈이 됩니다.

   - 도구 막대에서 **왼쪽 정렬**  을 선택하십시오.
9. 네개의 단추를 모두 균등하게 분배하려면 다음을 수행하십시오.

   - 단추를 모두 선택했으므로 도구 막대에서 **수직 분배**  를 선택하십시오.

To-Do List 프로그램용으로 두 개의 새로운 단추를 추가했습니다. 이제 몇 가지 조치와 이 단추를 연관시킬 준비가 되었습니다.

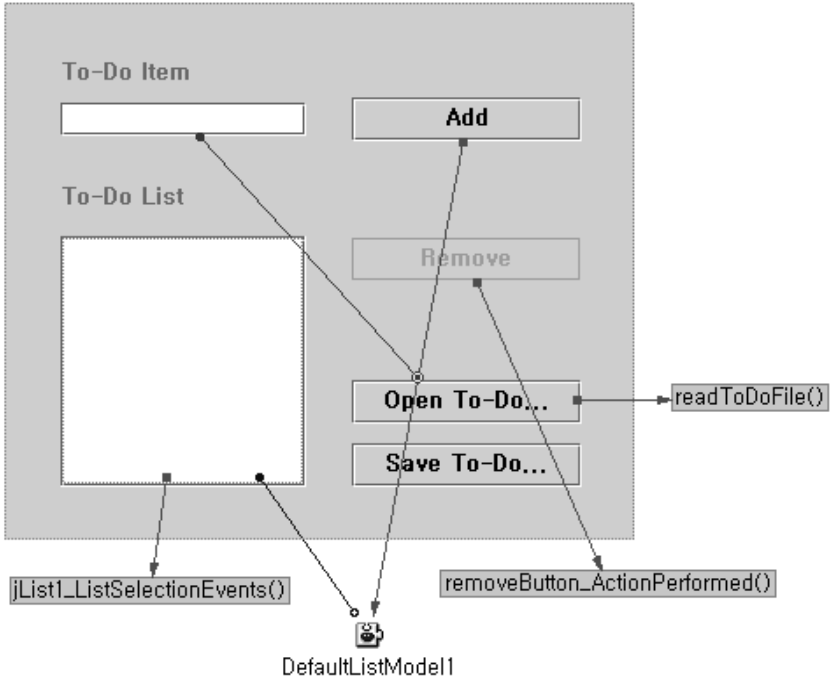
Open To-Do File 단추 연결

새로운 빈을 빈 작업 영역에 모두 추가했으므로, 이제 연결을 시작할 준비가 되었습니다.

To-Do List 애플릿은 사용자가 **Open To-Do File** 단추를 선택하면 *readToDoFile()* 메소드를 호출해야 합니다. 이를 구현하려면 ReadButton을 *readToDoFile()* 메소드에 연결해야 합니다.

1. ReadButton을 선택하고 마우스 오른쪽 단추를 클릭하십시오.
2. 팝업 메뉴에서 **연결**을 선택한 다음 **actionPerformed**를 선택하십시오.
3. 빈 작업 영역을 마우스 왼쪽 단추로 클릭하고 **코딩할 이벤트**를 선택하십시오.

4. 이벤트와 메소드 연결 창의 **메소드** 목록에서 **readToDoFile()**을 선택한 다음 **OK**를 선택하십시오. 빈 작업 영역은 다음과 같이 나타나야 합니다.



5. **빈** 메뉴에서 **빈 저장**을 선택하여 비주얼 컴포지션 편집기에 현재 작업을 저장하십시오. VisualAge for Java는 사용자가 지정한 연결을 구현하기 위해 코드를 생성합니다.

축하합니다! **Open To-Do File** 단추를 구현했습니다. 이제 지금까지 수행한 작업을 To-Do List 프로그램에서 테스트할 준비가 되었습니다.

Open To-Do File 단추 테스트

Open To-Do File 단추에 대한 연결을 모두 작성했으므로, 이제 지금까지 수행한 작업을 테스트할 준비가 되었습니다.

To-Do List 프로그램의 현재 상태를 테스트하려면 다음을 수행하십시오.

1. 테스트에 사용할 간단한 텍스트 파일을 준비하십시오. 스크랩북 창을 사용하여 다음 행이 있는 todo.list 파일을 작성하십시오.

```
Get a mortgage
Buy home
Buy 2nd car
Renovate home
Ask for a raise
```

   project_resources\My ToDoList Project 디렉토리에 파일을 저장하십시오.

2. 메뉴 막대에서 **수행**을 선택한 다음 **main 수행**을 선택하십시오. To-Do List 프로그램이 나타납니다.
3. **Open To-Do File** 단추를 선택하십시오. 사용자 프로그램에 있는 To-Do List가 todo.lst 파일의 항목과 함께 로드됩니다.

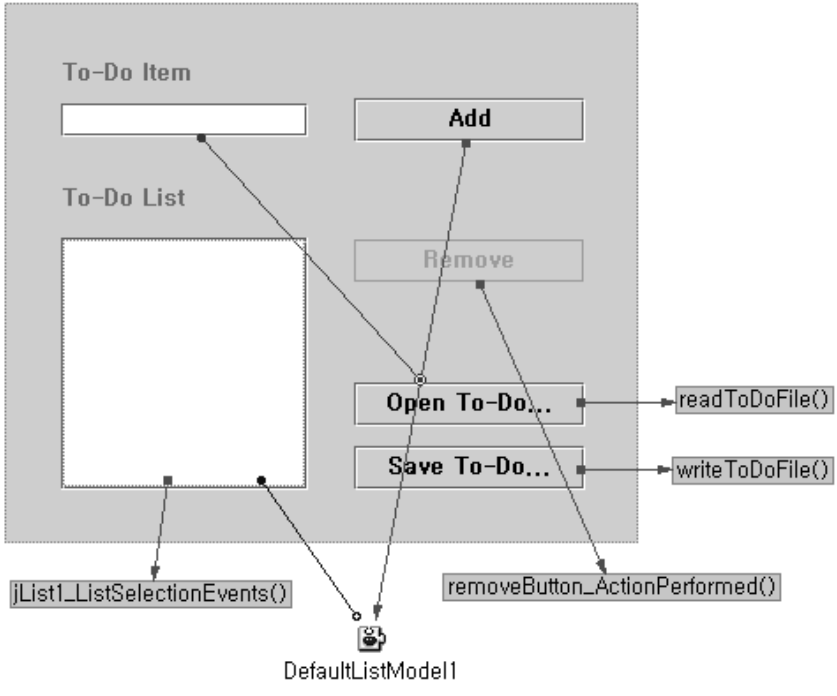
To-Do List 프로그램에서 현재 진행을 테스트했으므로, 이제 **Save To-Do File** 단추로부터의 연결을 작성하여 프로그램을 완료할 준비가 되었습니다.

Save To-Do File 단추 연결

Save To-Do File 단추에서 최종 연결을 작성할 준비가 되었습니다.

사용자가 **Save To-Do File** 단추를 선택하면 To-Do List 애플릿은 *writeToDoFile()* 메소드를 호출해야 합니다. 이를 구현하려면 SaveButton을 *writeToDoFile()* 메소드에 연결해야 합니다.

1. SaveButton을 선택하고 마우스 오른쪽 단추를 클릭하십시오.
2. 팝업 메뉴에서 **연결**을 선택한 다음 **actionPerformed**를 선택하십시오.
3. 마우스 오른쪽 단추로 빈 작업 영역을 클릭하십시오. 팝업 메뉴에서 **이벤트와 코드 연결**을 선택하십시오.
4. 이벤트와 코드 연결 창에서 **writeToDoFile()** 메소드를 선택하십시오. 빈 작업 영역은 다음과 같이 나타나야 합니다.



축하합니다! **Save To-Do File** 단추로부터 연결을 완료했습니다. 사용자의 To-Do File 프로그램이 완료되었고 이제 이를 테스트할 준비가 되었습니다.

완료된 To-Do List 프로그램 저장 및 테스트

To-Do List 프로그램을 완성했으므로, 이제 이를 저장 및 테스트할 수 있습니다.

완료된 To-Do List 프로그램을 저장 및 테스트하려면 다음을 수행하십시오.

1. **빈** 메뉴에서 **빈 저장**을 선택하여 변경을 저장하십시오. VisualAge for Java가 마지막으로 저장한 이후로 비주얼 컴포지션 편집기에서 사용자가 실행한 모든 작업을 구현하는 코드를 작성합니다.
2. 메뉴 막대에서 **수행**을 선택한 다음 **main 수행**을 선택하십시오.
3. 새로운 To-Do File을 작성 및 저장해보십시오.
   - To-Do List에 다음 항목을 추가하십시오. 각 항목에 대해, **To-Do Item** 필드에 항목을 입력하고 **추가**를 선택하십시오.

     ```
     Get paint
     Get wallpaper
     Spouse says OK?
     Start painting
     Start wallpapering
     ```

   - **Save To-Do File**을 선택하십시오.
4. 프로그램을 닫고 다시 실행하십시오.
5. **Open To-Do File**을 선택하십시오. 이전에 입력한 목록이 창에 나타납니다.

축하합니다! 비주얼 컴포지션 편집기에서 작성된 사용자 인터페이스를 사용자가 직접 작성한 넌-비주얼 코드와 결합하는 Java 프로그램을 완성했습니다.

계속하기 전에, ToDoList 클래스의 버전화된 개정판을 작성하십시오.

1. Workbench에서 ToDoList 클래스를 선택하십시오.
2. **선택** 메뉴에서 **관리**를 선택한 다음 **버전**을 선택하십시오. 선택된 항목 개정 SmartGuide가 나타납니다.
3. **자동**이 선택되었는지 확인하고 **완료**를 선택하십시오.

제7장 비주얼 컴포지션 편집기에서 할 수 있는 기타 작업 소개

15 페이지의 『제4장 첫번째 애플릿 빌드 소개』에서 비주얼 컴포지션 편집기의 빈 팔레트, 도구 막대, 빈 작업 영역을 사용하여 사용자 인터페이스를 구성하는 방법을 배웠습니다. 이러한 기본 기술로 작성하려면, 빈과 그 특성 조작 방법, 연결과 그 특성에 대한 작업 방법 및 실수 정정 방법에 대해 알아야 합니다.

이 절을 읽다보면, 새로운 애플릿을 작성하고, 그 위에서 비주얼 컴포지션 편집기를 열고, 기술된 타스크 중 일부를 시험해보고자 할 수 있습니다.

빈 조작

빈을 애플릿에 추가한 후, 보통 이들을 정렬시키거나 *크기조정*하거나 또는 비슷한 타스크를 수행하려고 합니다. 그런데 빈을 정렬 또는 *크기* 조정하려면, 먼저 이들의 조작 방법을 알아야 합니다. 이 절에서는 다음 타스크를 소개합니다.

- 『빈 선택』
- 65 페이지의 『빈 선택 취소』
- 65 페이지의 『빈 이동』
- 65 페이지의 『빈 복사』
- 66 페이지의 『빈 삭제』

빈 선택

빈을 선택하려면, 마우스 왼쪽 단추로 이를 클릭하십시오.

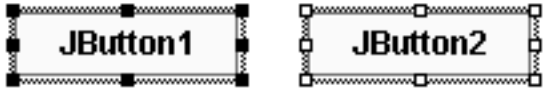
빈을 선택할 때 빈 모서리에 *선택 핸들*이라는 조그만 상자가 표시됩니다. 이 상자를 이용하면 빈을 쉽게 조작할 수 있습니다.

크기 조정할 수 없는 빈에는 선택 조정이 들어 있지 않습니다. 대신, 이러한 빈은 선택되었을 때 자신의 백그라운드 색을 변경합니다. 이러한 수행을 가진 빈에는 넌-비주얼 빈과 메뉴 빈이 포함됩니다.

사용자가 빈을 선택했을 때 기타 빈도 선택되어 있었다면, 이들의 선택은 자동적으로 취소됩니다. *단일 선택*으로 참조됩니다. 현재 선택된 빈 이름은 비주얼 컴포지션 편집기 아래쪽 상태 영역에 표시됩니다.

여러 빈 선택

여러 개의 빈이 선택되었다면, 마지막으로 선택된 빈에 *앵커* 빈임을 나타내는 단일 선택 핸들이 있습니다. 선택한 다른 빈에는 오목 선택 핸들이 표시됩니다.



크기조정 및 정렬과 같은 작동 수행시 앵커 빈이 중요합니다. 기타 선택된 빈은 자신의 위치 또는 *크기*를 앵커 빈의 위치 또는 *크기*로 설정합니다. Shift 키를 누른 채로 앵커가 되기 원하는 빈을 선택하여 앵커 빈을 변경할 수 있습니다.

빈을 여러 개 선택하려면 다음 중 **하나**를 수행하십시오.

- 선택하고 싶은 빈 중 하나에서 마우스 왼쪽 단추를 클릭한 후 Ctrl 키를 누른 채 선택하고 싶은 각 추가 빈에서 마우스 왼쪽 단추를 클릭하십시오. 기억해 둘 점은 마지막 Select 빈이 *크기조정* 및 정렬 작동이 발생하는 앵커(anchor)가 됩니다.
- UNIX 플랫폼과 OS/2에서는 빈을 클릭하여 마우스 왼쪽 단추를 누르고 있을 수 있습니다. 선택하고 싶은 각 추가 빈 위로 마우스 포인터를 이동하십시오. 원하는 모든 빈을 선택한 후 마우스 왼쪽 단추에서 손을 떼십시오.

빈을 여러 개 선택하는 경우에는 상태 영역에 선택한 빈 수가 표시됩니다.
(예: 3개의 빈이 선택되었습니다.)

빈 선택 취소

현재 선택되어 있는 빈을 모두 선택 취소하려면, 다른 빈에서 또는 빈 작업 영역의 개방 영역에서 마우스 왼쪽 단추를 클릭하십시오.

선택된 빈 그룹에서 한 빈을 선택 취소하려면, Ctrl 키를 누른 채 선택 취소하고 싶은 빈에서 마우스 왼쪽 단추를 클릭하십시오. 선택한 빈이 앵커(anchor) 빈일 경우, 이전에 선택한 빈이 앵커(anchor) 빈이 됩니다.

빈 이동

빈을 이동하려면, 다음 단계를 따르십시오.

1. 빈을 적절한 마우스 단추로 클릭하고 잠시 누르고 있으십시오.
   - OS/2에서는 마우스 오른쪽 단추를 누른 채로 있습니다.
   - UNIX 플랫폼에서는 마우스 가운데 단추를 누른 채로 있습니다.
   - Windows에서는 마우스 왼쪽 단추를 누른 채로 있습니다.
2. 빈을 넣고 싶은 위치로 마우스 포인터를 이동한 후 마우스 단추에서 손을 떼십시오.

이동하고 싶은 모든 빈을 선택하여 몇개의 빈을 한번에 이동할 수 있습니다. 그런 후 선택된 빈을 잡고(마우스 왼쪽 단추로 클릭하여) 선택된 모든 빈을 새로운 위치로 끌기 조작할 수 있습니다.

빈 복사

빈을 추가한 후, 빈 팔레트에서 다른 빈을 추가하는 대신 그 빈을 복사할 수 있습니다. 빈 복사는 같은 빈의 여러 사본을 추가하는 한 가지 방법입니다. 빈 복사의 한가지 명확한 장점은 한 빈에 공용 수정을 한 후 단지 필요한만큼 이를 중복시키면 된다는 것입니다. 연결을 가진 빈을 복사할 경우 연결은 중복되지 않습니다.

빈을 복사하려면, 다음을 실행하십시오.

1. Ctrl 키를 누른 채 복사하고 싶은 빈을 적절한 마우스 단추로 선택하십시오.
   - OS/2와 UNIX 플랫폼에서는 마우스 오른쪽 단추를 사용하여 빈을 복사합니다.

- Windows에서는, 마우스 왼쪽 단추를 사용하십시오.

2. 새로운 빈을 원하는 위치로 마우스 포인터를 끌기 조작하고 마우스 단추와 Ctrl 키에서 손을 떼십시오.

복사하고 싶은 모든 빈을 선택하여 몇개의 빈을 한번에 복사할 수 있습니다. 그런 후 Ctrl 키를 누르고 임의로 선택한 빈을 잡고서 빈의 사본을 새로운 위치로 끌기 조작하십시오.

클립보드를 사용하여 빈 복사

클립보드를 사용하여 빈을 복사하려면 다음 단계를 따르십시오.

1. 복사하고 싶은 하나 이상의 빈을 선택하십시오.
2. 비주얼 컴포지션 편집기의 **편집** 메뉴에서 **복사**를 선택하십시오.
3. 그런 후 **편집** 메뉴에서 **붙여넣기**를 선택하십시오. 마우스 포인터가 십자형이 됩니다.
4. 새로운 하나 이상의 빈을 추가하고 싶은 위치로 마우스 포인터를 이동하고 마우스 왼쪽 단추를 클릭하십시오.

빈 삭제

빈을 삭제하려면, 간단히 빈을 선택하고 Delete 키를 누르거나 또는 빈의 팝업 메뉴에서 **삭제**를 선택하십시오.

여러 개의 빈을 삭제하려면, 삭제 작동 수행에 앞서 삭제하려는 빈들을 같이 선택하십시오.

빈을 향한 또는 빈으로부터 나온 연결을 포함한 빈을 삭제할 경우, 빈과 그 모든 연결이 삭제됩니다. 그런데 이 경우, 빈과 연결을 삭제하기 전에 작업을 계속 진행할지 여부를 확인하도록 프롬프트됩니다. 실수로 유지하고 싶은 항목을 삭제한 경우, 단지 비주얼 컴포지션 편집기의 **편집** 메뉴에서 **실행 취소**를 선택하십시오.

빈 크기조정, 정렬 및 위치 지정

이 절에서는 빈의 *크기*조정, 정렬, 위치 지정에 사용할 수 있는 비주얼 컴포지션 편집기 내의 기능을 설명합니다.

주: 컨테이너(JApplet이나 JFrame같은)인 빈에는 배치 특성이 있습니다. 이 특성은 컨테이너 내에서 빈의 위치를 제어하는 특정 배치 관리 프로그램을 지원합니다. 배치 관리 프로그램 사용은 사용자 인터페이스를 작성하기 위해 선호되는 방법입니다. <null> 배치를 사용하는 경우 비주얼 컴포지션 편집기는 빈을 정렬하고 위치 지정하는 도구를 제공합니다. 원하는 대로 빈을 정렬한 다음 <null> 배치를 외양이 *크게* 변하지 않는 GridBagLayout으로 변환하십시오.

빈 크기조정

빈을 *크기*조정하려면, 다음 단계를 따르십시오.

1. *크기*조정하고 싶은 빈을 선택하십시오. 선택 조정이 각 모서리에 표시됩니다.
2. 마우스 왼쪽 단추를 사용해서 선택 조정 중 하나를 끌기 조작하여 빈을 *크기* 조정하십시오.

마우스에서 손을 떼기 전에, 빈의 새로운 *크기*를 알려 주는 빈의 윤곽선이 표시됩니다.

빈을 한 방향으로만 *크기*조정하려면, 선택 핸들을 가로로 또는 세로로 끌기 조작하는 동안 Shift 키를 누르고 있으십시오.

빈의 특성 창에서 *제약 조건* 특성을 사용하여 빈 *크기*를 조정할 수도 있습니다. 특성 창에 대한 자세한 정보는 70 페이지의 『빈 특성 변경』을 참조하십시오.

빈 정렬

<null> 배치에서 빈을 다른 빈과 정렬하려면 다음을 수행하십시오.

1. 마지막으로 선택된 빈이 다른 빈과 정렬시키고 싶은 빈인지 확인하고 정렬하고 싶은 빈을 선택하십시오.
2. 도구 막대에서 다음 정렬 도구 중 하나를 선택하십시오.

 **왼쪽 정렬**

 **위쪽 정렬**

 **중앙 정렬**

 **가운데 정렬**

 **오른쪽 정렬**

 **아래쪽 정렬**

다른 빈과 치수 일치

다른 빈과 동일한 가로 또는 세로를 가지도록 빈을 *크기조정*할 수 있습니다.

1. 마지막으로 선택된 빈이 다른 빈과 정렬시키고 싶은 빈인지 확인하고 일치시키려는 빈을 선택하십시오.
2. 도구 막대에서 다음 *크기조정* 도구 중 하나를 선택하십시오.

 **폭 일치**

 **높이 일치**

또한 두 개 이상의 빈을 선택한 후 마우스 오른쪽 단추를 클릭하여 이들의 치수를 일치시킬 수 있습니다. 이때 나타나는 팝업 메뉴에서 **배치**를 선택한 후 **크기 일치**를 선택하십시오. **폭**, **높이** 또는 **둘다** 일치를 선택할 수 있습니다.

빈 균등 분배

<null> 배치를 사용하는 컴포지트 빈내에서 빈들을 균등하게 분배하려면, 다음을 수행하십시오.

1. 균등하게 분배하고 싶은 빈을 선택하십시오.
2. 도구 막대에서 다음 분배 도구 중 하나를 선택하십시오.

   수평 분배

    **수직 분배**

다중 선택된 빈을 둘러싼 가상의 바운딩 박스 내에서 빈을 균일하게 분배하려면, 다음 단계를 따르십시오.

1. 균일하게 분배하고 싶은 빈을 다중 선택하십시오. 최소 3개의 빈을 선택해야 합니다.
2. 선택한 빈의 팝업 메뉴에서 **배치**와 **분배**를 선택하십시오. 다음 중 하나를 선택하십시오.

   바운딩 박스에서 수평 분배
   가장 왼쪽 빈의 왼쪽 가장자리와 가장 오른쪽 빈의 오른쪽 가장자리를 경계로 하는 영역에 선택된 빈을 균등하게 분배합니다.

   바운딩 박스에서 수직 분배
   가장 위쪽 빈의 위쪽 가장자리와 가장 아래쪽 빈의 아래쪽 가장자리를 경계로 하는 영역에 선택된 빈을 균등하게 분배합니다.

   분배에는 다음과 같이 선택이 두 개 더 있습니다.

   작업영역 내 수평
   선택된 빈을 도구 막대의 **수평 분배**와 동일한 방식으로 분배합니다.

   작업영역 내 수직
   선택된 빈을 도구 막대의 **수직 분배**와 동일한 방식으로 분배합니다.

빈 특성 변경

특성 창은 빈 또는 연결과 연관된 기타 옵션 및 특성을 표시하고 설정하는 방법을 제공합니다. 빈 고유 특성 외에 데이터 확인과 배치 특성을 설정할 수 있습니다.

빈 특성 창 열기

빈의 특성 창을 열려면, 다음 중 하나를 실행하십시오.

- 빈에서 두 번 클릭하십시오.
- 빈의 팝업 메뉴에서 **특성**을 선택하십시오.
- 빈을 선택하고 도구 막대에서 특성  을 선택하십시오.

특성 창이 열려 있는 동안 다음을 수행하여 창에서 다른 빈의 특성을 볼 수 있습니다.

- 다른 빈 선택
- 특성 창의 맨 위에 있는 드롭다운 목록에서 삽입된 또다른 빈 선택

다음은 빈의 특성 창 예제입니다.

빈 특성명과 그 값은 테이블 형식으로 표시됩니다. 특성 값이 변경되는 방식은 특성 유형 자체에 따라 달라집니다. JTextField 빈을 예로 들면 *beanName* 특성의

값은 문자열이며 특성 창의 셀 내에서 직접 변경할 수 있습니다. 몇몇 특성 값은 드롭다운 목록에서 선택하여 변경할 수 있습니다. 기타 빈 특성 값(예를 들어, 색 및 글꼴)은 그 목적에 맞게 표시된 두 번째 창을 통해 변경할 수 있습니다.

임의의 빈 특성을 편집하려면, 그 특성 창을 열고 변경하고 싶은 값을 클릭하십시오. 값이 문자열이거나 정수 값일 경우, 이를 직접 편집할 수 있습니다. 색상 값의 경우에는 값 열에서  단추를 선택하여 색상 창을 여십시오. 값이 부울값인 경우, 테이블의 값 열에 있는 셀을 클릭하고 드롭다운 목록에서 **True**나 **False**를 선택하십시오.

빈의 특성을 변경한 후, 다음 방법으로 특성을 적용할 수 있습니다.

- 특성 창에서 다른 입력항목을 선택합니다.
- 특성 창을 닫습니다.
- 다른 창을 클릭하거나 빈 작업 영역을 클릭합니다.

빈 색상 및 글꼴 변경

비주얼 빈에 추가된 또다른 개선점은 빈이 사용하는 색상 및 글꼴을 변경하는 것입니다.

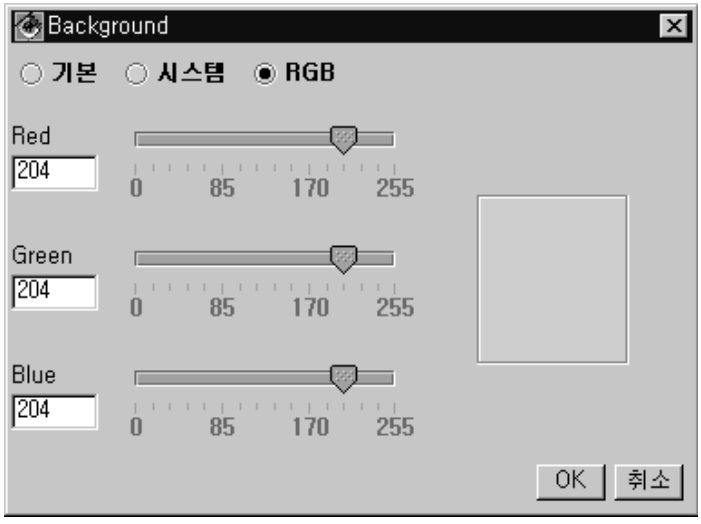
다중 플랫폼에서 사용될 애플릿을 개발중이라면, 기본 시스템의 색상 및 글꼴과 다른 색상 및 글꼴 선택시 미치는 영향을 주의깊게 고려해야 합니다. 예를 들어, OS/2에서 사용할 수 있는 특정 글꼴을 선택한 경우, 그 글꼴은 Windows에서 사용하지 못할 수 있습니다. 자세한 정보는 73 페이지의 『색상과 글꼴 이식성』을 참조하십시오.

빈 색상 변경

1. 비주얼 컴포지션 편집기에서 색을 변경하려는 JButton 빈을 두 번 클릭하십시오. 그러면 특성 창이 나타납니다.

2. 빈의 배경색을 변경하려면 특성 창에서 *Background* 특성 값을 선택합니다.

   표시되는 단추  를 선택하십시오. 백그라운드 창이 열립니다.



   색상 구성표는 사용자의 워크스테이션 디스플레이 설정에 따라 다르게 표시됩니다. 이런 경우에는 RGB 스펙이 사용됩니다.

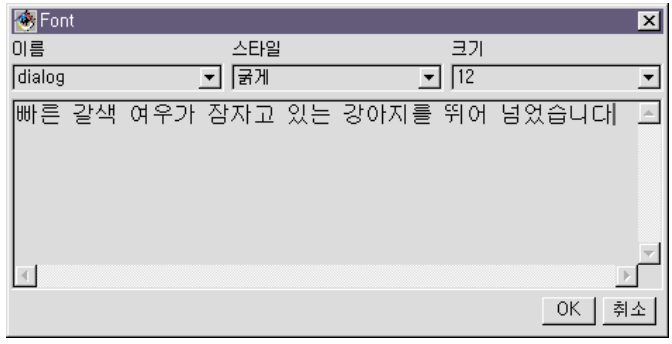
3. 색상을 변경하려면 슬라이더 제어를 사용하거나, **기본** 또는 **시스템**을 선택한 다음 색을 선택하십시오. **OK**를 선택하십시오.

주: 메뉴에서 빈 색상을 변경할 수 없습니다.

빈 글꼴 변경

1. 비주얼 컴포지션 편집기에서 변경하고 싶은 글꼴을 가진 빈을 두 번 클릭하십시오.

2. 열린 특성 창에서 *font* 특성 값을 선택하십시오. *font* 값 열에 나타나는 단추  를 선택하십시오. 글꼴 창이 열립니다.



3. **이름** 드롭다운 목록에서 원하는 글꼴을 선택하십시오.
4. **스타일**과 **크기** 목록에서 사용할 크기와 스타일을 선택하십시오. 선택한 글꼴의 샘플이 텍스트 영역에 표시됩니다. 이 영역에 추가 텍스트를 입력하여 다양한 문자의 모양을 볼 수 있습니다.
5. 글꼴 지정 완료시, **OK**를 선택하십시오. 선택한 글꼴이 *font* 값 열에 나타납니다.

주: 메뉴 빈같은 일부 빈에서는 대상 플랫폼에 따라 글꼴을 변경할 수 없는 경우도 있습니다.

색상과 글꼴 이식성

애플릿이 다중 플랫폼에서 사용될 경우, 빈의 색상과 글꼴은 애플릿을 수행할 모든 시스템에서 사용할 수 있어야 합니다.

애플릿에서 빈의 색상을 변경하기로 결정한 경우, 기본색이 아닌 색이 다른 플랫폼에서 다르게 나타날 수 있기 때문에 창에서 기본 색만을 사용하십시오.

빈의 글꼴을 변경하기로 결정한 경우, 사용자가 선택한 글꼴을 완료된 프로그램을 수행할 모든 시스템에서 사용할 수 있는지 확인하십시오. 또한 애플릿이 영어 이외의 언어로 디자인된 코드 페이지를 사용하는 시스템에서 수행할 경우, 특정 글꼴 문제를 가질 수도 있습니다.

빈 연결

15 페이지의 『제4장 첫번째 애플릿 빌드 소개』에서 연결 작성에 대해 배웠습니다. 이 절에서는 서로 다른 유형의 연결과 그 유형으로 실행할 수 있는 작업을 살펴봅니다. 서로 다른 유형의 연결이 설명되어 있는대로 비주얼 컴포지션 편집기에서 따라해 보고 논의된 예제를 시도해 보는 것이 가장 좋습니다. 연결을 작성하고 시험하는 것이 사용에 대해 배울 수 있는 가장 좋은 방법입니다.

주: 75 페이지의 『특성간 연결』에서는 새로운 애플릿을 작성합니다.이 절에 나타난 모든 예제에 이 애플릿을 재사용할 수 있습니다.

연결의 6가지 유형은 다음과 같습니다.

특성간 연결
: 특성간 연결은 소스 및 목표 이벤트가 연결의 특성 창에 지정된 경우 두 값이 동기 상태를 유지하므로 두 데이터 값을 함께 링크합니다.

이벤트와 메소드 연결
: 이벤트와 메소드 연결은 이벤트가 발생할 때 메소드를 호출합니다.

이벤트와 코드 연결
: 이벤트와 코드 연결은 이벤트 발생시 코드 일부를 수행합니다.

특성에서 매개변수 연결
: 특성에서 매개변수로 연결은 특성 값을 연결의 매개변수로서 사용합니다.

코드에서 매개변수 연결
: 코드에서 매개변수로 연결은 연결 매개변수가 필요할 때 코드를 수행합니다.

메소드에서 매개변수 연결
: 메소드에서 매개변수로의 연결에서는 메소드의 결과를 연결의 매개변수로서 사용합니다.

이벤트와 코드 연결 및 코드에서 매개변수 연결을 통해 컴포지트 빈의 메소드에 연결할 수 있습니다.

연결은 소스와 대상을 가집니다. 연결을 시작하는 지점을 *소스*라고 하고 연결이 끝나는 지점을 *대상*이라고 합니다. 연결 특성에 대한 정보는 82 페이지의 『연결 특성 변경』을 참조하십시오.

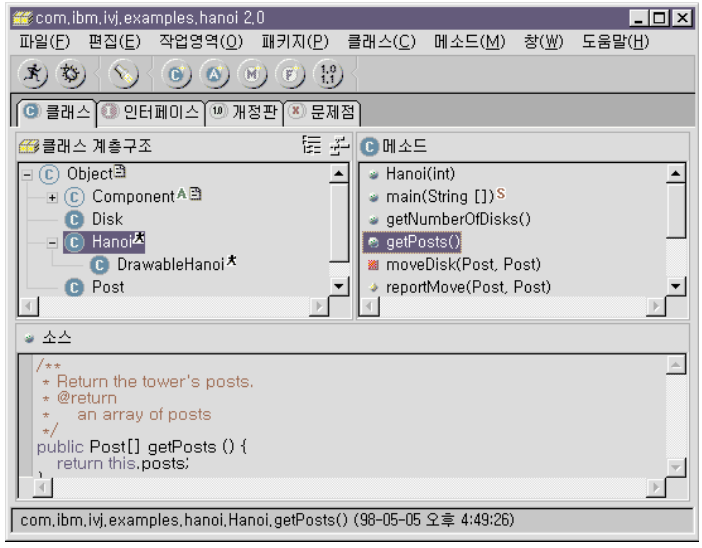
주: 특정한 빈 메소드, 특성 또는 이벤트는 빈이나 연결 팝업 메뉴의 빈 선호 기능 목록에 나타나지 않습니다. 전체 목록을 표시하려면 빈 팝업 메뉴나 연결 팝업 메뉴에서 **연결**, **연결가능 기능**을 선택하십시오. **연결가능 기능**을 선택하여 열린 창에 표시된 메소드, 특성 및 이벤트 목록은 빈의 전체 *public 인터페이스*를 나타냅니다.

특성간 연결

특성간 연결은 두 데이터 값을 함께 결합합니다. 이런 연결 유형은 파란색으로 표시됩니다. 특성간 연결의 간단한 예제는 다음과 같습니다.

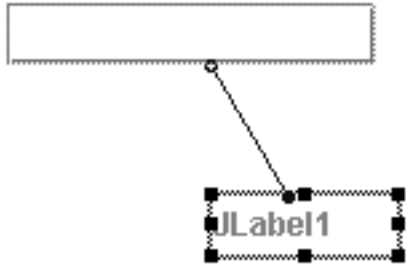
1. 애플릿 작성 SmartGuide를 사용하여 새로운 애플릿을 작성하십시오.
   - Workbench 도구 막대에서 애플릿 작성  을 선택하십시오.
   - 애플릿 작성 SmartGuide에서 애플릿의 이름을 입력하고 그 애플릿용 프로젝트와 패키지를 지정하십시오. **최상위 클래스**의 경우, **검색**을 사용하고 패턴으로 JApplet을 사용하여 com.sun.java.swing.JApplet을 가져오십시오.
   - **클래스를 시각적으로 구성**을 선택했는지 확인하고 **완료**를 선택하십시오.
2. 사용자가 만든 새로운 애플릿에 대해 비주얼 컴포지션 편집기가 열리면, 기본 애플릿 빈에 JTextField 빈과 JLabel 빈을 지정하십시오.
3. JTextField 빈의 *text* 특성을 JLabel 빈의 *text* 특성에 연결하십시오.
   - JTextField 빈을 선택하고 마우스 오른쪽 단추로 클릭하십시오. 팝업 메뉴에서 **연결**을 선택하고 **text**를 선택하십시오.
   - JLabel 빈을 마우스 왼쪽 단추로 클릭하고 팝업 메뉴에서 **text**를 선택하십시오.
4. 방금 작성한 새로운 연결을 선택하고 마우스 오른쪽 단추를 클릭하십시오. 이때 나타나는 팝업 메뉴에서 **특성**을 선택하십시오. 해당 연결의 특성 창이 나타

납니다.



5. 특성 창에서 **소스 이벤트**에 대해 **actionEvents**를 선택하고 **OK**를 선택하십시오.

   빈 작업 영역은 다음과 같이 나타나야 합니다.



6. JTextField 빈의 *text* 특성을 `Initial contents`로 설정하십시오.

이러한 빈이 들어 있는 애플릿을 수행할 때, JLabel 텍스트는 **Initial contents**가 됩니다.

특성간 연결의 경우, 끝점이 소스 또는 목표 역할을 할 수 있습니다. 초기화에서만 연결의 소스가 무슨 특성이고 목표가 무슨 특성인지 중요합니다. 초기화하는 동안 목표 값은 소스 값과 일치하도록 갱신됩니다.

특성간 연결은 팝업 메뉴에서 소스 빈의 **연결**을 선택하면 시작되고 적절한 대상 빈 기능을 선택하면 완료됩니다.

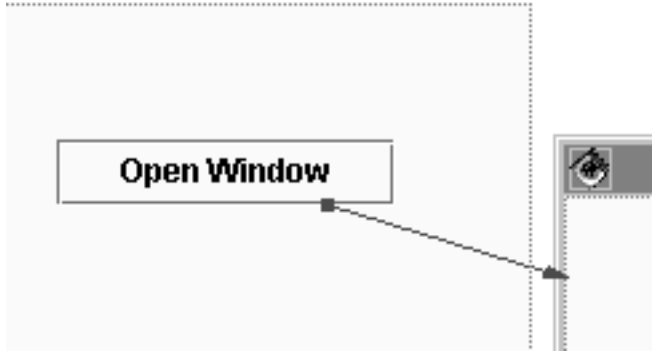
이벤트와 메소드 연결

이벤트와 메소드 연결은 특정 이벤트가 발생할 때 메소드를 호출합니다. 이 연결 유형은 초록색으로 표시됩니다.

이벤트에서 메소드 연결의 경우, 이벤트는 항상 소스이며 메소드는 항상 목표입니다. 이벤트에서 메소드로의 연결의 단순한 예제는 다음과 같습니다.

1. 비주얼 컴포지션 편집기에서 기본 JApplet 빈에 JButton 빈을 두십시오. 단추의 텍스트를 `Open Window`로 변경하십시오.
2. 비주얼 컴포지션 편집기의 빈 작업 영역에 JFrame 빈을 두십시오.
3. JButton의 *actionPerformed* 이벤트를 JFrame 빈의 *show* 메소드로 연결하십시오. 이 연결을 수행하려면, JFrame의 제목 막대(상자 안이 아니라)를 클릭하십시오. 단추가 선택되면 프레임이 나타납니다.

빈 작업 영역은 다음과 같이 나타나야 합니다.



또한 특성은 메소드를 호출하는 데 사용될 수 있고 이벤트는 특성 값을 변경하는 데 사용될 수 있습니다. 이러한 수행은 VisualAge for Java가 이벤트를 특성 값 내 변경과 관련시킬 수 있기 때문에 가능합니다. 따라서 다음 특성과의 연결을 이룰 수 있습니다.

특성에 이벤트 연결

메소드 호출 외에 이벤트도 특성 값을 설정하는 데 사용될 수 있습니다. 이 경우, 매개변수는 특성 값을 제공하는 연결과 함께 사용되어야 합니다. 특성과 이벤트 연결의 단순한 예제는 다음과 같습니다.

1. JLabel 빈을 비주얼 컴포지션 편집기의 기본 애플릿 빈 내에 놓고 특성 창에서 *text* 특성을 `This is a JFrame title`로 변경하십시오.
2. 비주얼 컴포지션 편집기에서 애플릿 빈에 JButton 빈을 두십시오.
3. 비주얼 컴포지션 편집기의 빈 작업 영역에 JFrame 빈을 두십시오.
4. JButton의 *actionPerformed* 이벤트를 JFrame 빈의 *show* 메소드로 연결하십시오. *show* 메소드를 보려면 JFrame의 제목 막대를 클릭해야 합니다.
5. JFrame 빈의 *componentShown* 이벤트를 JFrame 빈의 *title* 특성에 연결하십시오.
6. 이제, 방금 작성한 이벤트와 특성 연결에 대한 매개변수를 제공하려면, JLabel 빈의 *text* 특성을 연결 팝업 메뉴의 *value* 특성에 연결하십시오.

애플릿을 실행하고 단추를 선택하면 프레임의 제목 텍스트가 `This is a JFrame title`로 설정됩니다. 이 예제는 설명을 위한 것이지만 내용을 잘 전달해 줍니다. 자세한 정보는 83 페이지의 『연결 매개변수』를 참조하십시오.

이벤트와 코드 연결

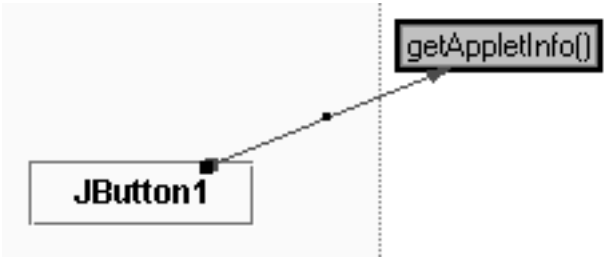
이벤트와 코드 연결은 특정 이벤트 발생시 주어진 메소드를 수행합니다. 이것은 Java 언어 사용을 통해 직접 애플릿 수행을 구현 또는 변경하는 방법을 제공합니다. 이벤트와 코드 연결 메소드의 대상은 사용자가 비주얼 컴포지션 편집기에서 조작중인 클래스의 임의의 메소드가 될 수 있습니다.

주: 이벤트와 메소드 연결이 두 빈 사이에 작성됩니다. 이벤트와 코드 연결은 컴포지트 빈에서 하나의 빈과 하나의 메소드 사이에 작성됩니다. 컴포지트 빈의 메소드가 반드시 public일 필요는 없습니다.

이벤트와 코드 연결은 초록입니다. 이벤트와 코드 연결을 작성하려면 다음을 수행하십시오.

1. 소스 빈(예: JButton)을 선택하십시오. 마우스 오른쪽 단추를 클릭하여 빈의 팝업 메뉴를 표시하십시오. **연결**을 선택한 후 *actionPerformed*와 같은 이벤트를 선택하십시오. 그러면 마우스 포인터 모양이 변합니다.
2. 마우스 포인터를 빈 작업 영역의 개방 영역으로 이동하십시오. 이것은 기본 애플릿 빈을 포함하여 임의의 빈 위에 있을 수 없습니다. 마우스 왼쪽 단추를 클릭하고 팝업 메뉴에서 **이벤트와 코드 연결**을 선택하십시오.
3. 그 결과 나타난 창에서 사용할 수 있는 메소드의 목록을 선택하거나 새로운 메소드를 작성할 수 있습니다.
4. 일단 메소드를 선택하면, **OK**를 선택하여 연결을 완료하십시오.

소스 빈과 메소드 이름이 포함된 이동가능한 텍스트 상자 사이에 연결선이 표시됩니다.



매개변수 연결

마지막 세 개의 연결 유형은 다양한 소스로부터 연결에 매개변수를 제공합니다.

- 특성에서 매개변수로
- 코드에서 매개변수로
- 메소드에서 매개변수로

이 연결 유형은 보라색으로 표시됩니다.

특성에서 매개변수 연결

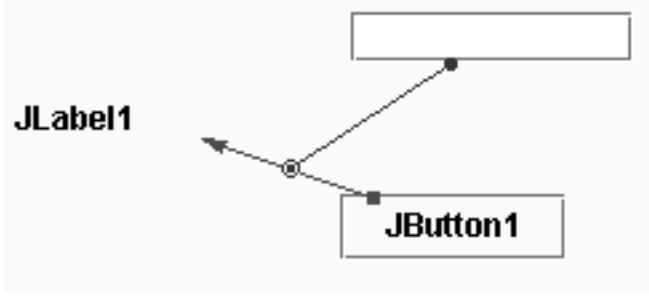
특성에서 매개변수 연결에서는 특성 값을 연결의 매개변수로서 사용합니다. 기타 연결 유형과 마찬가지로, 특성에서 매개변수 연결은 팝업 메뉴 내 소스 빈의 **연결** 선

택으로 시작하여 매개변수를 요구하는 목표 연결선 위에서의 마우스 왼쪽 단추를 클릭함으로써 종료됩니다. 그런 후 그 연결의 팝업 메뉴로부터 적절한 특성을 선택합니다.

특성에서 매개변수 연결이 To-Do List 샘플 작성에 사용되었습니다. JTextField 빈의 *text* 특성을 JButton 빈과 DefaultListModel 빈 사이의 연결에 연결했다면, 특성에서 매개변수 연결을 작성하고 있는 것입니다. 또한 다음 예제는 특성에서 매개변수 연결의 사용을 나타냅니다. 또한 다음 예제는 특성에서 매개변수 연결의 사용을 나타냅니다.

1. JLabel 빈, JTextField 빈 및 JButton 빈을 비주얼 컴포지션 편집기의 애플릿 빈에 두십시오.
2. JButton 빈의 *actionPerformed(awt.java.event.ActionEvent)* 이벤트를 JLabel 빈의 *text* 특성에 연결하십시오.
3. 이제 JTextField 빈의 *text* 특성을 연결 팝업 메뉴의 *value* 특성에 연결하여 특성에서 매개변수로의 연결을 작성하십시오.

빈 작업 영역은 다음과 같이 나타나야 합니다.



애플릿을 수행하고 text 필드에 텍스트를 입력하면, label text가 text 필드 내 텍스트와 일치하도록 설정됩니다.

코드에서 매개변수 연결

코드에서 매개변수 연결은 연결에 매개변수가 필요할 때마다 하나의 메소드를 수행합니다. 이 연결은 연결에 제공된 값이 빈 특성 값 대신 Java 메소드로부터 리턴되었다는 점을 제외하면 특성에서 매개변수로의 연결과 비슷합니다.

그림으로 나타내기 위하여 애플릿 클래스에 텍스트 `this is a string`을 리턴하는 간단한 메소드 *stringFromCode*를 만들었다고 가정합니다. 코드에서 매개변수 연결 예제는 다음과 같습니다. 기타 연결 유형과 달리, 코드에서 매개변수 연결은 연결 팝업 메뉴의 매개변수 이름으로부터 시작하여 다음과 같이 종료됩니다.

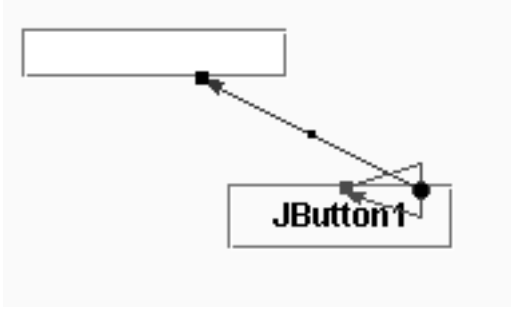
1. 비주얼 컴포지션 편집기의 기본 JApplet 빈에 JLabel 빈과 JButton 빈을 두십시오.
2. JButton 빈의 *actionPerformed* 이벤트를 JLabel 빈의 *text* 특성에 연결하십시오.
3. 이제 연결 팝업 메뉴에서 다음과 같이 *value* 특성을 메소드에 연결하여 코드에서 매개변수 연결을 작성하십시오.
   - 마우스 오른쪽 단추로 연결을 클릭하고 팝업 메뉴에서 **연결**을 선택한 다음 **value**를 선택하십시오. 그러면 마우스 포인터 모양이 변합니다.
   - 빈 작업 영역의 개방 영역을 마우스 왼쪽 단추로 클릭하여 팝업 메뉴에서 **코드에서 매개변수로 연결**을 선택하십시오.
   - 그 결과 나타난 창에서 사용할 수 있는 메소드의 목록을 선택할 수 있습니다. 이 경우, 메소드 *stringFromCode*가 나타납니다.(이것은 우리가 애플릿 클래스에서 작성한 메소드입니다.)
   - 일단 메소드를 선택하면, **OK**를 선택하여 연결을 완료하십시오.

메소드에서 매개변수 연결

메소드에서 매개변수 연결에서는 메소드의 결과를 연결의 매개변수로서 사용합니다. 메소드에서 매개변수 연결 사용 방법의 예제는 다음과 같습니다.

1. 비주얼 컴포지션 편집기에의 기본 애플릿 빈에 JButton 빈과 JTextField 빈을 두십시오.
2. JButton 빈의 *actionPerformed* 이벤트를 JButton 빈의 *text* 특성에 연결하십시오. (적절한 상황에서 동일 빈의 특성에 대한 이벤트 연결은 제대로 작동합니다.)
3. 그런 후 TextField 빈의 *getText()* 메소드를 연결 팝업 메뉴의 *value* 특성에 연결하여 메소드에서 매개변수 연결을 작성하십시오. 이것은 필요한 연결 매개변수를 제공하며 연결선이 단색이 되도록 합니다.

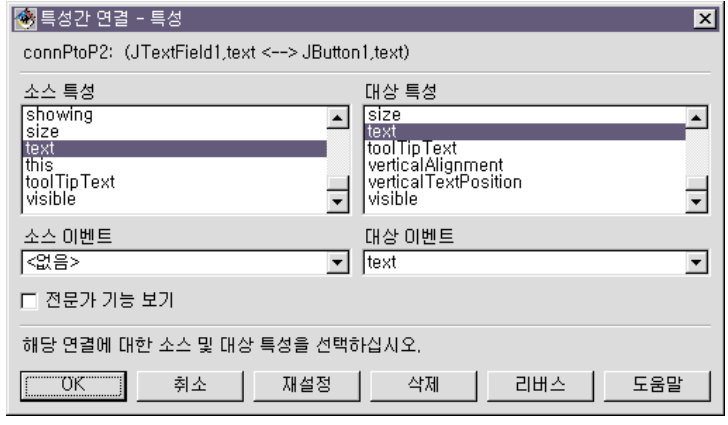
애플릿 테스트시 text 필드에 문자열을 입력한 후 Button을 선택하십시오. 문자열이 그 Button의 레이블이 됩니다.



연결 특성 변경

빈과 마찬가지로 연결은 특성을 가집니다. 연결 특성을 열려면, 연결의 팝업 메뉴에서 **특성**을 선택하십시오. 연결을 마우스로 두 번 클릭하십시오.

다음 그림은 특성간 연결의 특성 창을 나타냅니다.



연결의 특성 창을 사용해서 연결의 소스 또는 대상 특성을 변경할 수 있습니다. 이를 실행하려면, 적절한 목록에서 다른 소스 또는 대상 특성을 선택하십시오. 연결의 현재 소스 및 대상 특성을 표시하려면, **재설정**을 선택하십시오.

특성간 연결의 소스가 대상이 되게 하려면 또는 그 반대의 경우에도 해당 연결의 특성 창에서 **방향 전환**을 선택하여 소스와 대상 특성을 변경할 수 있습니다.

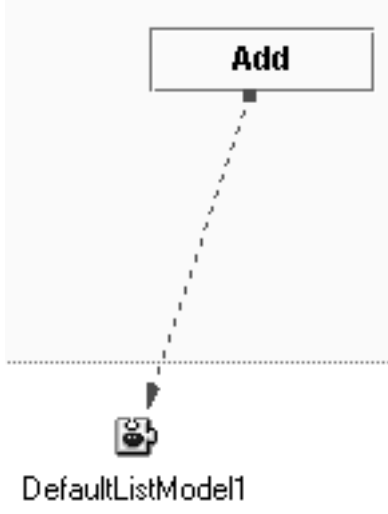
연결 특성 변경을 완료한 다음 **OK**를 선택하십시오.

연결 매개변수

이벤트와 메소드 연결 및 이벤트와 코드 연결은 때때로 매개변수(또는 인수)를 필요로 합니다. 메소드의 매개변수는 연결의 특성으로서 사용할 수 있습니다. 따라서 매개변수를 지정하려면, 단지 이벤트와 메소드 연결 자체의 매개변수 특성에 연결을 작성하십시오.

연결이 지정되지 않은 매개변수를 요구할 때 이것은 점선으로 나타나 연결이 완료되지 않았음을 표시합니다.

To-Do List 애플릿에서 JButton 빈의 *actionPerformed* 이벤트를 DefaultListModel 빈의 *addElement(java.langObject)* 메소드로 연결하여 점선이 표시되었습니다.



메소드가 요구하는 매개변수는 메소드명 내 괄호 () 속 항목에 의해 지시됩니다. 예를 들어, *addElement(java.lang.Object)* 메소드는 한개의 매개변수, Object를 취합니다. *insert ElementAt(java.lang.Object, int)* 메소드는 두 개의 매개변수, Object와 int를 취합니다.

필수 매개변수를 모두 지정했다면, 연결선이 실선이 되어 연결이 완료되었음을 나타냅니다. 연결에 대해 충분한 매개변수를 제공하지 않을 경우, 연결은 계속해서 점선으로 나타납니다.

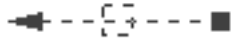
특성이나 상수를 사용하여 매개변수를 지정할 수 있습니다.

매개변수로서의 특성

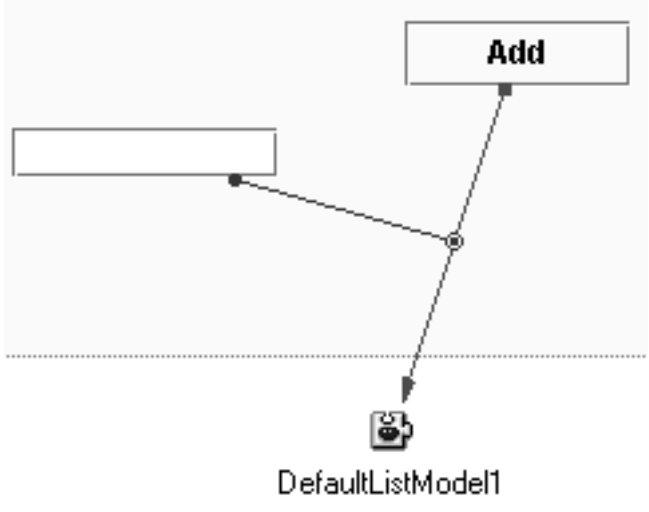
대부분의 경우, 필요한 매개변수는 비주얼 컴포지션 편집기에서 작업중인 다른 빈의 특성입니다. 빈의 특성을 매개변수로서 사용하려면, 다음을 실행하십시오.

1. 빈의 특성을 소스로 사용하여 새로운 연결을 작성하십시오.
2. 대상의 경우, 매개변수를 요구하는 연결선에서 마우스 왼쪽 단추를 클릭한 후 연결 메뉴에서 지정중인 특정 매개변수 특성을 선택하십시오.

   연결 선으로의 연결을 작성하는 동안 마우스 포인터가 연결선 바로 위에 있으면, 포인터가 적절한 위치에 있다는 표시로는 연결선 중간에 작은 표시가 나타납니다.



To-Do List 애플릿에서 JTextField 빈에 입력한 텍스트는 **Add** 단추와 DefaultListModel 빈 간의 이벤트와 메소드 연결 매개변수로 사용됩니다.



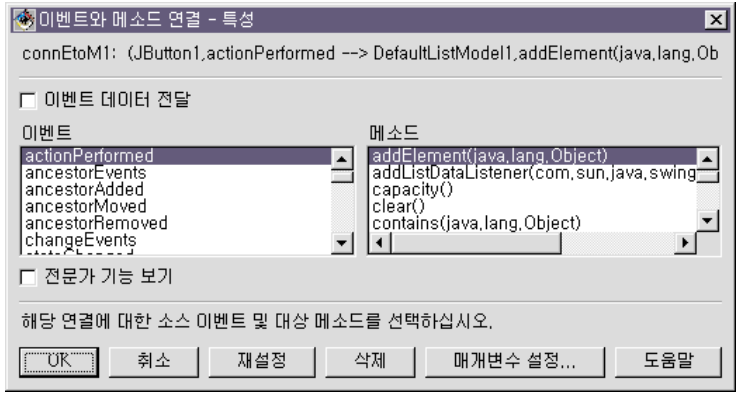
이 예에서 JTextField 빈의 *text* 특성과 이벤트와 메소드 연결의 *obj* 특성 사이에 특성간 연결을 이루어 이벤트와 메소드 연결의 매개변수를 제공했습니다. 연결의 *obj* 특성은 *addElement(java.lang.Object)* 메소드의 첫번째이자 유일한 매개변수 이름입니다.

매개변수로서의 상수

또한 매개변수 값이 상수도 될 수 있습니다. 연결의 특성 창에서 매개변수에 대해 상수 값을 지정합니다.

예를 들어, 이벤트와 메소드 연결의 매개변수로 상수 값을 지정하려면, 다음을 실행하십시오.

1. 이벤트와 메소드 연결에서 두 번 클릭하십시오. 그러면 이벤트와 메소드 연결의 특성 창이 열립니다.



2. 이 창에서 **매개변수 설정**을 선택하십시오. 그러면 상수 매개변수 값 설정 창이 열립니다.

3. 상수 매개변수 값 설정 창에서 추가할 매개변수의 상수 값을 입력하십시오.
4. 종료시 **OK**를 선택한 후 이벤트와 메소드 연결의 특성 창에서 **OK**를 선택하십시오.

연결 조작

빈과 마찬가지로 일단 연결이 작성되면, 여러 가지 방법으로 연결을 조작할 수 있습니다.

연결 선택 및 선택 취소

빈을 선택하고 선택 취소할 때와 같은 방법으로 연결을 선택하거나 선택 취소할 수 있고, 여러 개의 연결을 선택할 수 있습니다. 현재 선택된 연결에 대한 정보가 비주얼 컴포지션 편집기 아래 정보 영역에 표시됩니다.

주: 빈과 연결을 동시에 선택할 수 없습니다.

연결 삭제

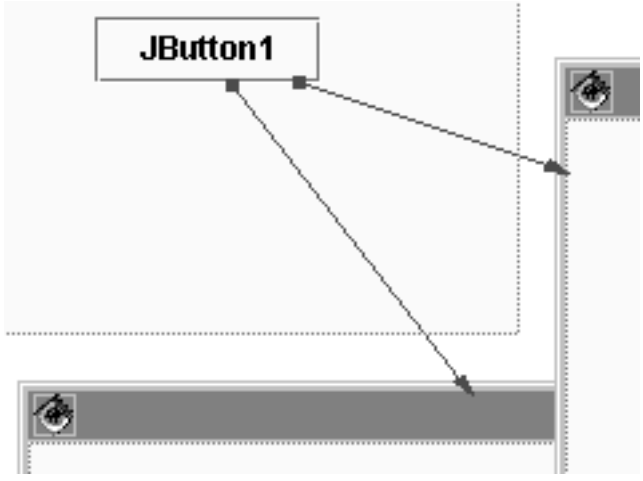
연결을 삭제하려면, 팝업 메뉴에서 **삭제**를 선택하십시오. 또한 연결을 선택하고 Delete 키를 눌러 연결을 삭제할 수도 있습니다.

여러 개의 연결을 삭제하려면, 삭제할 연결들을 선택한 후 선택된 연결 중 하나의 팝업 메뉴에서 **삭제**를 선택하십시오.

연결 재정렬

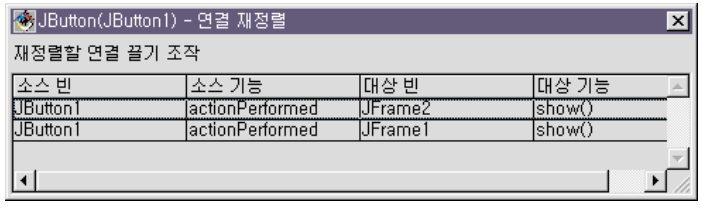
동일 이벤트 또는 빈의 특성에서 몇 개의 연결을 작성하면, 작성된 순서대로 수행됩니다. 그러나 수행하고 싶은 순서와 다른 순서로 연결을 작성한 경우에는 재정렬할 수 있습니다. 하나의 JButton과 두 개의 JFrame 컨테이너를 추가하십시오. 단추로부터의 *actionPerformed* 이벤트를 한 프레임의 *show* 메소드로 연결하십시오. 두 번째 프레임으로 같은 연결을 작성하십시오. 이제 열려 있는 프레임으로의 연결이 두 개가 되었습니다. 애플릿을 실행하면 한 프레임이 나타나고 단추를 클

릭한 후에 두번째 프레임이 나타납니다.



연결 순서를 변경해야 할 경우, 다음을 실행하여 빈으로부터 연결을 재정렬하십시오.

1. 빈의 팝업 메뉴에서 **나가는 연결 재정렬**을 선택하십시오. 그러면 연결 재정렬 창이 나타납니다.



   연결 재정렬 창에는 사용자가 선택한 빈의 모든 연결이 들어 있습니다. 이 예제에서는 `JButton1, actionPerformed -> JFrame2, show()` 연결이 먼저 실행됩니다.

2. 연결 재정렬 창에서 적절한 마우스 단추를 사용하여 프레임 표시 순서를 바꾸십시오. OS/2에서는 마우스 오른쪽 단추를 사용하고, Windows에서는 마우스 왼쪽 단추를 사용하십시오.

   목록에서 연결을 끌기하면 어두운 선이 나타나 마우스 단추에서 손을 뗄 때 연결이 삽입될 위치를 지시합니다. 효과를 보려면 애플릿을 재수행하십시오.

연결 보기, 숨기기 및 검색

도구 막대에서 연결 숨기기 도구  와 연결 보기 도구  를 사용하여 연결을 보거나 숨길 수 있습니다.

이러한 도구는 선택된 빈용으로 들어가거나 빈에서 나오는 모든 연결을 표시하거나 숨깁니다. 빈이 선택되어 있지 않으면 이 도구는 비주얼 컴포지션 편집기의 모든 연결을 표시하거나 숨깁니다.

빈의 팝업 메뉴에서 **연결 검색**을 선택하여 빈의 연결을 선택적으로 보거나 숨길 수 있습니다.

들어온 연결 보기
: 빈으로 들어오는 연결선을 모두 표시합니다.

나가는 연결 보기
: 빈에서 나가는 연결선을 모두 표시합니다.

연결 보기
: 빈으로 들어오거나 빈에서 나가는 연결선을 모두 표시합니다.

모두 보기
: 연결선을 모두 표시합니다.

들어온 연결 숨기기
: 빈으로 들어오는 연결선을 모두 숨깁니다.

나가는 연결 숨기기
: 빈에서 나가는 연결선을 모두 숨깁니다.

연결 숨기기
: 빈으로 들어오거나 빈에서 나가는 연결선을 모두 숨깁니다.

모두 숨기기
: 연결선을 모두 숨깁니다.

연결 조정

연결을 선택하면 선택 핸들이 연결선을 따라 양쪽 끝에 표시됩니다. 가운데 선택 핸들을 새로운 위치로 끌어 놓을 수 있습니다. 그렇게 하면, 빈 작업 영역의 다른

영역에 연결선이 그려지는데, 이를 통해 근접해 있는 몇 개의 연결선을 쉽게 구별할 수 있습니다. 추가 선택 핸들이 나타나면, 중간 선택 핸들을 새로운 위치로 끌어서 연결선을 더 구부릴 수 있습니다.

연결선을 원래 모양으로 복원할 수 있습니다. 연결선에 대한 팝업 메뉴에서 **모양 복원**을 선택하십시오.

연결 끝점 변경

VisualAge for Java는 연결의 끝점 빈을 변경할 수 있는 기능을 제공합니다. 즉, 사용자가 연결의 소스 또는 대상 빈을 변경할 수 있습니다. 이 작업의 속도가 연결 삭제 후 새로운 연결 작성의 경우보다 더 빠릅니다.

이벤트와 메소드 연결과 특성간 연결의 경우, 연결의 한쪽 끝을 이동할 수 있습니다. 이벤트와 코드 연결의 경우, 연결의 끝 이벤트만을 이동할 수 있습니다.

연결의 한쪽 끝을 변경하려면, 다음을 실행하십시오.

1. 변경하고 싶은 끝점을 포함한 연결을 선택하십시오. 선택 조정이 연결선을 따라 나타납니다.
2. 사용자가 변경하고 싶은 연결의 끝에 있는 선택 조정기 위로 마우스 포인터를 이동하십시오. 적절한 마우스 단추를 사용해서, 선택 조정을 새로운 빈으로 끌기 조작하십시오. OS/2에서는 마우스 오른쪽 단추를 사용하고, Windows에서는 마우스 왼쪽 단추를 사용하십시오.

   원래 연결에서 사용할 수 있는 동일한 특성을 포함하지 않은 빈으로 연결의 끝점을 이동할 경우, 사용자가 연결할 새로운 특성을 지정하도록 빈의 연결 팝업 메뉴가 나타납니다.

제8장 개정판 관리 소개

프로그램 개발의 이정표에 도달했으므로, 이제 몇몇 새로운 기능 코딩을 시작할 수 있습니다. 아마도 이미 작동하는 메소드의 상이한(아마도 더 효율적인) 구현을 탐색하길 원할 것이지만, 변경 또는 추가로 새로운 문제가 발생할지 여부를 확신하지 못할 것입니다. 지금이 사용자 코드의 버전화된 개정판을 작성하기에 적당한 때입니다.

VisualAge for Java에서 프로그램 구성요소의 다중 개정판을 관리할 수 있습니다. 개정판에 대한 몇 가지 개념은 이미 보았습니다. 이 절에서는 이러한 개념을 간략하게 검토해 보고 VisualAge for Java의 개정판 관리 기능 사용 방법을 보여 줍니다.

이 절에서는 다음을 배우게 됩니다.

- 『개정판에 대한 정보』
- 94 페이지의 『개정판 버전화』
- 94 페이지의 『코드 다시 변경』
- 100 페이지의 『이전 개정판으로 복귀』
- 101 페이지의 『저장소 탐색』

[엔터프라이즈] 개정판은 개발자 팀에서 관리합니다. 패키지 및 프로젝트는 소유자만이 버전화할 수 있으며, 클래스는 개발자만이 버전화할 수 있습니다. 개정판 버전화 다음에는 릴리스화가 자주 수행됩니다. 팀 개발 환경에 대한 자세한 정보는 VisualAge for Java, Enterprise Edition의 *시작하기 전에* 서적을 참고하십시오.

개정판에 대한 정보

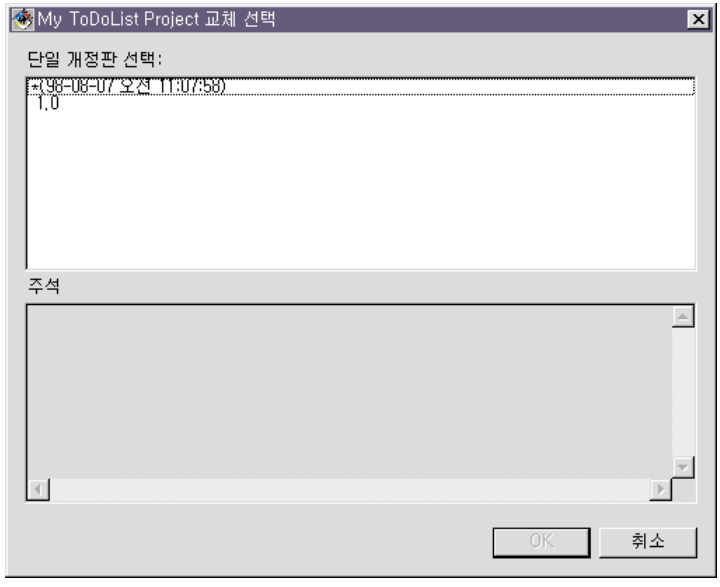
프로그램 구성요소를 저장하고 있을 때 VisualAge for Java에서는 사용자 코드를 계속 추적합니다. 실제로 작업하는 코드가 개정판에 저장됩니다. *개정판*은 특정 프로그램 구성요소의 일부를 떼어 놓은 "컷(cut)"이거나 "스냅샷"입니다.

작업하고 있는 개정판에 대한 자세한 내용을 보려면 Workbench 도구 막대를 사용하여 **개정판 이름 보기**  를 선택하십시오. 각 프로그램 구성요소가 영숫자 이름이나 또는 그 옆의 시간 소인을 포함한다는 것에 주의하십시오. 이것은 개정판 정보(아래에 더 자세히 기술된)입니다. 또한 소스 분할영역에서 동일한 정보를 볼 수도 있습니다. 예를 들어 ToDoList 클래스를 선택하고 마우스를 **소스** 분할영역 제목 막대에 있는 클래스 아이콘  으로 이동하십시오. 개정판 정보를 표시하는 팝업 도움말 창이 나타납니다. 개정판 정보는 **소스** 분할영역 아래에 있는 상태 영역에도 표시됩니다.

프로그램 구성요소의 개정판은 그 안의 프로그램 구성요소를 포함하여, 프로그램 구성요소 내 모든 코드의 트랙을 유지합니다. 예를 들어, 패키지의 개정판에는 클래스 및 인터페이스와 이러한 클래스 및 인터페이스 내의 메소드가 들어 있습니다.

언제든지 작업영역은 지정된 프로그램 구성요소에 대해 현재 작업 중인 하나의 개정판만을 포함할 수 있습니다. 프로그램 구성요소의 관리에 도움이 되도록 VisualAge for Java에는 동일한 프로그램 구성요소의 여러 개정판을 포함할 수 있는 소스 코드 저장소가 들어 있습니다. 작업영역은 VisualAge for Java 프로그래밍 환경에서 활동이 이루어 지는 중심부입니다. 이 저장소는 개발 환경은 아니지만, 필요할 경우 그 내용을 검색하여 가져올 수 있습니다. 원하는 만큼 프로그램 구성요소의 개정판을 많이 저장할 수 있습니다. 모든 개정판은 저장되며 저장소로부터 접근가능합니다.

작업영역에 있는 개정판을 저장소의 다른 개정판으로 대체할 수 있습니다. 저장소에서 개정판 목록을 검색할 때 현재 개정판 이름 왼쪽에 항상 * 기호가 기본값으로 표시됩니다.



개정판의 기본 유형에는 두 개가 있습니다.

- 개방 판

  프로그램 구성요소의 개방 판은 수정될 수 있습니다. 이 개방 판을 현재 개방 판으로 만들면서 작업영역으로 가져가 필요한 대로 변경할 수 있습니다. 위의 화면 이미지에서 개방 판은 시간 소인에 의해 표시됩니다. 예를 들어 (13/06/97 10:25:34 AM)은 개방 판입니다.

- 버전화된 개정판

  프로그램 구성요소의 버전화된 개정판은 변경될 수 없습니다. 개정판 버전화시, 언제든 복귀할 수 있는 고정 (읽기 전용) 코드 기본을 구성하십시오. 화면 이미지에서 버전화된 개정판은 Beta 2 또는 1.1같이 영숫자 이름으로 지정됩니다.

  사용자가 변경하고 자동적으로 저장한 변경이 새로운 개방 판을 작성하더라도, 작업영역에 있는 개정판은 버전화된 개정판일 수 있습니다.

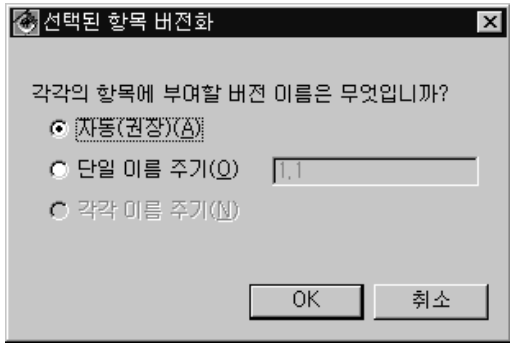
프로그램 구성요소 저장시, 이면에서 사용자의 코드가 증분식으로 컴파일될 뿐 아니라 개방 판이 작업영역과 저장소 둘다에서 갱신됩니다.

개정판 버전화

프로젝트, 패키지 또는 클래스를 버전화할 수 있습니다. 이러한 프로그램 구성요소 중 하나를 버전화할 때 그 안에 포함된 모든 프로그램 구성요소도 버전화됩니다. 예를 들어, 패키지를 버전화할 경우, 이러한 패키지의 파트인 모든 클래스도 버전화됩니다.

코드의 버전화 개정판을 작성해봅시다.

1. To-Do List 애플릿을 작성한 패키지를 선택하십시오. **선택** 메뉴에서 **관리**를 클릭한 다음 **버전**을 클릭하십시오. 선택된 항목 버전화 SmartGuide가 나타납니다.



2. **자동** 라디오 단추가 선택되었는지 확인한 다음 **OK**를 선택하십시오.

Workbench 계층에서, 패키지명 옆에 있는 시간 소인이 새로운 버전 번호로 대체되는 것에 주의하십시오. 이 버전화된 개정판은 이제 작업영역에서 사용자의 개정판에 어떤 일이 발생하는가에 상관없이 저장소에 영구적으로 저장됩니다. 이 버전화 개정판에 기초하여 개방 판을 열 수 있으며 버전화된 개정판은 항상 저장소로부터 사용할 수 있습니다.

코드 다시 변경

저장소에서 프로그램의 버전화된 개정판을 가지게 되었으므로, 이제 항상 버전화된 개정판으로 다시 변환할 수 있도록 작업영역에서 프로그램 구성요소를 변경할 수 있습니다.

새로운 개정판 작성

버전화된 개정판은 수정할 수 없기 때문에 프로그램 구성요소를 변경하려면 먼저 버전화된 개정판으로부터 새로운 개방 판을 작성해야 합니다. 작업영역에 있는 개정판이 버전화된 개정판일 경우, 프로그램 구성요소를 변경하고 이를 저장할 때 새로운 개정판이 자동적으로 작성됩니다. 예를 들어,

1. Workbench에서 ToDoList 클래스를 선택하고 소스 분할영역에 새로운 주석을 입력하십시오.
2. 소스 분할영역의 팝업 메뉴에서 **저장**을 선택하십시오.

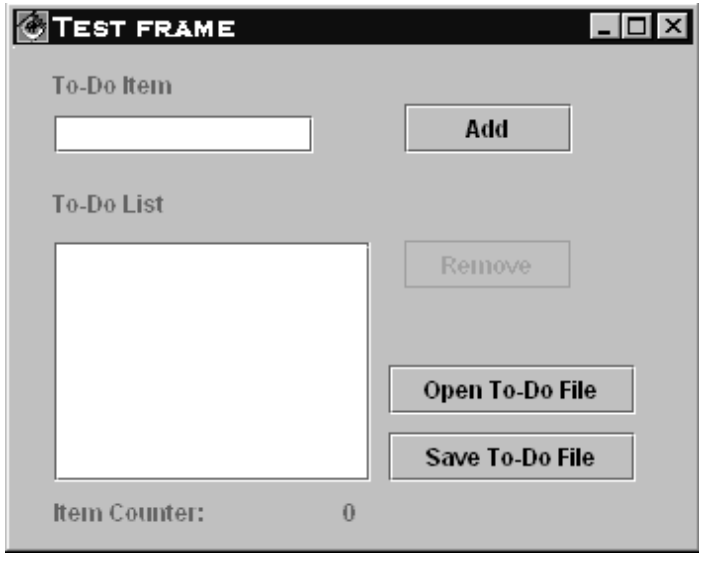
개정판 이름(계층 분할 영역 내)이 버전화된 개정판에서 시간 소인으로 변하는 것에 주의하십시오. 작업영역이 언제든 주어진 때에 프로그램 구성요소의 1개의 개정판만을 유지할 수 있기 때문에, 새로운 개정판이 버전화된 개정판을 대체합니다.(물론, 버전화된 개정판의 사본은 항상 저장소로부터 검색할 수 있습니다.)

ToDoList 프로그램에 카운터 추가

이제 To-Do List 프로그램에 카운터를 추가하십시오. 이 카운터는 지정한 때에 To-Do List에 있는 항목 수를 표시합니다. 이 기능을 추가하려면, 애플릿을 다음과 같이 변경해야 합니다.

1. 카운터 이름과 카운터 자체의 레이블을 추가하십시오.
2. 카운터 레이블에 **Add**, **Remove**, **Open To-Do File** 단추를 연결하십시오.

수정하면 실행중인 애플릿은 다음과 같이 나타납니다.



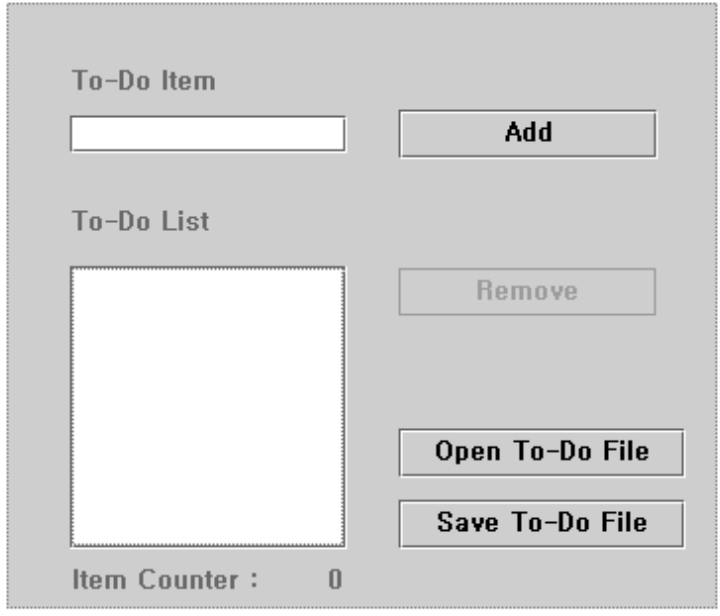
레이블 추가

비주얼 컴포지션 편집기를 사용하여 두 레이블을 추가하려면, 다음을 실행하십시오.

1. Workbench에서 ToDoList 클래스를 선택하십시오.
2. **선택** 메뉴에서 **선택 열기**를 선택한 다음 **비주얼 컴포지션**을 선택하십시오. 이것은 비주얼 컴포지션 편집기에서 ToDoList 클래스를 엽니다.
3. 새로운 연결을 쉽게 작성하려면 연결 숨기기  를 선택하여 기존 연결을 숨기십시오.

4. 팔레트에서 JLable 빈을 선택하십시오.
5. 목록 아래에서 마우스 왼쪽 단추를 클릭하여 레이블을 추가하십시오. 화면이동 분할영역, text 필드 및 label을 선택하여 조금 위로 이동함으로써 새로운 레이블을 위한 공간을 만들 수 있습니다.
6. JLabel 빈의 텍스트를 `Item Counter`로 수정하십시오.
7. 또다른 JLabel 빈을 방금 추가한 JLabel의 오른쪽에 추가하십시오.
8. 이 새로운 JLabel을 두 번 클릭하여 특성 창을 여십시오. **horizontalAlignment** 필드의 오른쪽에 있는 필드를 선택하십시오. 풀다운 메뉴에서 값을 오른쪽으로 정렬시키는 **RIGHT**를 선택하십시오. **text** 필드에서 값을 카운터의 초기 값인 **0**으로 변경하십시오. 특성 창을 닫으십시오.
9. 두 개의 새로운 JLabel 빈을 정렬하십시오.
   - 카운터명 레이블을 선택하고 text 필드를 선택한 후, **왼쪽 정렬** 도구를 선택하십시오.
   - 카운터 레이블을 선택하고 text 필드를 선택한 후, 도구 막대에서 **오른쪽 정렬** 도구를 선택하십시오.
   - 카운터명 레이블을 선택하고 카운터 레이블을 선택한 다음 **가운데 정렬**을 선택하십시오.

비주얼 빈이 추가 및 정렬되었습니다. 빈 작업 영역은 다음과 같이 나타나야 합니다.



이제 연결을 추가할 준비가 되었습니다.

주: 사용자가 만든 다른 연결이 모두 여기에 있지만 **연결 숨기기**를 선택했기 때문에 지금은 표시되지 않습니다. 다음 단계에서 작성할 새로운 연결은 숨겨지지 않습니다.

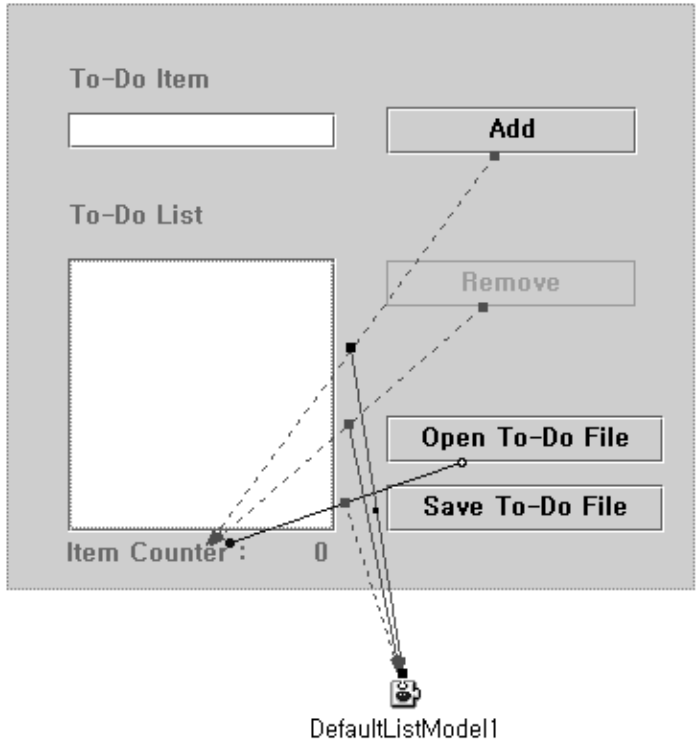
레이블 연결

카운터에 **Add** 단추를 연결하려면 다음을 수행하십시오.

1. **Add** 단추를 선택하고 마우스 오른쪽 단추를 클릭하십시오. 팝업 메뉴에서 **연결**을 선택한 다음 **actionPerformed**를 선택하십시오.

2. 카운터 레이블 위에 마우스를 놓고 마우스 왼쪽 단추를 클릭하십시오.
3. 팝업 메뉴에서 **text**를 선택하십시오. 이제 점선의 green 행이 나타나 완료되지 않은 연결을 지시합니다.
4. 연결을 선택하고 마우스 오른쪽 단추를 클릭하십시오. 팝업 메뉴에서 **연결**을 선택하고 **value**를 선택하십시오.
5. DefaultListModel 빈 위에 마우스를 놓고 마우스 왼쪽 단추를 클릭하십시오.
6. 팝업 메뉴에서 **연결가능 기능**을 선택하여(DefaultListModel1) 창으로 마지막 연결을 가져오십시오.
7. **특성** 목록에서 *크기*를 선택한 다음 **OK**를 선택하십시오. 이것은 항목 목록의 카운트를 카운터 문자열 설정용 입력으로 제공합니다. 이제 연결이 완료되었습니다.
8. 같은 방식으로 **Remove** 단추와 **Open To-Do File** 단추를 연결하십시오. 목록에 있는 항목의 카운트를 수정할 수 있는 조치가 취해질 때마다 목록 카운트를 갱신하기만 하면 됩니다. 이 애플릿에서 맨 위 세 단추 모두가 이 잠재성을 포함합니다.

이제 빈 작업 영역은 다음과 같이 나타납니다.



빈 메뉴에서 **빈 저장**을 선택하십시오. 사용자가 변경한 사항은 이 개방 판, 작업 영역과 저장소 모두에 반영됩니다. 도구 막대에서 **수행** 도구를 선택하여 애플릿 열람기를 시작하고 실제 표시되는 카운터를 보십시오.

이전 개정판으로 복귀

이제 사용자의 프로그램은 카운터를 포함합니다. 이것은 적절히 작동하지만, 잠시 이에 대해 생각해 보면 인터페이스를 되도록 깨끗하게 유지하는 것(어떤 특수 기능도 없이)이 낫다고 결정하게 될 것입니다. 따라서 카운터 코드를 없애려고 합니다. 물론, 사용자가 추가한 레이블과 연결을 삭제할 수 있지만, 실수로 기타 프로그램 구성요소나 연결 중 하나를 삭제할 경우 어떻게 해야 할까요? 이전 개정판을 버전화했으므로 염려하지 않아도 됩니다.

다음 단계를 따라 현재 개정판을 저장소에서 얻은 이전 개정판으로 대체하십시오.

1. Workbench에서 ToDoList 클래스를 선택하고 마우스 오른쪽 단추를 클릭하십시오.
2. 팝업 메뉴에서 **교체**를 선택한 다음 **다른 개정판**을 선택하십시오.
3. **ToDoList에 대한 교체 선택** 보조 창에서 이전에 버전화한 개정판을 선택한 다음 **OK**를 클릭하십시오.

   (현재 개정판을 이전 개정판으로 바꾸려는 것이므로 **교체**를 선택한 다음 팝업 메뉴에서 **이전 개정판**을 선택할 수도 있습니다.)

클래스명 옆에 있는 개정판 정보는 이제 사용자가 작업중이던 개방 판의 시간 소인이 아닌 버전 번호를 지시합니다.

마음을 다시 바꿔 카운터를 여전히 포함하기로 할 경우, 언제든 이를 다시 추가할 수 있습니다. 카운터를 포함했던 개정판은 여전히 저장소에 있습니다.

저장소 탐색

편집-컴파일-디버그 도구 외에, VisualAge for Java는 견고한 코드 관리 기능도 제공합니다. 프로그램 구성요소의 다중 개정판에 대해 쉽게 작업하는 방법을 살펴보았습니다. 그러면 이번에는 저장소에서 얻을 수 있는 기타 사항에 대해 알아보겠습니다.

창 메뉴에서 **저장소 탐색기**를 선택하십시오.

저장소 탐색기는 사용자 저장소의 비주얼 인터페이스를 제공합니다. 저장소에는 프로그램 구성요소의 모든 개정판이 들어 있습니다. 이것은 현재 작업영역에 있는 모든 프로그램 구성요소를 포함합니다.

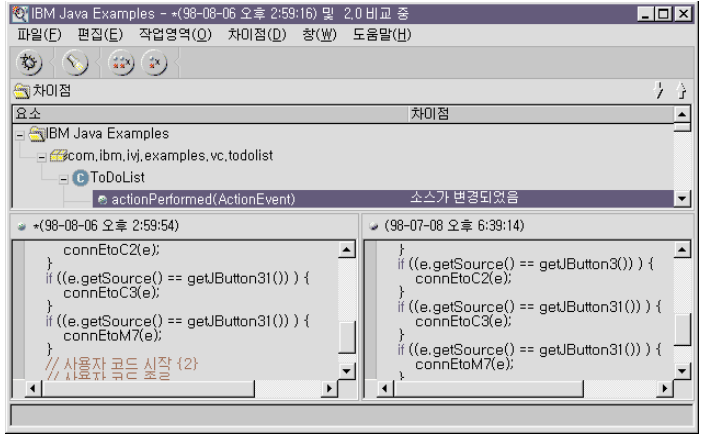
저장소 탐색기에서, 저장소에 저장된 프로그램 구성요소를 열거나 비교할 수도 있습니다. 이들을 열람 또는 비교하기 위해 작업영역 안으로, 그 밖으로 개정판을 교환할 필요가 없습니다.

상이한 개정판을 비교하여, 다음을 볼 수 있습니다.

- 코드 생성의 결과로 만들어진 변경
- 오류를 포함한 개정판이 정확히 버그 없는 개정판과 다른 방식

패키지의 두 개정판을 비교하려면, 다음을 실행하십시오.

1. **저장소 패키지** 페이지를 선택하십시오.
2. **패키지명** 목록에서 To-Do List 애플릿을 작성한 패키지를 선택하십시오.
3. **개정판** 목록에서, 마우스 왼쪽 단추를 누른 채로 맨 위에 있는 두 개정판을 끌어서 선택하십시오. **개정판** 메뉴에서 **비교**를 선택하십시오. 비교 창이 나타납니다.



4. 구성요소 분할영역에서 클래스나 메소드 이름을 선택하십시오. 아래의 텍스트 분할영역에 해당하는 두 개의 코드 세트가 표시됩니다. 여기에서 두 프로그램 구성요소를 비교할 수 있습니다. **차이점** 풀다운 메뉴에서, **다음 차이점**과 **이전**

차이점을 선택할 수 있습니다. 창의 오른쪽 위 모서리에 있는 화살표  를 선택하여 차이점 목록에서 앞뒤로 이동할 수 있습니다.

작업영역에 있는 모든 프로그램 구성요소는 별표(*)로 표시됩니다.

저장소에서 예제 점검

VisualAge for Java에는 다양한 예제 코드가 함께 들어 있습니다. 저장소 탐색기 창을 사용해서 이러한 예제를 점검하십시오. 예를 들어 저장소 탐색기에서 To-Do List 프로그램의 완전한 버전을 점검하려면, 다음을 수행하십시오.

1. **저장소 프로젝트** 페이지를 선택하십시오. **프로젝트 이름** 목록에서 **IBM Java Examples**를 선택하십시오.
2. **개정판** 목록에서 개정판을 하나 선택하고 **패키지** 목록에서 **com.ibm.ivj.examples.vc.todolist**를 선택하십시오. 이 패키지의 클래스는 **유형** 목록에 나타납니다. 이 클래스를 점검하려면 클래스를 선택한 다음 마우스 오른쪽 단추를 클릭하여 팝업 메뉴에서 **열기**를 선택하십시오.

사용자가 이같은 완전한 샘플을 실행하거나 또는 이들을 갱신하길 원한다고 가정해 보십시오. 먼저, 이들을 작업영역으로 가져와야 합니다. 예를 들어 완전한 To-Do List 프로그램 버전을 작업영역으로 가져오려면 다음을 수행하십시오.

1. Workbench에서 To-Do List 프로그램에 추가할 프로젝트를 선택하고 **추가**를 선택한 다음 **선택** 메뉴에서 **패키지**를 선택하십시오. 패키지 추가 SmartGuide가 나타납니다.
2. **저장소에서 패키지 추가**를 선택하십시오.
3. **사용할 수 있는 패키지명** 목록에서 **com.ibm.ivj.examples.vc.todolist**를 선택하십시오. **개정판 목록**에서 개정판을 선택하고 **완료**를 클릭하십시오.

To-Do List 프로그램용 패키지가 사용자의 작업영역에 추가되며 사용자는 이를 갱신하고 수행할 수 있습니다.

요약

저장소 및 사용자 프로그램 구성요소의 다중 개정판에 대해 작업하는 기능으로, 코드 관리는 쉬워집니다. VisualAge for Java는 사용자를 올바른 방향으로 인도해 줍니다.

제9장 IDE에서 할 수 있는 기타 작업

VisualAge for Java에서 실행할 수 있는 흥미로운 여러가지 작업을 이미 보았지만, 이외에도 사용가능한 기능이 더 있습니다. 이 절에서는 VisualAge for Java의 다음 기능에 대해 더 자세한 내용을 제공합니다.

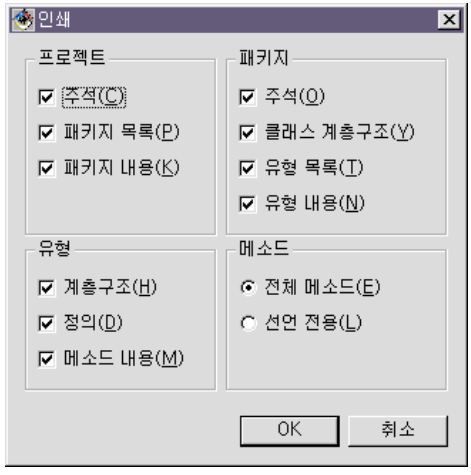
- 『프로그램 구성요소 인쇄』
- 107 페이지의 『네비게이트』
- 111 페이지의 『탐색』
- 115 페이지의 『검색』
- 120 페이지의 『직접 코드 작성』
- 130 페이지의 『국제화 지원』
- 131 페이지의 『빠른 시작 창 사용』
- 132 페이지의 『디버깅』
- 139 페이지의 『JavaBeans 컴포넌트 지원』
- 142 페이지의 『작업영역 조정』

프로그램 구성요소 인쇄

VisualAge for Java는 프로그램 구성요소 인쇄용으로 몇가지 옵션을 제공합니다. 프로젝트, 패키지, 클래스, 인터페이스 또는 메소드를 인쇄할 수 있습니다. 기타 프로그램 구성요소로 구성되는 프로그램 구성요소 인쇄시, 이같은 기타 프로그램 구성요소를 인쇄하는 옵션을 가지게 됩니다. 예를 들어, 패키지 인쇄시, 패키지 내의 클래스를 인쇄할 수도 있습니다.

프로그램 구성요소를 인쇄하려면, 다음을 실행하십시오.

1. 프로그램 구성요소를 선택하십시오. **문서**를 선택한 다음 팝업 메뉴에서 **인쇄**를 선택하십시오. 인쇄 대화 상자가 나타납니다.



2. 인쇄하기 위해 선택할 수 있는 항목은 인쇄중인 프로그램 구성요소의 종류에 따라 달라집니다.
   - **프로젝트** 아래 있는 선택사항은 프로젝트를 인쇄할 때 사용할 수 있습니다.
   - **패키지** 아래 있는 선택 사항은 프로젝트나 패키지를 인쇄할 때 사용할 수 있습니다.
   - **유형** 아래 있는 선택 사항은 프로젝트, 패키지, 클래스 또는 인터페이스를 인쇄할 때 사용할 수 있습니다.
   - **메소드** 아래 있는 선택 사항은 항상 사용할 수 있습니다.
3. 기본값으로 **프로젝트**, **패키지** 및 **유형** 아래의 모든 항목이 선택되며, **메소드** 아래에서는 **전체 메소드**가 선택됩니다. 필요하면 선택사항을 변경하고 **OK**를 선택하여 인쇄를 시작하십시오.
4. 기본 프린터가 선택되어 있지 않으면, 기본 프린터를 하나 선택하라는 메시지가 나타납니다.

기본 프린터 변경

창의 **파일** 메뉴에서 **인쇄 설정**을 선택하여 기본 프린터를 변경하거나 프린터의 설정을 변경할 수 있습니다.

클래스 계층구조의 그래프 인쇄

프로젝트 또는 패키지에 대한 클래스 계층구조의 그래프 열람을 인쇄할 수 있습니다. 출력은 기본 프린터에 인쇄됩니다. 클래스 계층구조 그래프를 인쇄하려면 다음을 수행하십시오.

1. Workbench 또는 다른 검색기에서 그래프를 인쇄하려는 프로젝트 또는 패키지를 선택하십시오.
2. 구성요소 팝업 메뉴에서 **열기**를 선택한 다음 **클래스**를 선택하십시오. 그러면 프로젝트 또는 클래스에 대한 검색기의 클래스 페이지가 열립니다.
3. 클래스 계층구조 분할영역 제목 막대에서 그래프형 배치 단추  를 클릭하십시오.
4. 클래스 계층구조 분할영역 팝업 메뉴에서 **문서**를 선택한 다음 **그래프 인쇄**를 선택하십시오. 각 클래스의 상속 관계를 보여 주는 그래프가 기본 프린터에 인쇄됩니다.

네비게이트

VisualAge for Java는 사용자의 코드를 보는 여러가지 방법을 제공합니다. 이 절에서는 VisualAge for Java 내의 기본창에 대한 간단한 개요를 제공하며 창 사이로 이동하는 방법을 알려 줍니다.

창 사이 이동

VisualAge for Java의 모든 창에는 **창** 메뉴가 있습니다. 이 메뉴에서 원하는 창을 선택하여 창 사이로 이동할 수 있습니다.

선택한 창이 이미 열려 있는 경우, 이것은 활성 창이 됩니다. 사용자가 원하는 창이 열려 있지 않은 경우, 이 창이 열리며 활성 창이 됩니다. **창** 메뉴에서 **창 전환**을 선택하면 현재 열려 있는 창 중에서 하나를 선택하여 그 창으로 이동할 수 있습니다.

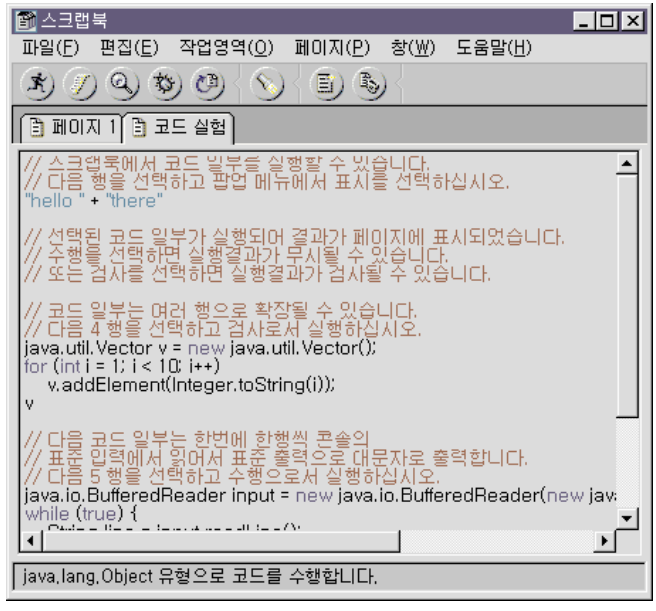
최근에 사용한 창은 **파일** 메뉴의 목록에 저장됩니다. 최근에 닫은 창을 다시 열려면 **파일** 메뉴의 목록에서 해당 창을 선택하십시오.

사용자가 명시적으로 연 것 외에, 사용자가 자신의 타스크를 수행할 때 VisualAge for Java가 연 창도 몇 개 있습니다. 예를 들어 Workbench 창에서 클래스를 선택하고 **선택** 메뉴에서 **수행**을 선택하여 프로그램을 실행한다고 가정합니다. 사용자 프로그램에 활성 중단 시점이 있다면, 중단점에 도달할 때 디버거 창이 열립니다. Workbench 창으로 돌아가려면 디버거 창의 **창** 메뉴에서 **Workbench**를 선택하십시오.

창 메뉴에서 열 수 있는 창

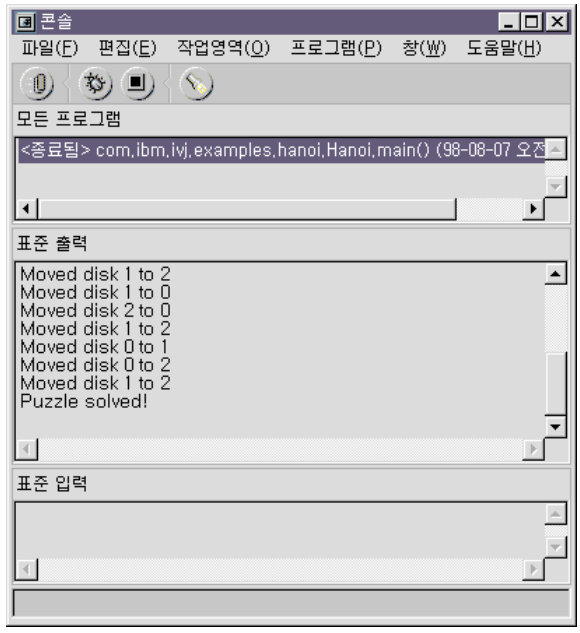
창 메뉴에서 다음 창을 열 수 있습니다.

- **스크랩북**: 코드를 시험하는 장소입니다. 프로젝트나 클래스 일부를 만들지 않고도 코드 일부를 입력하고 실행할 수 있습니다.

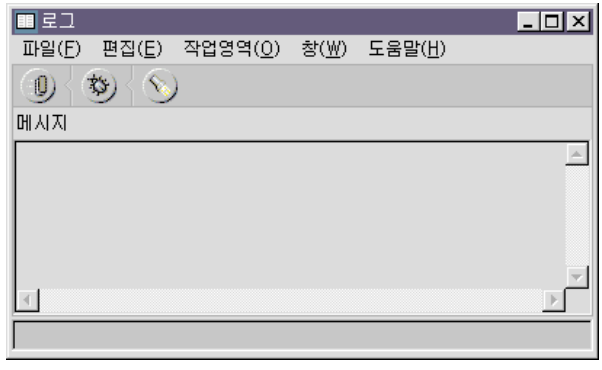


- **콘솔**: 표준 출력을 표시합니다. 또한 표준 입력에 입력할 영역도 제공합니다. 표준 입력으로부터 입력을 기다리는 스레드가 둘 이상 있는 경우에는 입력을 가

져올 스레드를 선택할 수 있습니다.

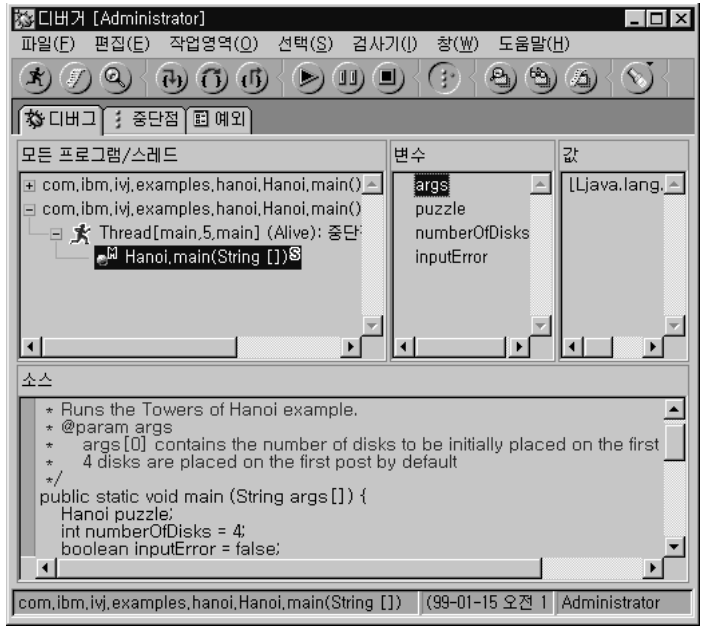


- 로그: VisualAge for Java가 보내는 메시지와 경고를 표시합니다.



- **디버거**: 수행중인 스레드와 런타임 스택 내용을 표시합니다. 디버거에서 스레드의 실행을 일시 중단했다가 실행을 재개하고, 변수 값을 검사 및 수정하고 중단점을 설정, 제거 및 구성할 수 있습니다. **창** 메뉴를 사용하면 디버거 검색기를 디버그 페이지나 중단점 페이지로 열거나, 외부 클래스 메소드와 포착 예외에 중단점을 설정할 수 있는 대화 상자를 열 수 있습니다. 자세한 내용은 132 페이

지의 『디버깅』을 참조하십시오.



- **저장소 탐색기**: 저장소 내 프로그램 구성요소 개정판을 모두 표시합니다. 자세한 내용은 101 페이지의 『저장소 탐색』을 참조하십시오.

이들 창 외에도 **창** 메뉴에서는 다음 조치를 수행할 수 있습니다.

- **복제**를 클릭하면 현재 창의 복제 창이 열립니다. 그런 다음 두 창을 독립적으로 검색할 수 있습니다. 프로그램 구성요소에 대한 변경은 두 창 모두에 반영됩니다.
- **잠금**을 클릭하여 열린 현재 창을 잠급니다. 창을 닫으려 하면, 창이 잠겨 있음을 알리는 메시지 박스가 나타납니다. 먼저 창을 잠금해제해야 닫을 수 있습니다.
- **분할영역 최대화**를 클릭하면 현재 창을 전체 화면 크기로 조정합니다.
- **방향 전환**은 창에서 분할 영역의 일반 배치를 변경합니다. 예를 들어, 이 정보에서 이미지는 IDE에서 선택적으로 수직 방향으로 변경할 수 있는 수평 방향을 표시합니다.
- **개정판 이름 보기**를 클릭하면 프로그램 구성요소에 대한 개정판 이름 레이블을 사용하거나 사용하지 않을 수 있습니다.
- **Workbench**를 클릭하여 Workbench 검색기를 열거나 초점을 가져옵니다.

탐색

VisualAge for Java는 사용자가 프로그램 구성요소를 쉽게 찾고 인터페이스 내에서 이동할 수 있도록 디자인되었습니다. 이 절에서는 IDE의 탐색 기능 활용 방법을 알려 줍니다.

프로그램 구성요소 탐색

IDE는 프로그램 구성요소 탐색용 선택사항을 몇가지 제공합니다. 예를 들어 프로그램 구성요소 분할영역에서 문자 키를 누를 경우, VisualAge for Java가 그 문자로 시작되는 첫번째로 표시된 프로그램 구성요소를 선택합니다. 동일한 문자를 다시 누르면, 그 문자로 시작되는 다음 프로그램 구성요소가 선택됩니다.

탐색 대화 상자를 사용한 탐색

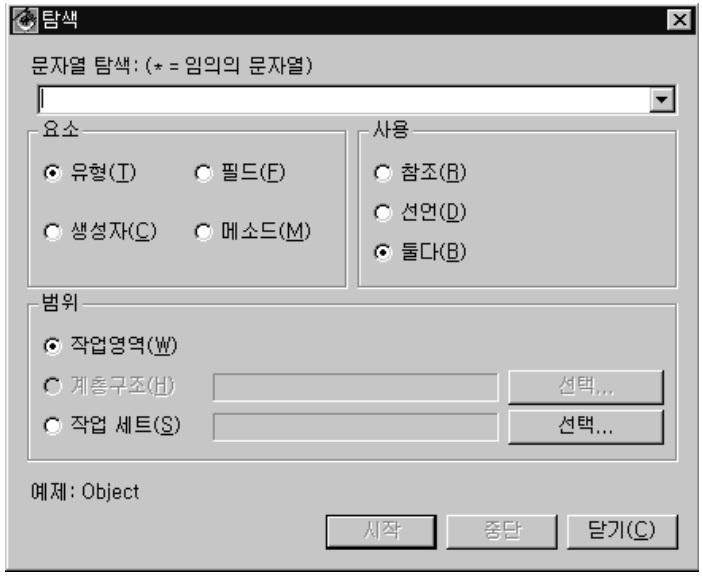
탐색 대화 상자를 사용하여 작업영역의 강력한 탐색을 수행할 수 있습니다. 탐색 대화 상자를 열려면 다음 메소드 중 하나를 사용하십시오.

- 창의 **작업영역** 메뉴에서 **탐색**을 선택하십시오.

- 소스 분할영역에서 텍스트를 선택한 다음 **편집** 메뉴에서 **탐색**을 선택하십시오.

- 도구 막대에서 탐색 단추  를 선택하십시오.

탐색 대화 상자가 열립니다. 탐색을 실행하기 전에 프로그램 구성요소나 텍스트를 선택했으면 탐색 문자열 필드에 선택한 항목이 입력됩니다.



적절한 라디오 단추를 사용가능화하여 탐색하려는 프로그램 구성요소의 유형, 탐색 범위 및 구성요소의 용도를 선택하십시오. **시작**을 클릭하십시오. 탐색이 완료되

면 IDE가 찾는 기준에 일치하는 항목을 찾은 경우 탐색 결과 창이 열립니다.

탐색 결과 창에서 포함된 프로그램 구성요소를 검색하고 수정하며 탐색을 재수행할 수 있습니다.

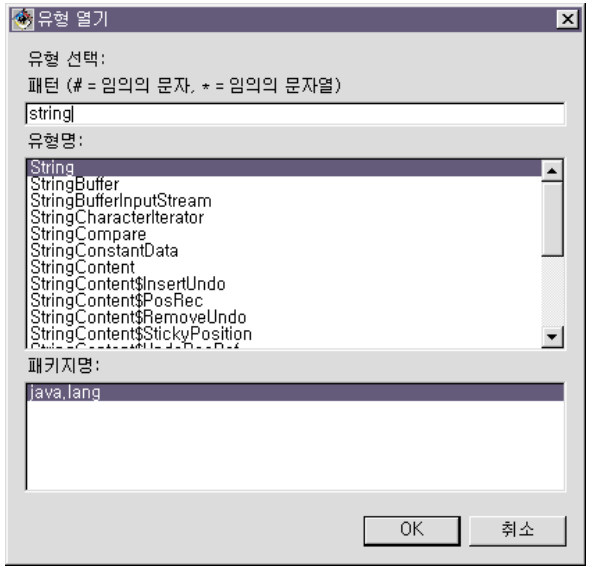
참조 및 선언 탐색

유형 및 메소드에 대한 팝업 메뉴에는 자주 요구되는 특수한 탐색이 포함됩니다. 클래스 및 인터페이스의 경우, **참조** 팝업 메뉴 옵션에는 선택된 유형 또는 그 필드 중 하나에 대한 참조를 작업영역에서 찾는 하위 옵션이 있습니다. 탐색 결과는 탐색 결과 창에 표시됩니다.

메소드의 경우, **참조** 팝업 메뉴 옵션에는 선택된 메소드, 호출되는 메소드, 액세스되는 필드 또는 참조되는 유형에 대한 참조를 작업영역에서 찾는 하위 옵션이 있습니다. **선언** 하위 옵션은 같은 프로그램 구성요소의 선언을 탐색합니다. 탐색 결과는 탐색 결과 창에 표시됩니다.

작업영역 메뉴에서 탐색

작업영역 메뉴에서 **열기** 선택사항 중 하나를 선택하여 프로그램 구성요소를 탐색할 수도 있습니다. 이 탐색을 수행하면 결과적으로 탐색된 구성요소에 대한 검색기가 열립니다. 예를 들어 **작업영역** 메뉴에서 **유형 검색기 열기**를 선택하면, 유형 열기 대화 상자가 표시됩니다.



패턴 필드에 `string`을 입력하면 **유형명** 목록이 갱신되어 지금까지 사용자가 입력한 탐색 기준과 일치하는 클래스와 인터페이스만 보여 줍니다. **유형명** 목록에서 문자열을 선택한 다음 **OK**를 선택하여 String 클래스에 대한 검색기를 엽니다. 둘 이상의 패키지에 존재하는 일부 유형명의 경우에는 **패키지명**에서 패키지를 선택해야 합니다.

검색기 페이지에서 프로그램 구성요소 탐색

현재 검색기 페이지에 포함된 프로그램 구성요소를 찾으려면, 구성요소 유형의 **이동** 메뉴 옵션을 사용하십시오. 예를 들어 현재 프로젝트 페이지에 있고 java.lang.String 클래스를 찾으려면 **이동**을 선택한 다음 **선택** 메뉴에서 **유형**을 선택하십시오. 유형으로 이동 대화 상자에서 **패턴** 필드에 `string`을 입력한 다음 유

형 목록에서 **문자열**을 선택하고 패키지 목록에서는 **java.lang**을 선택하십시오. **OK**를 클릭하면 IDE가 이동하여 Workbench의 **모든 프로젝트**에 있는 java.lang.String 클래스를 선택합니다.

검색

VisualAge for Java는 프로그램 구성요소 검색을 위한 광범위한 기능을 제공합니다.

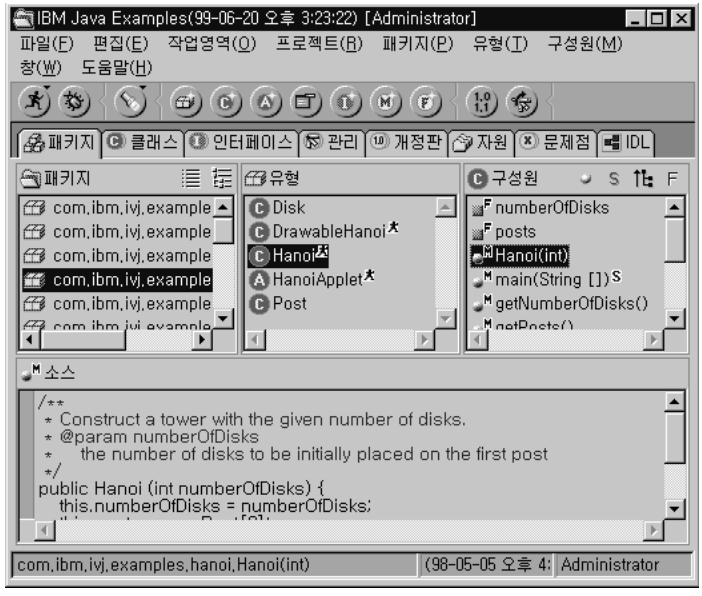
IDE에서는 프로그램 구성 요소를 열어 검색합니다. VisualAge for Java에서 프로그램 구성요소를 열 수 있는 방법이 많이 있지만, 지금으로선 다음의 2가지 간단한 방법이 있습니다.

- 프로그램 구성 요소를 선택하고 **선택** 메뉴나 프로그램 구성 요소의 팝업 메뉴에서 **열기**를 선택하십시오.
- 해당 프로그램 구성요소의 **작업영역** 메뉴(**유형 검색기 열기**, **패키지 검색기 열기** 또는 **프로젝트 검색기 열기**)에서 적절한 검색기를 선택합니다. 작업영역 내 모든 클래스, 인터페이스, 패키지 또는 프로젝트를 나열하는 보조 창이 나타납니다. 프로그램 구성요소 이름을 입력하고 **OK**를 선택하십시오.

프로그램 구성요소를 열 때, 이 프로그램 구성요소에 대한 정보를 표시하는 창이 나타납니다. 다음 절은 각 프로그램 구성요소를 열 때 나타나는 창을 상세히 기술합니다.

프로젝트 검색

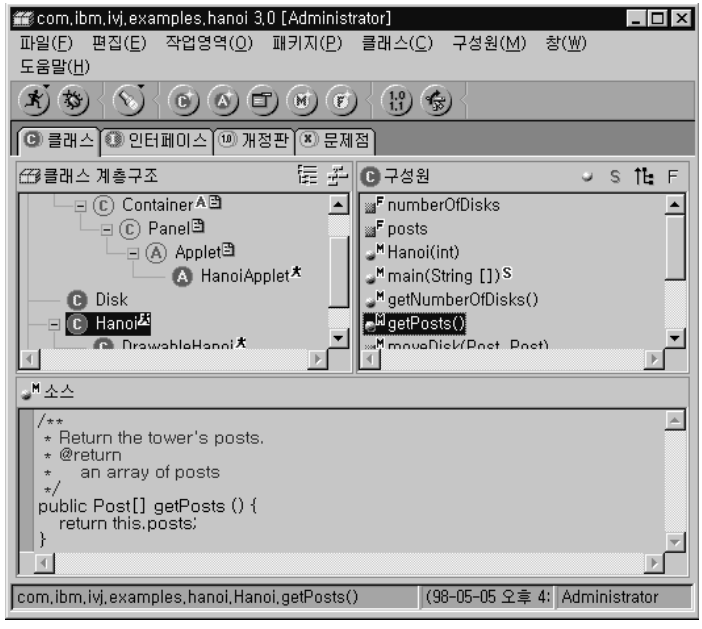
프로젝트를 열면 다섯 페이지로 이루어진 창이 열립니다.



- **패키지** 페이지는 이 프로젝트에 포함된 패키지의 계층구조를 표시합니다.
- **클래스** 페이지는 이 프로젝트에 포함된 클래스의 계층구조를 표시합니다.
- **인터페이스** 페이지는 이 프로젝트에 포함된 인터페이스를 표시합니다.
- **개정판** 페이지는 이 프로젝트의 모든 개정판을 표시합니다.
- **문제점** 페이지는 프로젝트에서 오류가 있는 프로그램 구성요소를 모두 나열합니다.

패키지 검색

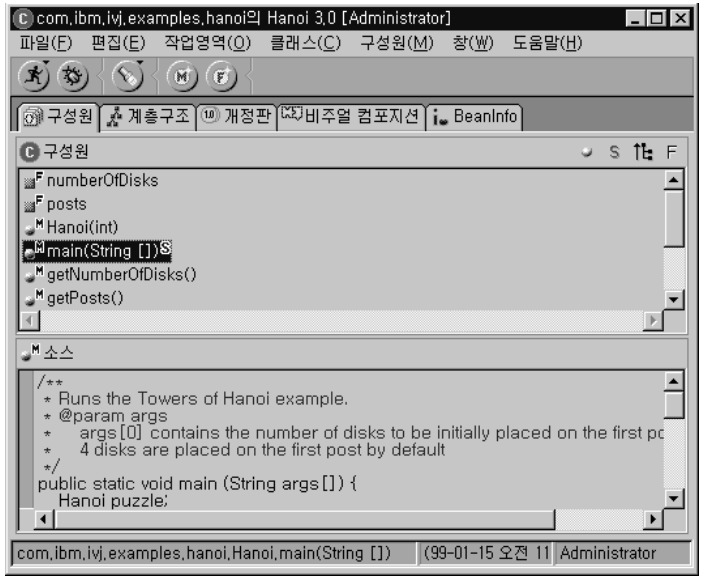
패키지를 열면 네 페이지로 이루어진 창이 열립니다.



- **클래스** 페이지는 이 프로젝트에 포함된 클래스의 계층구조를 표시합니다.
- **인터페이스** 페이지는 이 패키지에 포함된 인터페이스를 표시합니다.
- **개정판** 페이지는 이 패키지에 포함된 모든 개정판을 표시합니다.
- **문제점** 페이지는 패키지에서 오류가 있는 프로그램 구성요소를 모두 나열합니다.

클래스 검색

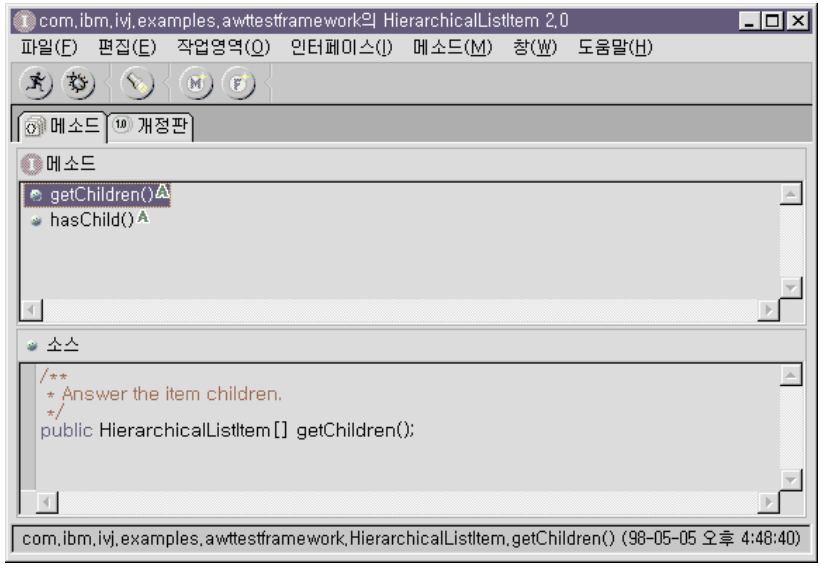
클래스를 열면 다섯 페이지로 구성된 창이 열립니다.



- **메소드** 페이지는 이 클래스에 포함된 메소드를 표시합니다.
- **계층구조** 페이지는 전체 클래스 계층구조에서 해당 클래스의 위치를 표시합니다.
- **개정판** 페이지는 이 클래스의 개정판을 표시합니다.
- **비주얼 컴포지션** 페이지는 비주얼 컴포지션 편집기를 표시합니다.
- **BeanInfo** 페이지는 이 클래스의 JavaBeans 특정 인터페이스에 대한 정보를 표시합니다.

인터페이스 검색

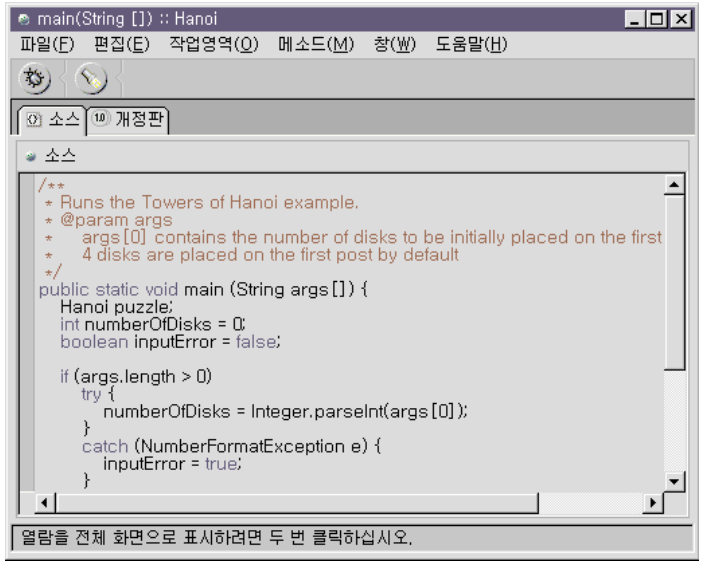
인터페이스를 열면 두 페이지로 구성된 창이 열립니다.



- **메소드** 페이지는 이 인터페이스에 포함된 메소드를 표시합니다.
- **개정판** 페이지는 이 인터페이스의 개정판을 표시합니다.

메소드 검색

메소드를 열면 두 페이지로 구성된 창이 열립니다.



- **소스** 페이지는 메소드 소스를 나열합니다.
- **개정판** 페이지는 이 메소드의 개정판을 표시합니다.

직접 코드 작성

시작하기 전에 서적을 따라 진행할 때 이제까지는 대부분 비주얼 컴포지션 편집기와 SmartGuide를 사용하여 Java 코드를 생성하거나 프로그램에서 제공하는 코드 영역을 추가하는 방법을 사용하였습니다. 사용자 소유 응용프로그램 작성시, IDE 검색기의 소스 분할영역에 직접 코드를 작성해야 합니다. VisualAge for Java는 직접 정확한 코드를 작성하는 데 도움이 되는 여러 도구를 제공합니다. 이 절에서는 간단한 직접 코드를 작성하는 예제를 보여 주고 여기에 사용되는 도구를 설명합니다.

코드 보조(Code Assist)

소스 분할 영역, 일부 대화 상자, 중단점 구성 창이나 스크랩북 창같은 검색기에는 클래스 라이브러리 참조 정보를 참고하지 않고 메소드, 클래스, 필드를 찾는 데 도움을 주는 *코드 보조*라는 도구가 있습니다. 코드 보조는 Ctrl+스페이스바를 눌러서 액세스합니다.

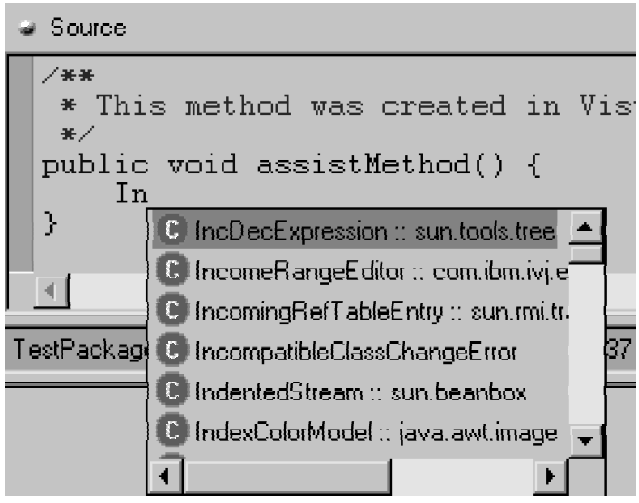
Ctrl+스페이스바를 입력하면, 커서에서 코드에 입력할 수 있는 클래스, 메소드 및 유형이 팝업 목록에 표시되고 사용자는 이 중에서 하나를 선택할 수 있습니다. 코드 보조는 시각적 점검을 수행하고 볼 수 없는 클래스, 메소드 및 필드는 표시하지 않습니다. 코드 보조 메카니즘이 커서의 현재 위치에 맞는 구성원을 찾을 수 없다면, 분할영역의 맨 아래에 있는 정보 행은 현재 문맥에 사용할 수 있는 코드 보조가 없음을 나타냅니다.

유형에 대한 코드 보조

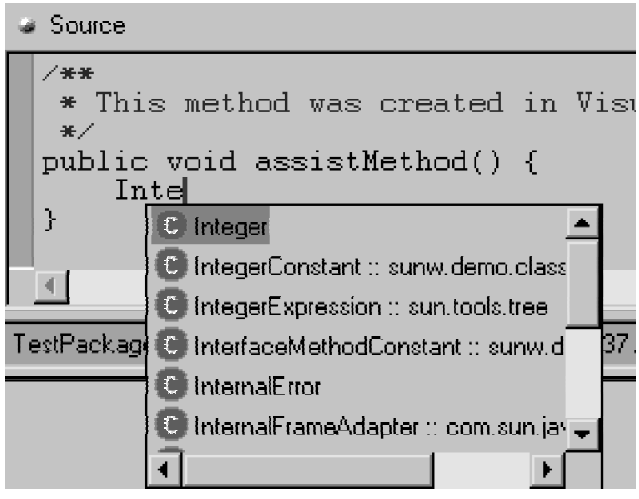
코드에 클래스 또는 인터페이스 이름을 삽입하려면, 유형명의 첫 문자 또는 여러 문자를 입력한 다음 Ctrl+스페이스바를 입력하십시오. 입력한 문자로 시작되는 유형이 팝업 목록에 나타납니다. 여러 문자를 입력하여 목록을 아래로 좁힐 수 있습니다. 항목을 선택하여 코드의 커서 위치에 삽입할 수 있습니다. 유형을 규정화해야 할 경우, 규정도 자동으로 삽입됩니다.

예제: 테스트 프로젝트와 패키지를 작성하십시오. 테스트 패키지에서 AssistTest라는 클래스를 작성하십시오. AssistTest 클래스에 assistMethod라는 메소드를 작성하십시오. 국지 정수 변수 i를 선언한다고 가정하면, assistMethod 소스 본문에 `In`을 입력하십시오.

Ctrl+스페이스바를 입력하십시오. 옵션 팝업 목록이 나타납니다.



사용할 수 있는 유형 목록이 long입니다. Integer를 찾으려면 te를 입력하십시오. Integer가 목록 맨 위에 나타납니다.



화살표 키를 사용하여 선택한 다음 Enter 키를 누르십시오.

이제 선언을 완료하였으므로 메소드는 다음과 같습니다.

```
public void assistMethod() {
    Integer i;
}
```

Ctrl+S 키를 입력하여 메소드를 저장하십시오. 다음 예제에서 이 테스트 메소드를 사용합니다.

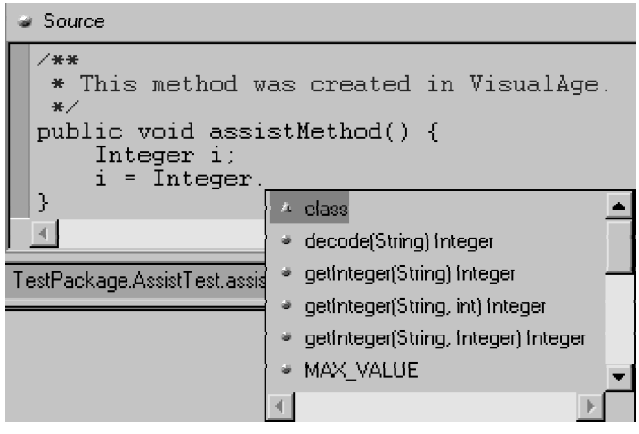
메소드와 필드에 대한 코드 보조

코드 보조는 객체 또는 클래스에 사용할 수 있는 메소드와 필드도 나열할 수 있습니다. 객체 이름과 마침표(.)를 입력하십시오. 메소드나 필드 이름에서 앞 글자를 몇개 입력한 다음 Ctrl+스페이스바를 누르십시오. 객체에 대한 메소드 및 필드 목록이 나타납니다. 하나를 선택하여 코드에 삽입하십시오.

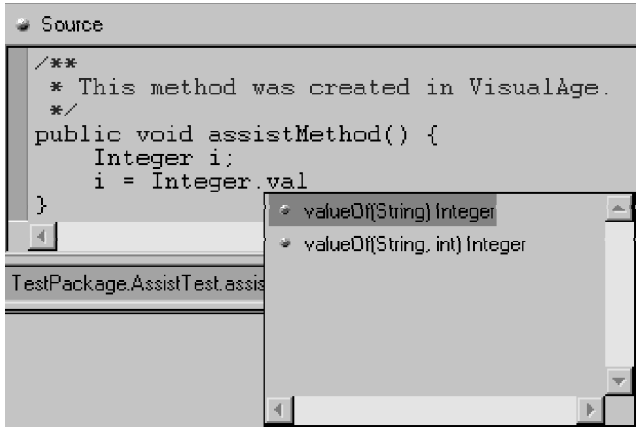
예제: 이전 예제에서 작성한 assistMethod 메소드에서 i를 선언한 명령행 아래에 다음 코드를 입력하십시오. 이 때 마침표에 주의하십시오.

```
i = Integer.
```

Ctrl+스페이스바를 입력하십시오. Integer 클래스의 메소드와 필드 목록이 나타납니다.



val를 입력하여 **valueOf(String) Integer**를 찾으십시오. 매개변수 유형(여기서는 String)과 리턴 유형(Integer)이 표시됩니다.



valueOf(String) Integer를 선택한 다음 Enter 키를 누르면 코드에 삽입됩니다. "35"처럼 괄호 안에 문자열을 입력하고 행 끝에 세미 콜론을 입력하십시오. 메소드 소스는 이제 다음과 같습니다.

```
public void assistMethod() {
    Integer i;
 i = Integer.valueOf("35");
}
```

규정화를 요구하는 클래스에서 메소드 또는 필드에 대한 코드 보조를 요청하는 경우, Ctrl+스페이스바를 누르기 전에 이 클래스가 규정화되어야 합니다. 그렇지 않으면 코드 보조를 사용할 수 없게 됩니다. 일반적으로 커서 앞에 나타나는 코드는 코드 보조를 요청하기 전에 컴파일할 수 있어야 합니다.

예제: 사용자 클래스 반입 명령문에 java.util.*이 없다고 가정합니다. 이는 클래스 ResourceBundle을 사용자 클래스에서 사용할 때 규정화해야 함을 의미합니다. 다음 코드를 입력한 다음 Ctrl+스페이스바를 입력하여 사용할 수 있는 메소드 목록을 가져오려는 경우, 목록을 사용할 수 없게 됩니다.

```
public String newMethod ( ) {
ResourceBundle a = ResourceBundle.
  // place cursor after period
  // and type Ctrl+Spacebar
```

그러나 클래스 규정이 제공된 다음 코드를 입력하면 코드 보조를 사용할 수 있습니다.

```
public String newMethod ( ) {
ResourceBundle a = java.util.ResourceBundle.
  // place cursor after period
  // and type Ctrl+Spacebar
```

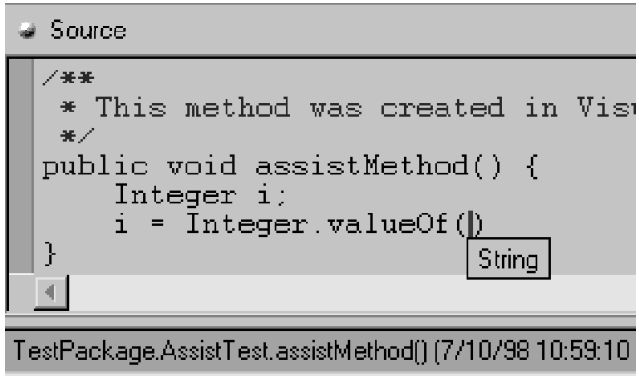
커서를 마침표 **앞**에 놓고 Ctrl+스페이스바를 누르면 전체이름을 쉽게 만들 수 있습니다.(이 경우에는 반입 목록에 이 클래스나 패키지를 추가하지 않는다고 가정합니다.) 목록에서 클래스 이름을 선택하면 이 이름이 자동으로 규정화됩니다. 그런 다음 마침표를 입력하고 Ctrl+스페이스바를 누르십시오. ResourceBundle에서 메소드 목록이 이제 나타납니다.

메소드 매개변수에 대한 코드 보조

코드 보조에는 메소드 매개변수용 팝업 도움말이 포함됩니다. 예를 들어 예제의 팝업 목록에서 **valueOf(String) Integer**를 선택하면 다음 텍스트가 커서에 삽입됩니다.

```
valueOf()
```

커서는 자동으로 괄호 사이에 오고 팝업 레이블 **문자열**이 표시되므로 추가한 매개변수 유형을 알 수 있습니다.



코드 보조 조정

다음과 같이 옵션 창에서 코드 보조를 조정할 수 있습니다. 이들 옵션에 대한 자세한 정보는 온라인 도움말을 참조하십시오.

- 트리거 코드 보조에 사용되는 키 순서를 지정하려면 **일반** > **키 바인딩**을 선택하고 코드 보조에 대한 입력항목을 편집하십시오.
- 트리거 키워드 완료에 사용되는 키를 지정하려면 **코딩** > **코드 보조**를 선택하십시오.
- 키워드 완료에 사용되는 템플리트를 지정하려면 **코딩** > **키워드 완료**를 선택하십시오. 또한 사용자 자신을 추가할 수 있습니다.
- 명명된 코드 보조 매크로를 정의하려면 **코딩** > **매크로**를 선택하십시오.

코드 보조 팁

- 코드 보조는 클래스 정의에 사용할 수 없습니다.
- 코드 보조는 대소문자를 구분합니다. 예를 들어 대문자 C로 시작하는 프로그램 구성요소를 찾으려면 Ctrl+스페이스바 앞에 대문자 C를 입력해야 합니다. 마찬가지로, 팝업 목록을 아래로 좁힐 때 적절한 대소문자를 입력하십시오.
- `for`, `while` 또는 `if`같은 Java 키워드로 시작하는 이름에 대한 코드 보조에 액세스하려면 키워드 **앞이나 뒤의** 최소한 한 글자에 Ctrl+스페이스바를 입력해야 합니다. 그렇지 않으면 정보 막대가 현재 문맥에서 코드 보조를 사용할 수 없음을 나타냅니다.
- 사용자 시스템에서 Ctrl+스페이스바를 눌러도 코드 보조가 시작되지 않을 경우, Ctrl+L 키를 사용하십시오.

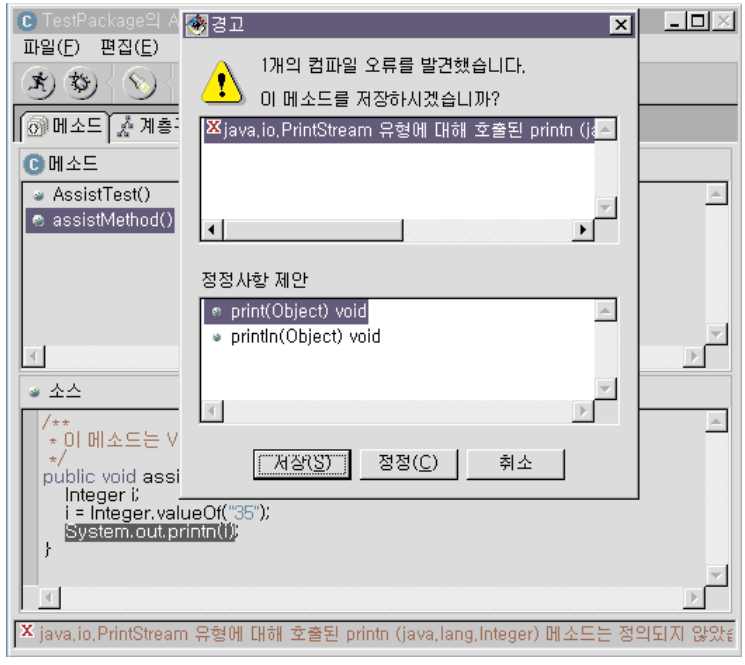
코드 해결

오류가 있는 코드를 저장하려고 하면, IDE가 코드에 오류가 있음을 경고합니다. 오류 유형을 판별할 수 있는 경우 가능한 해결방안 목록을 표시합니다. 하나를 선택하여 오류를 정정하거나 오류가 있는 코드를 저장할 수 있습니다.(프로그램 구성요소를 포함하는 IDE 검색기의 문제 페이지에 문제 목록이 추가됩니다.)

예를 들어 다음 행(실수 포함)을 위의 assistMethod 메소드에 추가합니다.

```
System.out.pritn(i);
```

메소드를 저장할 때 문제점을 해결하는 대체 코드를 추천하는 대화 상자가 나타납니다.



추천된 단어 **print(Object) void**를 선택한 다음 **정정**을 선택하십시오. 이 메소드는 대체 코드와 함께 저장됩니다. **저장**을 선택하면 메소드가 오류와 함께 저장됩니다. **취소**를 선택하면 메소드는 저장되지 않고 오류가 코드에 그대로 남게 됩니다.

코드 포맷팅

읽기 쉬운 코딩의 향상을 위해 IDE는 소스 분할영역에 코드 작성시 코드가 표시되는 방법을 자동으로 제어하는 자동 코드 포맷터를 제공합니다. 들여 쓰기 및 신규행 제어를 포함한 코드 포맷터 옵션을 설정하려면, 다음을 수행하십시오.

1. **창** 메뉴에서 **옵션**을 선택하여 옵션 대화 상자를 여십시오.
2. 옵션 대화 상자의 왼쪽 목록에서 **코딩** 항목을 확장하십시오.
3. **포맷터** 항목을 선택하십시오. 포맷터 페이지에서, 소스 분할영역에 복합 명령문의 각 명령문에 대한 신규 행 시작 또는 여는 중괄호 사용을 알리는 옵션을 사용가능하게 만들 수 있습니다.

4. **들여쓰기** 항목을 선택하십시오. 들여 쓰기 페이지에서 들여 쓰기 양식을 선택할 수 있습니다.

이러한 스펙은 새로운 모든 코드에 자동으로 적용됩니다. 파일 시스템에서 반입된 코드가 있거나 옵션을 변경한 경우에는 분할영역의 팝업 메뉴에서 **형식 코드**를 선택하여 특정 소스 분할영역의 코드에 옵션을 적용할 수 있습니다.

직접 간단한 빈 작성

비주얼 컴포지션 편집기에서 시각적으로 빈을 구성하는 것이 처음부터 빈을 일일이 작성하는 것보다 더 빨리 빈을 만들 수 있습니다. 그러나 직접 코드를 작성하여 빈을 만드는 옵션도 있습니다. 다음은 직접 간단한 애플릿 코드를 작성하는 방법입니다. 이 프로세스를 19 페이지의 『애플릿, 프로젝트 및 패키지 작성』에 있는 비주얼 컴포지션 예제와 비교해 보십시오.

Workbench의 **파일** 메뉴에서 **빠른 시작**을 선택하십시오. 빠른 시작 창에서 다음 단계를 수행하십시오.

1. **기본**을 선택한 다음 **클래스 작성**을 선택하십시오.
2. **OK**를 선택하십시오. 클래스 작성 SmartGuide가 열립니다.

클래스 작성 SmartGuide에서 다음 단계를 수행하여 애플릿을 작성하십시오.

1. **프로젝트** 필드에 프로젝트 이름을 입력하십시오.
2. **패키지** 필드에 패키지명을 입력하십시오.
3. **애플릿 이름** 필드에 이름을 입력하시오. 이 예제에서는 *HelloWorld*를 애플릿 이름으로 사용합니다.
4. **최상위 클래스** 필드에서 **JApplet**을 선택하십시오.
5. **완료시 클래스 검색**을 선택하십시오.
6. **완료**를 선택하십시오.

HelloWorld에 대한 클래스 검색기가 열리고 소스 분할영역에 다음이 표시됩니다.

```
/**
 * This type was created in VisualAge.
 */
public class HelloWorld extends com.sun.java.swing.JApplet {
}
```

다음과 같이 생성된 주석 위에 사용자 클래스 반입 명령문을 추가하십시오.

```
import com.sun.java.swing.*;
import java.awt.event.*;
import java.awt.*;
/**
 * This type was created in VisualAge.
 */
public class HelloWorld extends com.sun.java.swing.JApplet {
}
```

Ctrl+S 키를 눌러 새로운 코드를 저장하십시오.

메소드 분할영역에서 *init()* 메소드를 작성하여 애플릿을 완성하십시오.

1. 오른쪽 마우스 단추를 클릭하여 **추가**를 선택한 다음 **메소드**를 선택하십시오. 메소드 SmartGuide 작성이 나타납니다.
2. 강조 표시된 텍스트를 `void init()`로 변경하십시오.
3. **완료**를 선택하십시오. 클래스 검색기의 소스 분할영역에 *init()*에 대해 생성된 스텁이 표시됩니다.

새로운 메소드에 대한 코드를 작성하십시오. 이 예제에서는 스텁에 코드를 추가하여 다음과 같이 만드십시오.

```
public void init() {
   JButton button = new JButton("Hello World!");
   setBackground(Color.lightGray);
   getContentPane().add(button, "Center");
}
```

Ctrl+S 키를 눌러 새로운 코드를 저장하십시오.

지금 HelloWorld를 실행하면 애플릿 열람기가 애플릿을 초기화하지 못합니다. 마지막으로 정확한 클래스 경로를 설정합니다.

1. **클래스** 메뉴에서 **특성**을 선택하십시오. 특성 창이 열립니다.
2. 클래스 경로 페이지에서 **지금 계산**을 선택한 다음 **OK**를 선택하십시오.

이제 클래스를 실행하십시오. 간단한 애플릿이 작동합니다.

국제화 지원

VisualAge for Java는 두 가지 방법의 로케일에 의한 두 가지 텍스트 분리 방법 즉, 목록 묶음과 특성 파일을 지원합니다. 목록 묶음은 java.util.ListResourceBundle의 영속 양식입니다. 특성 파일은 java.util.PropertyResourceBundle의 영속 양식입니다.

두 가지 유형의 자원 묶음 모두 키 값 쌍을 포함합니다. 목록 자원 묶음의 경우 이들 쌍은 저장소의 번들 클래스에 저장되어 있습니다. ListResourceBundle.getContents()는 키 값 쌍 배열을 리턴합니다. 특성 자원 묶음의 키 값 쌍은 파일 시스템에 저장됩니다.

각 자원 묶음은 하나(또는 기본)의 로케일에 대한 값을 포함합니다. 묶음의 이름은 VM이 현재 로케일 설정에 따라 적절한 자원을 로드하도록 로케일별로 키 순서화될 수 있습니다.

VisualAge for Java는 클래스에서 발견되는 모든 텍스트에 대한 자원 묶음의 작성, 편집 및 사용을 지원합니다. 비주얼 컴포지션 편집기에서 설정할 때처럼 문자열 특성 값을 분리할 수 있으며, 또는 Workbench에서 한번에 모든 텍스트를 분리할 수도 있습니다.

사용자 소유의 자원 묶음을 사용하거나 VisualAge for Java를 사용하여 자원 묶음을 작성할 수 있습니다. 사용자가 직접 또는 VisualAge for Java에서 기존의 자원 묶음을 편집할 수 있습니다. 여러 자원 묶음을 단일 빈 내에서 참조할 수 있습니다. VisualAge for Java는 다음 번에 빈 저장시 적절한 코드를 생성합니다.

빠른 시작 창 사용

IDE에서 가장 자주 수행되는 몇몇 타스크는 빠른 시작이라는 액세스하기 쉬운 창에 수집됩니다. 빠른 시작 창을 열려면 IDE 창의 **파일** 메뉴에서 **빠른 시작**을 선택하거나 F2 키를 누르십시오.

왼쪽 목록에는 빠른 시작 타스크의 범주가 표시됩니다. 하나를 선택하여 오른쪽에 표시된 사용할 수 있는 타스크를 보십시오. 타스크를 시작하려면 타스크를 선택하고 **OK**를 클릭하십시오.

기본 타스크

프로그램 구성요소를 작성하는 기본 타스크는 사용자가 애플릿, 클래스, 인터페이스, 프로젝트 및 패키지를 작성하는 데 도움이 되는 적절한 SmartGuide를 시작합니다.

코드 시험 타스크는 스크랩북에서 하나의 페이지를 엽니다. 이 페이지는 스크랩북이 수행할 수 있는 작업을 배우기 위한 소개 지침을 제공합니다. 스크랩북에서 코드 및 변수 값을 평가하는 기본 단계를 배울 수 있습니다.

팀 개발 타스크

[엔터프라이즈] 팀 개발 타스크를 사용하면 저장소의 프로젝트에 쉽게 액세스할 수 있습니다. 또한 팀 관리자는 사용자를 관리할 수 있습니다. **관리 조회** 옵션으로 상태, 소유자 또는 개발자를 근거로 프로그램 구성요소를 탐색할 수 있습니다.

저장소 관리 타스크

저장소 압축 타스크를 통해 제거되고 열린 개정판을 그대로 두고 현재 저장소를 복제할 수 있습니다.

[엔터프라이즈] **저장소 변경** 옵션을 사용하여 작업영역을 공유 저장소나 국지 저장소 등에 연결하거나, 서버가 다운될 경우 복구할 수 있습니다.

기능 타스크

기능 추가 및 **기능 제거** 옵션을 사용하여 VisualAge for Java와 함께 제공되지만 저장소의 일부로 설치되어 있는 사용자 작업영역 프로젝트에 추가할 수 있습니다. 이러한 프로젝트에는 IDE 샘플과 데이터 액세스 빈이 포함됩니다. 추가하거나 제거할 프로젝트(*기능*)를 선택할 수 있는 보조 창이 나타납니다.

디버깅

VisualAge for Java는 풍부한 기능 세트와 함께 통합된 비주얼 디버거를 포함합니다. 이 절에서는 이러한 기능 중 몇가지를 살펴봅니다.

디버거 열기

창 메뉴에서 **디버그**를 선택한 다음 **디버거**를 선택하십시오. 프로그램이 수행 중이라면, 스레드를 일시 중단하고 스택과 변수 값을 볼 수 있습니다. 또는 디버거가 자동으로 열리고 다음과 같은 여러가지 이유로 현재 스레드가 일시 중단되기도 합니다.

- 코드에서 중단점이 발견되었습니다.
- true로 평가되는 조건 중단점이 발견되었습니다.
- 예외가 발생되었는데 포착하지 못했습니다.
- 예외 포착 대화 상자에서 선택된 예외가 발생되었습니다.

- 외부 클래스에서 중단점이 발견되었습니다.

중단점 설정

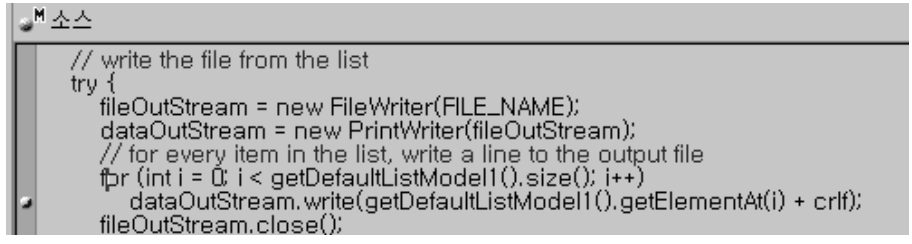
프로그램이 IDE에서 수행 중이고 중단점이 발견되면, 수행 중인 스레드는 일시 중단되고 메소드 스택에 대해 작업하고 변수 값을 검사할 수 있도록 디버거 검색기가 열립니다. IDE에서 소스를 표시하는 텍스트 분할영역에 중단점을 설정할 수 있습니다. To-Do List 프로그램에서 ToDoList 클래스의 *writeToDoFile()* 메소드에 중단점을 설정한다고 가정합니다.

중단점을 설정하려면, 다음을 실행하십시오.

1. Workbench에서 ToDoList 클래스를 선택하십시오. 클래스를 확장하여 그 메소드를 보십시오.
2. *writeToDoFile()* 메소드를 선택하십시오. 메소드의 소스가 소스 분할영역에 표시됩니다.
3. 다음 행(항목을 작성하는 루프에서) 옆에 있는 **소스** 분할 영역의 왼쪽 여백을 마우스 오른쪽 단추로 두 번 클릭하십시오.

   ```
   dataOutStream.write(getDefaultListModel1().getElementAt(i) + crlf);
   ```

4. 중단점 표시기가 이 행 옆의 소스 분할영역 여백에 나타납니다.



또한 다음 단계를 따라 중단점을 이미 포함한 행에서도 중단점을 설정할 수 있습니다.

1. 커서를 그 행으로 이동하십시오.
2. 마우스 오른쪽 단추를 클릭하여 팝업 메뉴에서 **중단점**을 선택하십시오.

중단점 제거

소스 분할영역에서 중단점을 제거하려면, 중단점 표시기를 두 번 클릭하십시오. 또한 다음 단계를 따라 중단점을 제거할 수도 있습니다.

1. 커서를 그 행으로 이동하십시오.
2. 마우스 오른쪽 단추를 클릭하여 팝업 메뉴에서 **중단점**을 선택하십시오.

방금 설정한 중단점 제거를 시도해보십시오. 이제 이를 재설정하십시오. 다음 절에서는 이 중단점을 사용하여 디버거 검색기의 기능을 점검합니다.

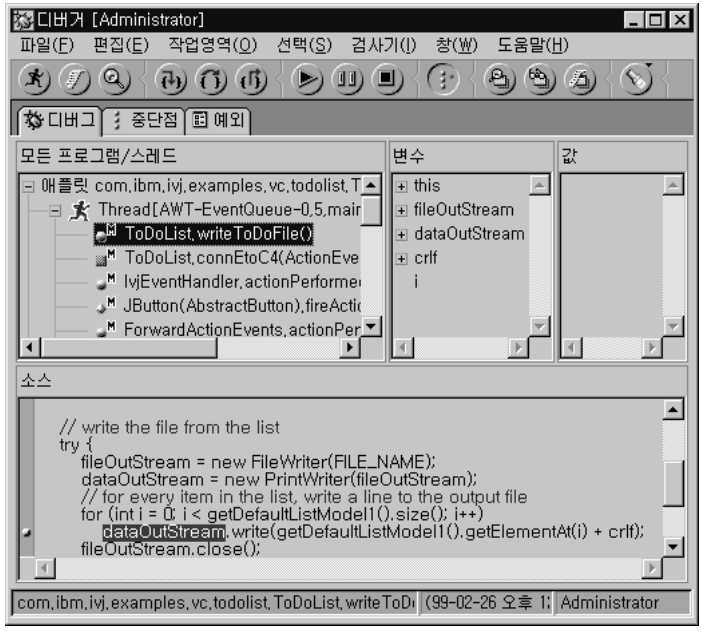
디버거 검색기 사용

사용자가 실행 중인 프로그램이 활성 중단점에 도달하거나 처리되지 않은 예외를 포함할 경우 디버거 검색기가 자동으로 열립니다.

중단점을 설정했다면, 이제 To-Do List 프로그램을 수행하여 발생한 것을 보기로 합니다.

1. Workbench에서 중단점이 사용가능한지 확인하십시오.
   - **창** 메뉴에서 **디버그**를 선택하고 **중단점**을 선택하십시오.
   - **전역 사용가능 중단점** 도구 막대 단추  가 사용가능 위치에 있는지 확인하십시오. 그렇지 않으면 클릭하십시오.
   - 디버거 창을 닫으십시오.
2. Workbench로 돌아가 ToDoList 클래스를 선택하십시오. **수행** 도구 막대 단추  를 선택하십시오.
3. To-Do File 프로그램이 나타나면 To-Do List에 최소한 세 개 항목을 추가하고 **Save To-Do File**을 선택하십시오. 디버거 검색기가 나타납니다. 검색기는

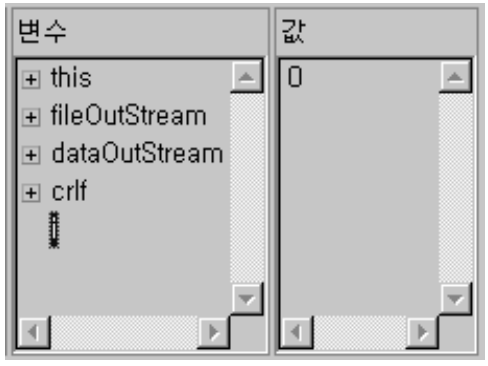
다음과 같이 나타나야 합니다.



디버그 중인 스레드가 모든 프로그램/스레드 목록에서 선택되었습니다. 스레드 아래의 메소드 목록은 현재 스택입니다. 스택에서 메소드를 하나 선택하면 가시적 변수 분할영역에 가시적 변수가 표시됩니다. 소스 분할영역에는 중단점이 설정된 소스가 나타납니다.

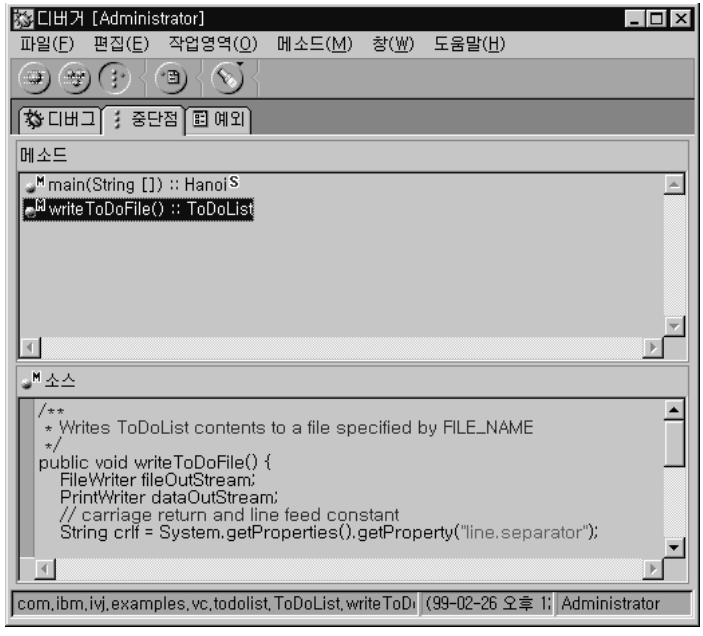
4. **재개** 도구 막대 단추  를 선택하여 프로그램을 계속 실행하십시오. 이 중단점이 각 항목을 파일에 쓰는 하나의 루프 내에 있기 때문에, 스레드가 다시 일시 중단되고 디버거 창이 이를 표시합니다.
5. 변수 목록에서 몇몇 변수를 점검하십시오. 예를 들어 루프 카운터 변수 i 값을 보려면 목록 아래에 있는 변수 분할영역에서 **i**를 선택하십시오. 값 분할영역에

값이 표시됩니다.



루프 카운터의 이 값은 루프가 한번 실행되었기 때문에 사용자가 예상한 값과 정확하게 일치합니다.

6. 이제 중단점을 사용불가능으로 설정합니다.
   - 디버거 검색기의 **중단점** 탭을 선택하십시오. 중단점 페이지가 나타납니다.



- 중단점 페이지는 사용자가 작업영역에 설정한 모든 중단점을 나열합니다. 메소드 분할영역은 사용자가 중단점을 설정한 모든 메소드를 나열합니다. 소스 분할영역은 메소드 분할영역에서 선택된 메소드에 대한 소스를 표시합니다.
- 중단점을 사용불가능으로 설정하려면 **전역 사용가능 중단점** 도구 막대 단추

  
  를 클릭하십시오. 중단점이 사용불가능 위치로 됩니다. 중단점 표시기는 색을 변경하여 이것이 사용불가능해졌음을 나타냅니다. 제거되지는 않지만 프로그램 재개시 무시됩니다.
- 디버그 페이지로 돌아가려면 **디버그** 탭을 선택하십시오.

7. 디버거 창의 **소스** 분할영역에서 코드를 갱신하고 저장할 수 있습니다. 프로그램의 실행 재개시 사용자가 코드에 가한 변경을 보십시오. 예를 들어 항목이 파일에 역순으로 쓰여 지도록 *writeToDoFile()* 메소드를 변경한다고 가정합니다. `for` 루프의 시작 부분을 다음과 같이 수정하여 메소드를 변경할 수 있습니다.

   ```
   for (int i = getDefaultListModel1().size(); i >= 0; i–)
   ```

   디버거 페이지의 **소스** 분할영역에서 이렇게 변경한 다음 **편집** 메뉴에서 **저장**을 선택하십시오.

8. 이제 도구 막대에서 **재개**를 선택하여 프로그램을 계속 실행하십시오. To-Do File 프로그램에서 To-Do List에 다음 값을 추가한 다음 **Save To-Do File**을 선택하여 파일에 저장하십시오.

   ```
   item A
   item B
   item C
   last item
   ```

9. 이제 **Open To-Do File**을 선택하고 방금 저장한 파일을 여십시오. **To-Do List**는 다음과 같습니다.

   ```
   last item
   item C
   item B
   item A
   ```

계속하기 전에 중단점 페이지로 돌아가 **전역 사용가능 중단점** 단추를 클릭하여 중단점을 다시 실행하십시오.

통합 디버거와 함께 할 수 있는 기타 기능

디버거는 프로그램을 디버그하는 데 도움이 되는 기타 많은 기능을 가지고 있습니다. 다음 타스크에 대해 자세히 알려면 통합 디버거 관련 온라인 도움말을 참조하십시오.

조건 중단점 설정

때때로 특정 조건하에서만 스레드를 일시 중단하는 중단점이 필요합니다. 디버거가 실행의 일시 중단을 결정하기 전에 표현식이 계산되도록 중단점을 구성할 수 있습니다. 표현식에 true로 계산되는 부울값이 있는 경우, 보통 중단점이 실행을 일시 중단합니다. false로 계산되는 경우 중단점은 무시됩니다.

중단점을 구성하려면, 소스 분할영역의 여백에서 그 기호를 마우스 오른쪽 단추로 클릭하십시오. 팝업 메뉴에서 **수정**을 선택하십시오. 필드에 표현식을 입력하십시오. 중단점 구성에 대해 자세히 알려면 통합 디버거 관련 온라인 도움말을 참조하십시오.

외부 및 포착된 예외 중단점 설정

작업영역의 코드에 중단점을 설정할 뿐만 아니라, 외부 클래스(파일 시스템에서 작업영역 외부에 상주하며 런타임시 로드되는 클래스)의 메소드에 중단점을 설정할 수도 있습니다. 예외가 발생될 경우 심지어는 사용자 코드가 이를 포착하고 처리하는 경우에도 실행을 중단시키는 예외 유형을 지정할 수 있습니다. 외부 중단점 및 포착된 예외의 중단점에 대해 자세히 알려면 통합 디버거 관련 온라인 도움말을 참조하십시오.

코드 Step through

수행 중인 스레드가 일시 중단되는 경우, 다양한 방법으로 행별로 또는 메소드별로 코드를 step through할 수 있습니다. 이것은 프로그램의 각 시점에서 변수 값을 점검하는 제어된 방법입니다.

검사기를 사용한 변수 열람 및 수정

검사기 창을 열어 일시 중단된 스레드에서 특정 변수를 볼 수 있습니다. 검사기를 통해 변수 값을 열람 및 수정하고 표현식을 계산할 수 있습니다.

JavaBeans 컴포넌트 지원

VisualAge for Java는 JavaBeans 컴포넌트 지원을 포함합니다. 이 절에서는 JavaBeans 컴포넌트에 대한 간략한 소개와 이에 대한 VisualAge for Java 지원 방식을 자세하게 설명합니다.

JavaBeans 컴포넌트란 무엇인가?

JavaBeans 컴포넌트는 JavaBeans 스펙에 따라 작동하는 Java 객체입니다. JavaBeans 컴포넌트 (또는 더 간단하게, *빈*)는 사용자가 VisualAge for Java와 같은 개발 환경에서 조작할 수 있는 재사용가능한 소프트웨어 컴포넌트입니다. 빈의 메소드 기호와 클래스 정의는 VisualAge for Java와 같은 환경에서 그 특성과 수행 판별을 허용하는 패턴을 따릅니다. 빈 인식 환경의 빈 특성 판별 기능을 *내성(introspection)*이라 합니다.

빈 기능

빈에는 다음 세 가지 종류의 기능이 있습니다.  이벤트,  메소드 그리고  특성.

비주얼 컴포지션 편집기에서 To-Do File 프로그램의 빈에 연결했다면, 이러한 세 가지 범주를 본 적이 있었을 것입니다. 빈은 다른 빈에서 같은 해당 기능을 사용할 수 있게 만들 때 기능을 *드러냅니다*.

다음은 세 가지 기능에 대한 간략한 설명입니다.

1. *이벤트*는 빈이 일으킬 이벤트입니다. 기타 빈은 이러한 이벤트에 자신의 관련성을 등록할 수 있으며, 이벤트 발생시 이를 통지받을 수 있습니다.

2. *메소드*는 다른 빈이 호출할 때 빈이 노출하는 조치입니다. 빈 메소드는 해당 빈을 구성하는 Java 클래스에 대한 public 메소드의 하위 세트입니다.
3. *특성*은 빈에 노출하는 속성입니다. 특성을 읽거나 쓰거나 또는 읽고 쓸 수 있습니다. 특성은 다음 특징을 가질 수 있습니다.
   - *바운드* 특성은 값이 변경될 때 *propertyChange* 이벤트를 시작합니다.
   - *제한* 특성은 특성값을 변경할 수 있는지 다른 빈이 판단할 수 있게 해 줍니다.(*vetoableChange* 이벤트를 시작합니다.)
   - *색인화* 특성은 하나의 배열로, 추가 메소드를 노출하여 개별 구성요소를 주소 지정합니다.
   - *숨김* 특성은 보이지 않게 숨기는 특성입니다. 빈 인식 도구에서만 사용됩니다.
   - *전문가* 특성은 전문 사용자만 조작할 수 있습니다.
   - *보통* 특성은 숨김도 아니고 전문가도 아닌 특성입니다.

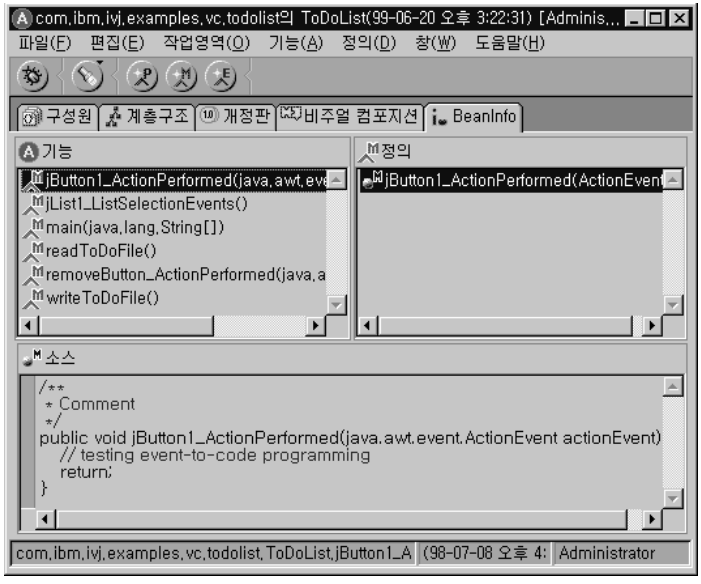
BeanInfo 클래스

빈은 동반 BeanInfo 클래스를 포함할 수 있습니다. 이러한 클래스는 빈이 노출한 이벤트, 메소드 및 특성을 명시적으로 기술합니다. VisualAge for Java는 사용자의 빈에 대해 BeanInfo 클래스를 작성할 수 있습니다. BeanInfo 클래스는 이름 뒤에 *-BeanInfo*가 접미어로 있는 빈과 같은 이름입니다.

BeanInfo 클래스에는 빈의 클래스, 빈 클래스의 이름, 그 빈의 이벤트, 메소드, 특성에 대한 상세한 내용을 포함하여 빈에 대한 정보를 리턴하는 public 메소드가 들어 있습니다.

BeanInfo 페이지

VisualAge for Java에서 클래스 검색기의 BeanInfo 페이지에서 빈 특성을 조작합니다.



위 왼쪽 창은 빈의 기능을 나열합니다. **기능** 메뉴의 **보기**에서 항목을 선택하여 BeanInfo 페이지가 표시하는 기능의 종류를 지정할 수 있습니다. 다음 기능 그룹을 사용할 수 있습니다.

모두	VisualAge for Java에 의해 작성되는 기능을 포함하여 빈의 모든 기능
보통	사용자가 빈용으로 명시적으로 정의한 기능
특성	특성들
이벤트	이벤트들
메소드	메소드들
숨김	기능 숨김
전문가	고급 기능

기능을 선택하면, VisualAge for Java는 선택된 기능의 종류에 따라 위 오른쪽 창에 정보를 나열합니다.

이벤트	인터페이스, 수신기 메소드, 수신기 추가 메소드, 수신기 제거 메소드
특성	유형, 읽기 메소드, 쓰기 메소드
메소드	기호

위 오른쪽 창은 선택된 기능과 관련된 프로그램 구성요소를 나열합니다. 프로그램 구성요소 중 하나를 선택하면, 그 소스가 아래쪽 창에 표시됩니다.

위 오른쪽 창에서 프로그램 구성요소를 선택하지 않으면, 아래쪽 창에는 설명, 표시명, 고급인지 숨김인지 여부를 포함하여 선택된 기능에 대한 빈 정보를 나열합니다.

BeanInfo 페이지 사용

BeanInfo 페이지를 사용해서 빈의 기능을 작성 및 조작하는 방법은? com.ibm.ivj.examples.vc.customerinfo 패키지는 특성 사용 기능을 만드는 예제입니다. 예를 들어 Address 클래스는 시/군/구, 도, 우편번호에 대한 특성을 가집니다. IBM Java Examples 프로젝트에서 이 샘플을 찾을 수 있습니다.

이 샘플을 빌드하는 방법에 대한 설명을 보려면 제품 설명서로 이동하십시오. 검색 막대에서 **샘플**을 선택하십시오. 네비게이션 분할영역에서 **비주얼 컴포지션**을 선택한 다음 **CustomerInfo**를 선택하십시오.

작업영역 조정

VisualAge for Java는 자신의 요구와 구미에 맞게 IDE를 조정하기 위해 변경할 수 있는 특성 범위를 제공합니다. 이 절에서는 조정 옵션 설정 방법을 보여 주고 조정할 수 있는 항목에 대한 간단한 개요를 제공합니다.

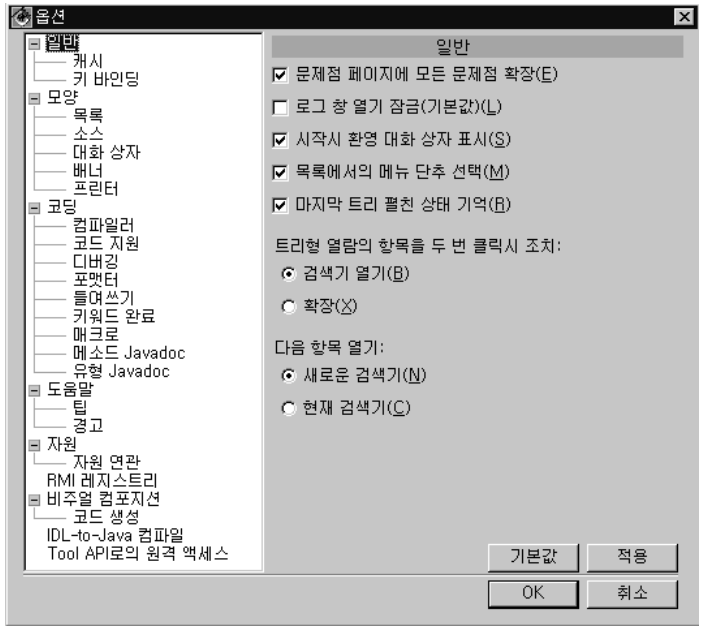
조정 옵션 설정

VisualAge for Java의 옵션 창에서 옵션을 설정하여 작업영역을 조정합니다. 프로그램 구성요소를 두 번 클릭할 때 무엇이 나타나는지 판별하는 옵션을 설정하여 이 창이 작동하는 방식을 알아보십시오.

기본값으로 프로그램 구성요소 아이콘을 두 번 클릭하면 자체 검색기에서 프로그램 구성요소가 열립니다. 예를 들어 패키지 아이콘을 두 번 클릭하면, 패키지에 들어있는 모든 유형을 표시하는 패키지 검색기가 열립니다. 프로그램 구성요소 아이콘을 두 번 클릭할 때 해당 아이콘 아래에 프로그램 구성요소 트리가 확장되거나 접히도록 이 동작을 변경할 수 있습니다.

프로그램 구성요소를 두 번 클릭할 때 프로그램 구성요소 아래에 트리 열람이 확장되도록 지정하려면 다음을 수행하십시오.

1. **창** 메뉴에서 **옵션**을 선택하십시오. **옵션** 창이 나타납니다.



2. **일반** 페이지를 선택하십시오.
3. **트리형 열람의 항목을 두 번 클릭시 조치** 아래에 있는 **확장** 라디오 단추를 선택하십시오.
4. **OK**를 클릭하십시오.

이제 프로그램 구성요소 아이콘을 두 번 클릭하면, 클릭한 구성요소가 포함하는 프로그램 구성요소를 표시하도록 트리 열람이 확장됩니다.

페이지의 옵션을 모두 기본값으로 설정하려면 **기본값**을 선택하고 **OK**를 선택하십시오.

옵션 창 왼쪽에 있는 트리 열람을 확장하여 사용가능한 페이지를 모두 표시할 수 있습니다. 트리에서 각 상위 항목(예를 들면 **코딩**)에도 하나의 페이지가 있습니다.

다음은 옵션 창에서 사용할 수 있는 페이지입니다.

- **일반** 페이지는 몇 가지 기타 IDE 동작에 대한 옵션을 표시합니다.
- **캐시** 페이지는 메모리 관리 옵션을 표시합니다.
- **키 바인딩** 페이지는 제품을 통해 사용가능한 단축 키 순서에 대한 현재 설정을 표시합니다. 명명된 프로파일에 이 설정을 편집하고 저장할 수 있습니다.
- **모양** 페이지는 목록, 소스 코드, 배너 및 색상, 크기, 글꼴을 포함한 프린터 출력 표시 방법을 표시합니다.
- **코딩** 페이지는 코드 탭 조작 및 저장 옵션을 표시합니다.
- **컴파일러** 페이지는 컴파일러 오류를 보고하기 위한 옵션을 표시합니다.
- **코드 보조** 페이지는 사용가능한 코드 보조에 대한 옵션을 표시합니다.
- **디버깅** 페이지는 통합 디버거와 스택 추적을 생성하는 옵션을 표시합니다.
- **포맷터** 페이지는 소스 분할영역에 코드를 표시하는 방법을 표준화하는 자동 코드 포맷터 옵션을 표시합니다.
- **들여쓰기** 페이지는 각 코드 행의 들여쓰기 방법에 대한 옵션을 선택합니다.
- **키워드 완료** 페이지는 소스 코드에서 Java 구문의 자동적 완료에 대한 현재 설정을 표시합니다.
- **매크로** 페이지는 IDE에 대해 현재 정의된 코드 보조 매크로를 표시합니다.
- **Javadoc 메소드** 및 **Javadoc 유형** 페이지는 각 메소드와 유형에 대한 표준 Javadoc 주석을 생성하는 옵션을 표시합니다.
- **도움말** 페이지는 도움말 정보를 표시하는 데 사용할 웹 브라우저를 지정하는 옵션을 표시합니다.
- **팁** 및 **경고** 페이지는 사용자가 참조할 팁이나 경고를 지정하는 옵션을 표시합니다.
- **자원** 페이지는 작업영역에서 실행중인 프로그램에 필요한 자원 파일과 클래스를 찾을 수 있도록 클래스 경로를 설정하는 곳입니다.

- **자원 연관** 페이지는 외부 프로그램 또는 Java 클래스와 함께 자원 유형을 연관하게 할 수 있습니다.
- **RMI 레지스트리** 페이지는 RMI 옵션을 표시합니다.
- **비주얼 컴포지션** 페이지는 BeanInfo 지원과 비주얼 컴포지션 편집기에 대한 옵션을 표시합니다.
- **코드 일반** 페이지는 시각적으로 합성된 빈에 대한 이벤트 핸들링 코드 및 데타 데이터를 생성하는 옵션을 표시합니다

제10장 관계형 데이터 작업 소개

VisualAge for Java는 JDBC를 통해 관계형 데이터베이스에 대한 액세스를 지원합니다. 비주얼 컴포지션 편집기 빈 팔레트에서 데이터 액세스 빈을 사용하여 애플릿 또는 응용프로그램에 있는 관계형 데이터에 액세스할 수 있습니다. 데이터 액세스 빈은 다음을 포함합니다.

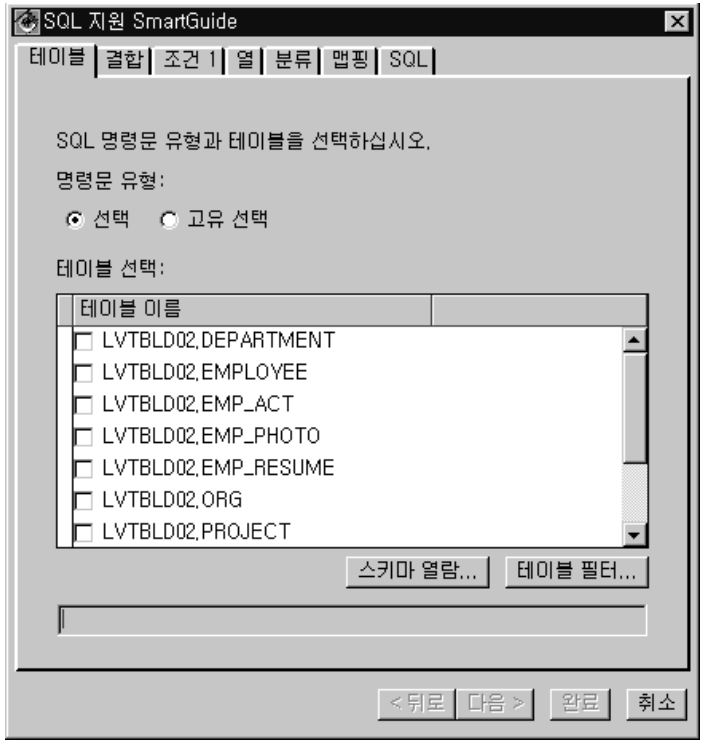
- Select 빈은 조회를 수행하고 조회에 의해 리턴된 결과 세트를 액세스할 수 있습니다. SQL INSERT, UPDATE 및 DELETE 명령문을 분리하여 작성할 필요없이 결과 세트에서 행을 삽입, 갱신 및 삭제할 수 있습니다.
- Modify 빈은 조회를 첫번째 수행과 그 결과 세트의 검색없이 SQL INSERT, UPDATE 또는 DELETE 명령문을 수행할 수 있습니다.
- ProcedureCall 빈은 입력 매개변수에 대한 값을 경과하고 출력 매개변수에 대한 값을 검색하는 저장된 프로시저를 수행할 수 있습니다. 저장된 프로시저가 하나 이상의 결과 세트를 리턴하면 ProcedureCall은 사용자가 결과 세트를 액세스하게 하고 사용자는 SQL INSERT, UPDATE 및 DELETE 명령문을 분리하여 작성할 필요없이 결과 세트 내의 열을 삽입, 갱신 및 삭제할 수 있습니다.
- Selector 빈은 리턴된 데이터의 다른 열람을 제공합니다. 다음과 같습니다.
  - CellSelector, 단일 값 보기 제공
  - RowSelector, 행 보기 제공
  - ColumnSelector, 열 보기 제공
  - CellRangeSelector, 2차원 보기 제공
- DBNavigator 빈은 Select 빈 또는 ProcedureCall 빈과 함께 사용할 수 있는 비주얼 빈입니다. DBNavigator 빈은 결과 세트에서 행을 네비게이트하는 관련된 넌-비주얼 빈에 대한 SQL 명령문을 수행하거나 데이터베이스에 갱신 확인과 같은 다른 관계형 데이터베이스 작동을 실행하는 단추의 세트를 제공합니다.

데이터 액세스 빈을 사용하려면 먼저 빠른 시작 창을 사용하여 데이터 액세스 빈 기능을 VisualAge for Java에추가하고 클래스 경로를 변경해야 합니다. 관계형 데이터 액세스에 대한 자세한 정보는 온라인 도움말을 참조하십시오.

Select 빈을 사용하여 데이터 액세스

Select 빈을 사용하여 관계형 데이터에 액세스하려면 다음을 수행하십시오.

1. Select 빈을 비주얼 컴포지션 편집기에 추가하십시오.
2. Select 빈 특성을 편집하십시오. 조회 특성으로 다음을 정의할 수 있습니다.
   - **연결 별명**. 이것은 Select 빈에 대한 데이터베이스 연결의 특성을 식별합니다. 여기에는 연결용 URL, 연결 요청과 함께 전달될 사용자 ID 및 암호 같은 특성이 포함됩니다.
   - **SQL 스펙**. 이것은 Select 빈에 대한 SQL문을 지정합니다. 수동으로 SQL 문을 입력하기 위해 제공되는 SQL 편집기를 사용하거나 SQL 지원 SmartGuide를 사용하여 SQL문을 시각적으로 구성할 수 있습니다. 하나 이상의 테이블을 선택하고 테이블을 결합하고 탐색 조건을 지정하고 표시할 열을 식별하고 결과를 분류하고 결과 열의 데이터 유형을 Java 클래스로 맵핑하며 결과 SQL문을 볼 수 있습니다. 조회를 테스트 실행할 수도 있습니다.



3. Select 빈을 단추 빈과 같은 사용자 애플릿 또는 응용프로그램의 비주얼 컴포넌트에 연결하십시오. 사용자가 응용프로그램 또는 애플릿을 사용하고 비주얼 컴포넌트를 선택할 경우, 예를 들어 단추를 누를 때, Select 빈의 SQL문이 결과 세트를 검색합니다.

또한 Select 빈을 사용하여 사용자가 결과 행에 작성한 변경사항, 심지어는 행의 삭제까지 적용하고 데이터베이스에 대해 변경사항을 확정할 수도 있습니다. 이 경우, Select 빈에서 제공하는 갱신 및 삭제 메소드를 사용합니다.

DBNavigator 빈을 사용하여 결과 세트 탐색

DBNavigator 빈을 사용하면 프로그램 구성요소에 경로가 세트 행을 네비게이트하고 다양한 관계형 데이터베이스 작동을 수행하는 단추 세트를 쉽게 추가할 수 있습니다.

예를 들어 단추 중 하나는 결과 행의 다음 행을 현재 행으로 만들고 데이터베이스에 대한 변경사항을 확정합니다.

DBNavigator 빈은 Select 빈과 같이 사용됩니다. 148 페이지의『Select 빈을 사용하여 데이터 액세스』Select 빈은 관계형 데이터베이스에서 결과 세트를 검색할 때 사용됩니다. DBNavigator 빈은 결과 세트에 대해 동작합니다.

DBNavigator 빈을 사용하려면 다음을 수행하십시오.

1. DBNavigator 빈을 비주얼 컴포지션 편집기에 추가하십시오.
2. DBNavigator 빈 특성을 편집하십시오. 편집하는 특성 중에는 표시될 단추를 지정하는 특성도 있습니다.
3. DBNavigator 빈을 Select 빈에 연결하십시오.

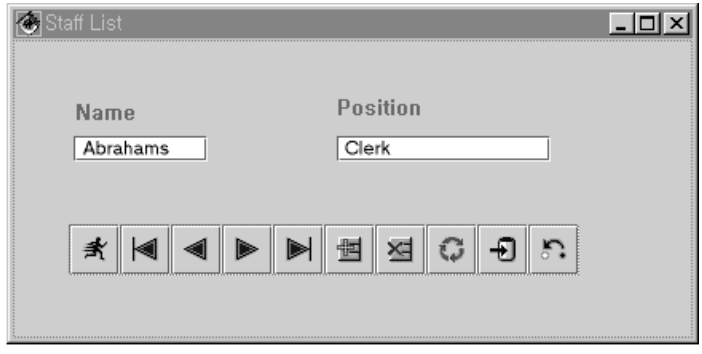
실제에서의 데이터 액세스빈 개요

간단한 다음 예제를 통해 DB2 SAMPLE 데이터베이스로부터 직원 데이터를 표시하는 간단한 응용프로그램을 빌드합니다. 진행하기 전에 워크스테이션에 DB2가 설치되어 있는지 확인하십시오.

이 데이터 액세스 응용프로그램를 빌드하는 과정은 다음과 같습니다.

1. 필요하면 149 페이지의『DB2 데이터베이스에 대한 액세스 설정』
2. 비주얼 컴포지트 StaffList에서 152 페이지의『StaffList 사용자 인터페이스 빌드』
3. 153 페이지의『Select 빈 추가 및 설정』
4. 응용프로그램 UI에 159 페이지의『Select 빈 연결』
5. 161 페이지의『StaffList 응용프로그램 테스트』

StaffList 응용프로그램은 DB2와 같이 제공된 SAMPLE 데이터베이스에서 직원 목록을 검색하여 목록을 탐색할 수 있게 해줍니다. 완료된 직원 응용프로그램은 다음과 같아야 합니다.



DB2 데이터베이스에 대한 액세스 설정

응용프로그램을 빌드하기 전에 먼저 VisualAge for Java에서 DB2의 SAMPLE 데이터베이스에 액세스할 수 있는지 확인하십시오. 이 예제에서는 Windows NT용 DB2 Universal Database, 버전 5.2를 사용했습니다.

SAMPLE 데이터베이스 점검

1. 바탕 화면에서 **시작**을 선택한 다음 **Windows NT용 DB2**를 선택하십시오.
2. **관리 도구**를 선택한 다음 **제어 센터**를 선택하십시오. 제어 센터 창이 나타납니다.
3. 데이터베이스 폴더가 나타날 때까지 시스템 트리의 항목을 확장하십시오. 데이터베이스 폴더를 확장하여 SAMPLE 데이터베이스가 있는지 확인하십시오.

워크스테이션에 SAMPLE 데이터베이스가 없으면 설명서를 참고하여 워크스테이션에 설치된 DB2 레벨을 찾으십시오.

VisualAge for Java에서 DB2에 대한 액세스 설정

1. Workbench의 메뉴 막대에서 **창**을 선택한 다음 **옵션**을 선택하십시오.
2. 왼쪽 분할영역에서 **자원**을 선택하십시오.
3. **작업영역 클래스 경로** 필드 왼쪽에서 **편집**을 선택하십시오. 대화 상자가 나타납니다.
4. 대화 상자에서 **Jar/Zip 추가**를 선택하십시오. 표준 파일 선택 창이 나타납니다.
5. 파일 선택 창에서 DB2를 설치할 때 작성된 sqllib 디렉토리를 찾으십시오.
6. java 폴더를 두 번 클릭하고 **db2java.zip**을 선택하십시오.
7. **열기**를 선택하십시오.
8. **OK**를 선택하여 대화 상자를 닫으십시오.
9. 클래스 경로에 디렉토리를 추가하려면 **적용**을 선택하십시오.
10. 옵션 창을 닫으려면 **OK**를 선택하십시오.

StaffList 사용자 인터페이스 빌드

데이터베이스 액세스가 제대로 설정되어 있으면 StaffList 응용프로그램에 대한 사용자 인터페이스를 빌드할 준비가 된 것입니다. 여기서 표준 작동을 위한 완전한 키보드 순서는 표시되지 않습니다. 이 표준 작동 사용법을 다시 보려면 15 페이지의 『제4장 첫번째 애플릿 빌드 소개』로 시작하는 ToDoList 예제를 다시 읽으십시오.

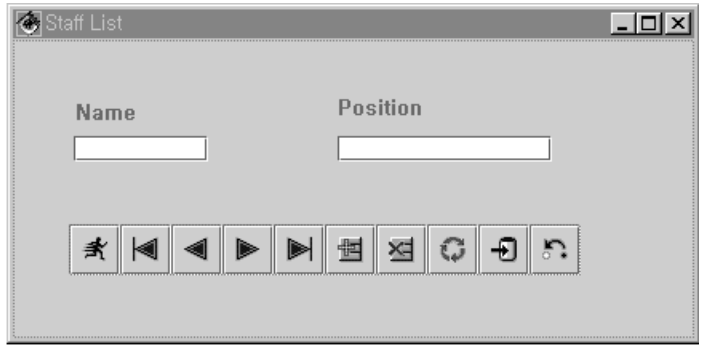
컴포지트 설정

1. 새로운 프로젝트 *Data Access Example*을 작성하십시오.
2. 새로운 패키지 *DAStaffList*를 작성하십시오.
3. 다음과 같이 새로운 클래스를 작성하십시오.
   - **클래스명** 필드에 StaffList를 입력하십시오.
   - **최상위 클래스** 필드에 com.sun.java.swing.JFrame을 입력하십시오.
   - **완료시 클래스 검색**과 **클래스를 시각적으로 구성**을 클릭하십시오.

비주얼 컴포지션 편집기에서 빈 드롭 및 설정

1. *title* 특성을 편집하여 JFrame에 제목을 추가하십시오. 이 예제에서는 제목을 StaffList로 변경하십시오.
2. 두 개의 JLabel 빈을 드롭하십시오. 다음과 같이 특성을 편집하십시오.
   - 첫번째 빈의 *text* 특성을 Name으로 변경하십시오.
   - 두번째 빈의 *text* 특성을 Position으로 변경하십시오.
   - 필요하면 레이블 *크기*를 조정하여 전체 레이블 텍스트를 표시하십시오.
3. 두 개의 JTextField 빈을 드롭하십시오.이름을 각각 NameTF와 PositionTF로 변경하십시오.
4. 빈 팔레트의 데이터베이스 범주에서 DBNavigator 빈  을 선택하고 JFrame 빈에 드롭하십시오.

5. 컴포넌트를 정렬하고 이동하여 아래 그림처럼 만드십시오.



6. 수행한 작업을 저장하십시오.

Select 빈 추가 및 설정

StaffList 사용자 인터페이스의 배치가 완료되었습니다. Select 빈을 추가하고 데이터베이스 조회를 설정하십시오.

Select 빈 추가

1. 빈 팔레트의 데이터베이스 범주에서 Select 빈  을 선택하고 StaffList 컴포지트의 빈 작업 영역에 드롭하십시오. 새로 드롭된 빈 이름이 자동으로 Select1으로 지정됩니다.

조회에 대한 데이터베이스 연결 설정

1. Select1의 특성 창을 엽니다.
2. *조회* 특성에 대한 값 필드를 클릭하십시오. 작은 단추가 나타납니다.
3. 작은 단추를 선택하십시오. Query 창이 나타납니다.
4. 연결 페이지에서 **데이터베이스 액세스 클래스** 필드 오른쪽에 있는 **신규**를 클릭하십시오. 프롬프터 창이 나타납니다.
5. VisualAge for Java에서는 이 때 지정한 별도의 클래스로 데이터 액세스 코드를 생성합니다. 다음과 같이 새로운 데이터 액세스 클래스에 대한 정보를 입력하십시오.

- **패키지** 필드에 DAStaffList를 입력하십시오.
- **클래스명** 필드에 Staffers를 입력하십시오.
- **OK**를 클릭하여 프롬프터 창을 닫으십시오.

데이터베이스 액세스 클래스 필드에는 이제 DAStaffList.Staffers 값이 있어야 합니다.

6. **연결** 필드 오른쪽에 있는 **Add** 단추를 선택하십시오. 연결 별명 정의 창이 나타납니다.
7. **연결 이름** 필드에 staffnames를 입력하십시오.
8. 연결이 제대로 되었는지 확인하려면 **연결 테스트**를 선택하십시오. 모두 제대로 설정되면 메시지 창에 연결이 성공적입니다.라는 메시지가 표시됩니다.
9. **OK**를 클릭하여 창을 닫으십시오.

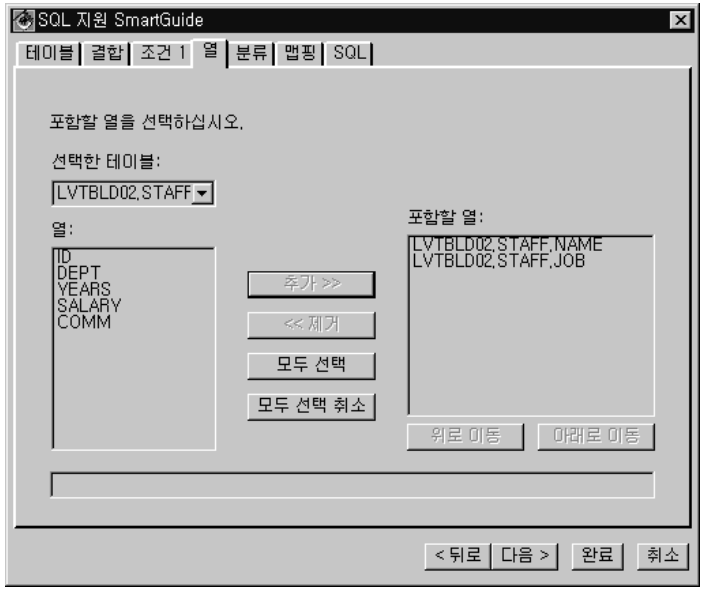
VisualAge for Java는 사용자가 지정한 데이터 액세스 클래스로 연결하기 위한 코드를 생성합니다. Query 창은 다음과 같이 나타나야 합니다.

SQL 문 어셈블리

1. Query 창에서 SQL 페이지를 선택하십시오. 첫번째 필드 **데이터베이스 액세스 클래스**에는 이미 값이 입력되어 있습니다.
2. **SQL** 필드 오른쪽에 있는 **추가**를 선택하십시오. 새로운 SQL 스펙 창이 나타납니다.
3. 다음과 같이 정보를 입력하십시오.
   - **SQL 이름** 필드에 `NamesTitles`를 입력하십시오.
   - **SQL 지원 SmartGuide 사용(권장)**을 선택했는지 확인하십시오.
   - **OK**를 클릭하여 창을 닫으십시오.

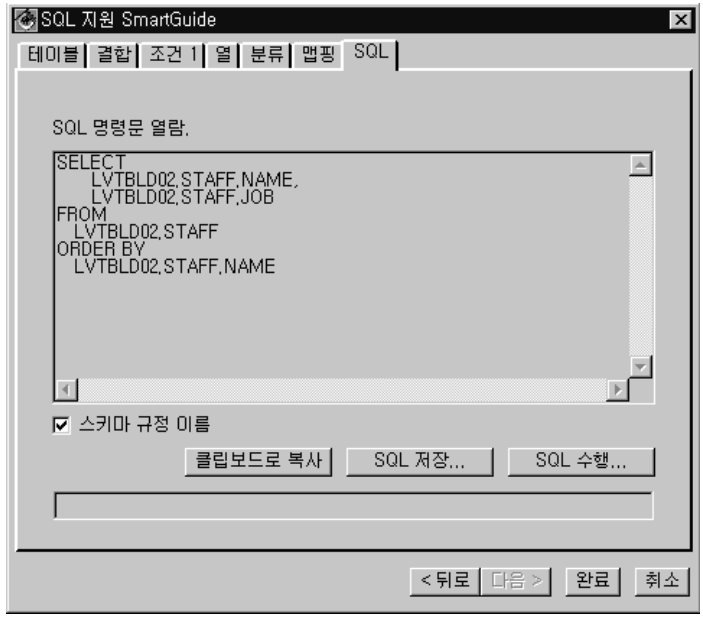
     VisualAge for Java는 SQL 지원 SmartGuide를 시작하고 SAMPLE 데이터베이스로 연결을 설정합니다.
4. SmartGuide의 테이블 페이지에서 *xxx*.STAFF 테이블을 선택하십시오. 여기서 *xxx*는 DB2를 설치할 때 설정된 기본 사용자 ID 입니다.
5. 열 페이지에서 조회에 포함할 열을 선택하십시오.
   - 열 목록에서 **NAME**을 선택한 다음 **추가**를 선택하십시오.
   - **JOB**을 선택한 다음 **추가**를 선택하십시오.

열 페이지는 다음과 같이 나타나야 합니다.



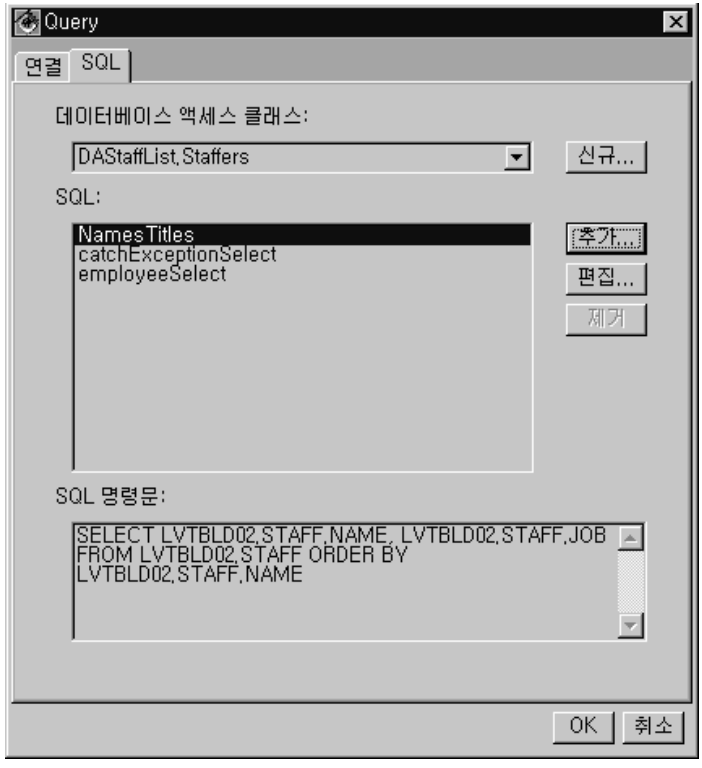
6. 분류 페이지에서 기본 분류 키를 지정하십시오. 열 목록에서 **NAME**을 선택한 다음 **추가**를 선택하십시오.

7. 조회를 보려면 해당 SQL 페이지를 선택하십시오. SQL 페이지는 다음과 같이 나타나야 합니다.

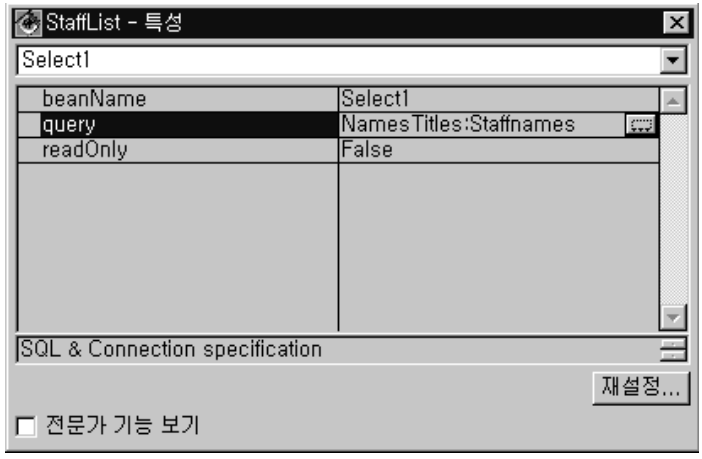


8. 종료를 선택하십시오.

SmartGuide 창이 닫히고 SQL 조회 코드가 생성된 다음 Query 창으로 돌아갑니다. Query 창의 SQL 페이지는 다음과 같이 나타나야 합니다.



OK를 클릭하여 Query 창을 닫은 다음 Select 빈의 특성 창으로 돌아가십시오.

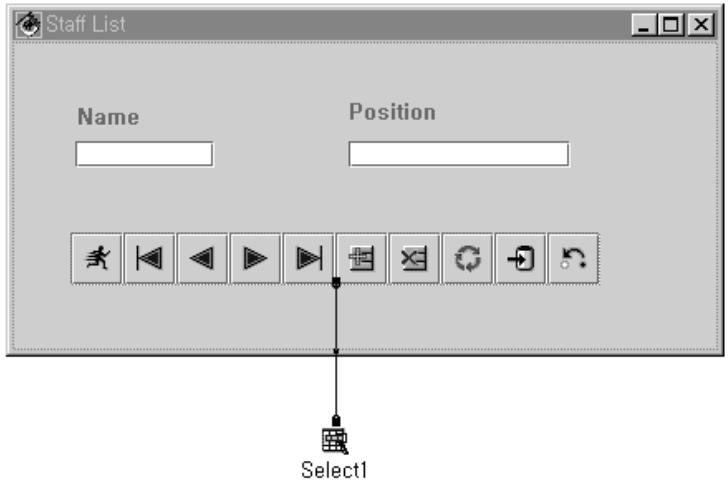


특성 창을 닫고 작업을 저장하십시오.

Select 빈 연결

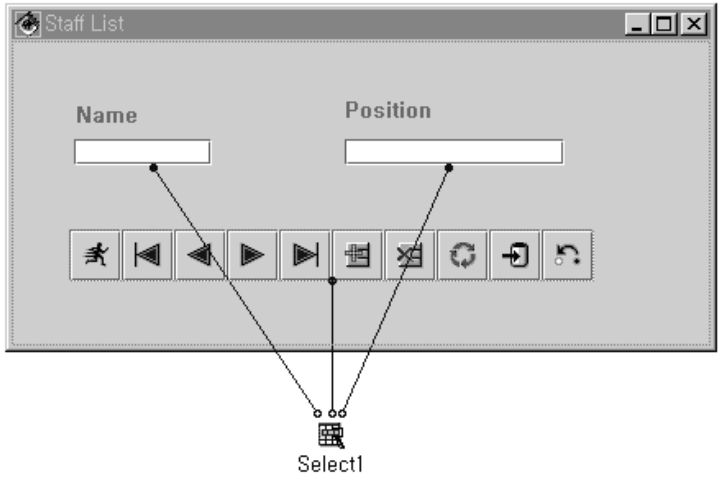
데이터베이스 조회가 설정되었습니다. 마지막 단계는 StaffList 컴포지트의 비주얼 및 넌-비주얼 컴포넌트를 연결하는 것입니다.

1. Select1의 *this* 특성을 DBNavigator 빈의 *model* 특성에 연결하십시오.



2. Select1의 *STAFF.NAME_String* 특성을 NameTF의 *text* 특성에 연결합니다. 이름이 *STAFF.NAME_String*인 기능이 두 개 있습니다. 특성을 선택하십시오.
3. Select1의 *STAFF.JOB_String* 특성을 PositionTF의 *text* 특성에 연결합니다. 이름이 *STAFF.JOB_String*인 기능이 두 개 있습니다. 특성을 선택하십시오.

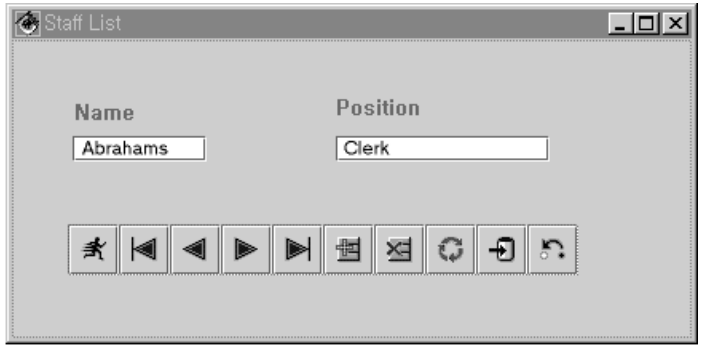
완료된 연결은 다음과 같이 나타나야 합니다.



이제 응용프로그램을 테스트할 준비가 되었습니다. 작업을 저장하십시오.

StaffList 응용프로그램 테스트

ToDoList에서 한 것처럼 완성된 StaffList 응용프로그램을 테스트하십시오. 도구 막대에서 **수행**을 선택하십시오.



처음에는 직원 데이터가 표시되지 않은 상태로 Staff List 창이 열립니다. 조회는 아직 수행되지 않았습니다. DB2에 조회를 제출하려면  를 선택하십시오.

도구 막대에서 일부 단추에 마우스를 놓으면 팝업 도움말이 나타납니다.

직원 목록을 탐색하려면  또는  를 선택하십시오.

축하합니다! 데이터 액세스 응용프로그램을 빌드했습니다.

제11장 Domino AgentRunner 사용

Domino(TM) AgentRunner는 VisualAge for Java에서 Domino 에이전트를 빌드, 수행, 디버그할 때 사용할 수 있습니다. AgentRunner는 Notes(TM) 문맥 정보를 액세스하는 디버거 클래스 세트를 사용하여 IDE에서 에이전트를 실행할 수 있습니다.

AgentRunner를 사용하려면 다음 단계를 수행하십시오.

1. 『Domino에서 AgentRunner 설정』
2. 164 페이지의 『Workbench에 Domino Java 클래스 추가』
3. 165 페이지의 『VisualAge for Java에서 에이전트 반입 또는 작성』
4. AgentRunner.nsf 파일에서 166 페이지의 『AgentContext 문서 생성』 AgentContext 문서는 IDE와 Domino 간에 전환할 필요없이 빌드, 디버그 및 수행할 수 있습니다.
5. VisualAge for Java 디버거를 사용하여 IDE에서 167 페이지의 『에이전트 디버깅』하십시오.
6. 168 페이지의 『생산 에이전트 작성』하십시오.

Domino에서 AgentRunner 설정

AgentRunner를 지원하도록 Domino 5.0 환경을 설정하려면 PATH 환경 변수에 Domino 디렉토리를 추가하십시오.

AgentRunner를 지원하도록 Domino 4.6 환경을 설정하려면 다음 단계를 수행하십시오.

1. IVJAgentRunner.jar 파일을 사용자 notes.ini 파일의 JavaUserClasses 문에 추가하십시오. JavaUserClasses 문이 사용자 notes.ini 파일에 없다면, 다음 명령문을 notes.ini 파일의 끝에 잘라 붙일 수 있습니다.

   ```
   JavaUserClasses=X:\IBMVJava\eab\runtime30\domino\ar\IVJAgentRunner.jar
   ```

여기서 X:\IBMVJava는 VisualAge for Java가 설치된 경로입니다. notes.ini 파일을 편집한 후, 변경사항을 적용시키려면 Domino를 종료했다가 재시작해야 합니다.

2. PATH 환경 변수를 Domino 디렉토리를 가리키도록 설정하십시오.
3. X:\IBMVJava\eab\runtime30\domino\ar 디렉토리에서 AgentRunner.nsf 파일을 사용자 Domino 데이터 디렉토리로 복사하십시오.

Domino 환경은 이제 AgentRunner를 지원하도록 설정되었습니다.

Workbench에 Domino Java 클래스 추가

AgentRunner를 사용할 수 있도록 VisualAge for Java IDE를 설정하려면 저장소에서 작업영역으로 Domino Java 클래스 라이브러리를 추가하십시오.

- Domino 5.0이 설치되었으면 Lotus Domino Java 라이브러리 5.0을 사용하십시오.
- Domino 4.6.*x*가 설치되었으면 Lotus Domino Java 라이브러리 4.6.1.을 사용하십시오.

1. Workbench에서 **파일**을 선택하고 **빠른 시작**을 선택하십시오.
2. 왼쪽의 범주 목록에서 **기능**을 선택하십시오.
3. 오른쪽의 타스크 목록에서 **기능 추가**를 선택하십시오. **OK**를 클릭하십시오.
4. 요구된 선택 창에서 설치한 Domino의 레벨에 맞는 Java 라이브러리 버전을 선택하십시오. **OK**를 클릭하십시오.

이제 Domino Java 클래스 라이브러리가 Workbench 창의 모든 프로젝트 보기에 표시되어야 합니다. 이 프로젝트는 AgentRunner를 지원하는 추가 디버그 클래스를 더한 Notes Object Interface/Domino의 적당한 버전에 대해 Java 클래스의 패키지(lotus.notes라 호출하는)를 포함합니다.

이제 IDE에서 에이전트 수행 또는 디버그시 이러한 클래스를 사용할 수 있습니다.

VisualAge for Java에서 에이전트 반입 또는 작성

Workbench에서 Domino 에이전트라는 프로젝트를 작성하십시오.

Domino에서 에이전트 반입

1. 다음은 반입 SmartGuide를 사용하여 사용자 에이전트(.java 파일)를 Domino에서 이 새로운 프로젝트로 반입하십시오.

   반입된 Java 코드는 컴파일되고 미해결 문제가 모든 문제점 페이지에 추가됩니다. 사용자 .java 파일이 Workbench의 사용자 Domino 에이전트 프로젝트에 있는 패키지에 나타납니다.

2. 새롭게 반입된 Java 소스 코드에서 클래스 선언을

   ```
   public class ... extends AgentBase
   ```

   에서

   ```
   public class ... extends DebugAgentBase
   ```

   로 변경하십시오.

3. 사용자 코드를 저장하십시오.

새로운 에이전트 작성

1. Workbench에서 최상위 클래스로 *DebugAgentBase*를 사용하여 Domino 에이전트 프로젝트에서 클래스를 작성하십시오.
2. 클래스 이름에서 마우스 왼쪽 단추를 클릭하십시오. Workbench의 소스 분할 영역에서 사용자 에이전트에 대한 코드를 작성하십시오.

   Domino 5.0에 대해서는 *Domino 5 Designer Help*를 참조하십시오. Domino 4.6에 대해서는 Lotus Notes의 *Java Programmer's Guide*를 참조하십시오.
3. 에이전트 작성을 완료했으면, 반출 SmartGuide를 사용하여 파일 시스템으로 .class 파일을 반출하십시오.

AgentContext 문서 생성

Domino 5.0에 대해서는 다음 단계를 수행하십시오.

1. Lotus Domino를 여십시오.
2. 사용자 에이전트를 적절한 데이터베이스에 작성하십시오.
3. 에이전트에 대한 세부사항을 채우십시오.
4. **수행** 목록에서, **반입된 Java**를 선택하십시오.
5. **클래스 파일 반입**을 클릭하십시오.
6. VisualAge for Java에서 반입한 .class 파일을 선택하고 **OK**를 클릭하십시오.
7. 저장하고 에이전트를 수행하십시오.

Domino 4.6에 대해서는 다음 단계를 수행하십시오.

1. Lotus Domino를 여십시오.
2. 사용자 에이전트를 적절한 데이터베이스에 작성하십시오.
3. 에이전트에 대한 세부사항을 채우십시오.
4. **에이전트를 수행해야 하는 작업**에 대해 **Java** 라디오 단추를 선택하십시오.
5. **클래스 파일 반입**을 클릭하십시오.
6. VisualAge for Java에서 반입한 .class 파일을 선택하고 **OK**를 클릭하십시오.
7. 저장하고 에이전트를 수행하십시오.

*DebugAgentBase*를 확장하는 클래스에서 사용자 에이전트를 수행할 때 AgentContext 문서가 자동으로 AgentRunner.nsf에 생성됩니다. AgentContext 문서를 생성한 후 *getSession()*를 호출하면 null이 리턴됩니다.

이제 165 페이지의 『에이전트 디버깅』할 준비가 되었습니다.

에이전트 디버깅

AgentContext 문서를 생성했으면, VisualAge for Java에서 이 문서를 디버그할 수 있습니다.

1. 에이전트의 *NotesMain()* 메소드에 중단점을 하나 이상 설정하십시오.
2. 에이전트 이름에서 마우스 오른쪽 단추를 클릭하십시오.
3. **도구**를 선택한 다음 **Domino AgentRunner**를 선택하십시오.

두 가지 옵션이 있습니다.

- 기본 AgentContext 문서로 에이전트를 수행하려면 **수행**을 선택하십시오.

  기본값은 수행한 마지막 AgentContext 또는 저장한 AgentContext입니다. 코드에 중단점을 설정했거나 오류가 있으면 디버거 창이 열려 코드를 step through 할 수 있습니다. 디버거 사용에 대해 자세히 알려면 온라인 도움말을 참조하십시오.

- AgentContext를 수정하거나 다른 AgentContext를 선택하려면 다음을 수행하십시오.
  - **특성**을 선택하십시오. AgentRunner 창이 열립니다.
  - 사용할 AgentContext를 선택한 다음 **에이전트 수행**을 클릭하십시오.
  - 에이전트를 수정하려면 **에이전트 문맥 갱신**을 클릭하십시오.

    에이전트 수행과 **탐색 기준** 필드를 변경하여 디버깅에 사용할 UnprocessedDocuments 집합을 생성할 수 있습니다. 에이전트 수행으로부터 판별할 수 없으므로, 이 정보를 제공해야 합니다.
  - AgentContext 갱신을 완료한 후 **AgentContext 문서 갱신** 단추를 클릭한 다음 창을 닫으십시오.
  - 새로운 AgentContext를 기본 선택사항으로 만들려면 **선택 저장**을 클릭하십시오.
  - **에이전트 수행**을 클릭하십시오.

생산 에이전트 작성

에이전트의 개발이 완료되면, 사용자 에이전트를 Notes로 이동할 수 있습니다.

1. IDE에서 에이전트의 최상위 클래스를 *AgentBase*로 변경하십시오.
2. 에이전트의 .class 파일을 파일 시스템으로 반출하십시오.
3. .class 파일을 Notes로 반입하십시오.

이제 Notes에서 사용자 에이전트를 수행할 수 있습니다.

제12장 JSP/서블릿 개발 환경

JavaServer Pages (JSP)는 서버측 스크립팅 기술로 static 웹 페이지(HTML 문서)에 Java 코드를 삽입하여 페이지를 작동할 때 Java 코드를 실행하게 해 줍니다. 비즈니스 논리(내용 생성)에서 표시 논리(내용 표시)를 분리함으로써 JSP 기술을 사용하면 웹 페이지 디자이너와 Java 프로그래머는 동적 내용을 가지는 HTML 페이지를 쉽게 작성할 수 있습니다.

이 절에서는 다음을 배우게 됩니다.

- JavaServer Pages 기술
- VisualAge for Java에서 JSP/서블릿 개발 환경 작성

개요

Java 서블릿은 웹 서버에 플러그되는 Java 프로그램입니다. 웹 서버는 서블릿 엔진을 통해 서블릿에 대한 호스트로 작동하기 위해 확장될 수 있습니다. 서블릿은 확장성과 신축성이 높아서 클라이언트/단일 서버 응용 프로그램에서 다중 계층 응용 프로그램으로 쉽게 확장할 수 있습니다. 서블릿은 데이터베이스를 웹에 연결할 수 있게 합니다.

VisualAge for Java IDE에서 서블릿을 개발, 디버그, 전개할 수 있습니다. IDE에서 서블릿 객체 내부에 중단점을 설정하고 매번 다시 시작하지 않고도 변경 사항이 실행 중인 서버에서 실행되고 있는 서블릿으로 동적으로 삽입되도록 코드를 step through할 수 있습니다.

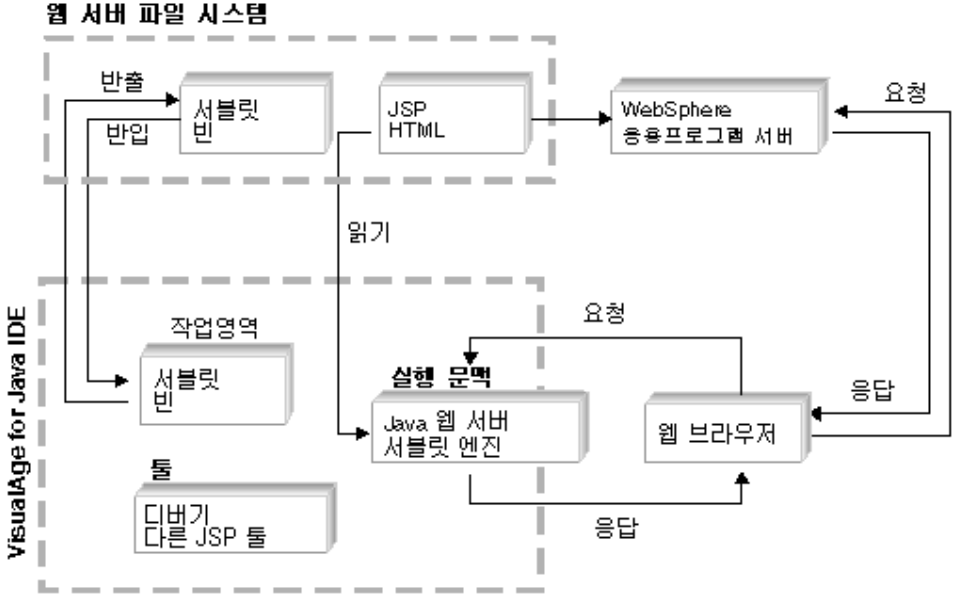
JSP에서 생성한 서블릿 코드가 VisualAge for Java IDE로 반입되면 다음과 같은 현상이 발생합니다.

1. JSP 소스가 페이지 컴파일러로 입력됩니다. 페이지 컴파일러는 Java HTTP 서블릿 같은 실행가능 객체를 만듭니다.
2. 생성된 서블릿 코드가 VisualAge for Java에 반입됩니다. 서블릿을 생성한 JSP를 호출하는 검색기를 사용하여 서블릿을 수행 및 디버그할 수 있습니다. 예를

들어 아래에서 설명하는 SimpleJSP 샘플의 경우 다음을 갱신합니다. http://127.0.0.1:8080/JSP/SimpleJSP/SimpleJSP.jsp

스크립트된 HTML 파일 확장명은 서버에서 JavaServer Page 파일로 식별하도록 .jsp가 됩니다. JSP 페이지를 실행하기 전에 JSP 구문이 분석되고 서버측 객체로 처리됩니다. 그 결과 객체는 동적 HTML 내용을 생성하여 클라이언트로 다시 전송합니다.

JSP 파일은 직접 URL로 만들어지거나, 서블릿에서 호출하거나 또는 HTML 페이지에서 호출됩니다. 세 가지 경우 모두 서블릿 엔진이 JSP를 서블릿으로 컴파일하여 실행합니다. JSP가 처음 요청될 때와 JSP 소스가 변경될 때마다 컴파일됩니다. 요구시 컴파일할 수 있다는 것은 새로운 버전의 JSP 파일을 실행중인 웹 응용프로그램으로 전개할 수 있다는 것입니다. 마찬가지로 서버에 요청될 때마다 서블릿을 컴파일, 로드, 실행할 필요가 없기 때문에 성능도 향상됩니다.



JSP 파일을 쉽게 만드는 방법 중 하나는 WebSphere Site Developer Studio 툴을 사용하는 것입니다. WebSphere Application Server는 웹 응용프로그램을 전개하고 관리하도록 도와주는 IBM의 Java 서블릿 기반 웹 응용프로그램입니다.

WebSphere Application Server는 서블릿, JSP 기술, Enterprise JavaBeans(EJB) 컴포넌트를 사용하는 서버측 Java 프로그래밍 모델에 기초한 웹 서버 플러그인입니다.

VisualAge for Java와 같이 제공되는 도구

VisualAge for Java에는 JSP 개발을 쉽게 하는 다음 도구를 제공합니다.

- **서블릿 실행기.** 서버를 시작하고 검색기를 실행하며 서블릿을 실행하는 툴입니다.
- **JSP 실행 모니터.** 생성된 Java 서블릿과 HTML 소스와 같이 JSP 소스 실행을 모니터하도록 도와주는 툴입니다. 이 툴을 사용하면 생성된 Java 코드, JSP 원본 코드, HTML 출력을 볼 수 있습니다.

JSP 기술을 구현하는 샘플 작업

VisualAge for Java를 설치할 때 사용자 시스템의 문서 루트를 표시하라는 메시지가 표시됩니다.(문서 루트는 HTML 파일과 JSP 파일을 비롯한 웹 자원이 저장된 곳입니다.) 문서 루트 디렉토리가 있고 유효한지 확인하십시오.

주: 문서 루트 디렉토리를 나중에 변경하려면 doc.properties 파일에서 변경할 수 있습니다. 이 파일은 다음 하위 디렉토리에 있습니다.

```
\ide\project_resources\IBM WebSphere Test Environment\properties\server\servlet\httpservice\
```

VisualAge for Java에서 JSP로 작업하려면 저장소에서 VisualAge for Java 작업영역으로 필요한 기능을 먼저 로드해야 합니다. 기능을 설치하려면 **파일** > **빠른 시작** > **기능** > **기능 추가**를 차례로 선택하십시오.

IBM JSP 실행 모니터를 선택하십시오. 다음 기능이 자동으로 설치됩니다.

- IBM WebSphere 테스트 환경
- Sun JSDK 클래스 라이브러리
- IBM 서블릿 IDE 유틸리티 클래스 라이브러리
- IBM 데이터 액세스 빈 1.0 (IBM JSP 예제에서 작업하기 위해 필요)

OK를 클릭하여 작업영역으로 기능을 로드하십시오.

주: IBM JSP 실행 모니터에는 IBM JSP 예제가 들어 있습니다. IBM WebSphere 테스트 환경 기능에는 WebSphere 통합 컴포넌트가 들어 있습니다.

간단한 JSP 샘플로 작업을 시작할 수 있습니다.

간단한 JSP 샘플을 사용하여 JavaServer Page 기술과 JSP 실행 모니터에 익숙해질 수 있습니다. 이 샘플에는 몇 가지 기본 기능을 수행하는 간단한 JSP 스크립트가 들어 있습니다. 하드 드라이브에 설치되어 있는 샘플 인덱스 페이지, *IBM WebSphere JSP 실행 모니터 샘플*에 액세스하려면 다음을 수행하십시오.

1. 웹 브라우저를 시작하십시오.
2. VisualAge for Java IDE를 실행하십시오.
3. WebSphere Application Server를 실행하십시오. 이렇게 하려면 SERunner 클래스(**IBM WebSphere 테스트 환경** > **com.ibm.servlet** 아래의 프로젝트 작업영역에 있는)를 마우스 오른쪽 단추로 클릭한 다음 **수행** > **main 수행**을 선택하여 SERunner 클래스를 수행하십시오. 콘솔 창에서 `ServerProcess:` 서버 시작이 표시되면 서버가 성공적으로 샐행된 것입니다.

   주: 서버가 시작되었는지 확인하기 위해 5-10초간 기다립니다.

   런타임 오류가 발생하면 WebSphere 테스트 환경 창에서 **중단 및 종료**를 클릭하여 프로그램을 종료하고 WebSphere Application Server를 다시 시작하십시오.

   WebSphere Application Server가 VisualAge for Java에서 실행되는 즉시 대상 문서 루트 디렉토리에서 JSP 파일과 HTML 파일을 사용할 수 있습니다.
4. 다음 URL을 입력하여 웹 브라우저로 JSP 샘플 인덱스 페이지를 로드하십시오. http://localhost:8080/JSP/index.htm

   이 페이지에는 간단한 JSP 샘플을 포함하여 네 개의 JSP 샘플로 연결되는 링크가 있습니다.
5. JSP 소스를 step through하려면 JSP 실행 모니터를 작동해야 합니다. (기본값으로 JSP 실행 모니터 모드는 사용하지 않습니다.)
   - IDE Workbench에서 **작업영역** 메뉴 아래에서 **도구** > **JSP 실행 모니터**를 선택하십시오. JSP 실행 모니터 옵션 대화 상자가 나타납니다.

- **JSP 실행 모니터링 사용가능**을 선택한 다음 **OK**를 클릭하여 JSP 파일이 로드될 때 JSP 실행 모니터를 사용하십시오. **취소**를 클릭하여 이전 상태로 돌아갑니다.

6. JSP 실행 모니터를 작동할 때 샘플을 사용하려면 JSP 샘플 색인 페이지로 돌아가서 **간단한 JSP 예제**를 클릭하십시오.

JSP를 구현하는 자세한 예제는 온라인 도움말에 있습니다. 개발자 도메인 등록 샘플은 JSP를 사용하고 데이터를 get 및 set 하는 방법을 보여 줍니다.

부품 번호: CT6DCKO

Printed in Singapore

CT6DCKO