



BlueJ 튜토리얼

한국어 버전 2.0
BlueJ Version 2.0.x 用

English Version 2.0.1

Michael Kölling
Mærsk Insitute
University of Southern Denmark

Korean Version 2.0

황석형 교수
Project Member
Ver. 1.0 : 강석진, 민상호, 오경목, 유형순, 이승환.
Ver. 2.0 : 강원준
선문대학교 컴퓨터정보학부



1. 서문	4
1.1. BlueJ의 소개	4
1.2. 설명범위와 사용자	4
1.3. 복제, 저작권과 배포	4
1.4. 피드백	4
2. 설치	5
2.1. Windows에 설치하는 방법	5
2.2. MacOS에 설치하는 방법	5
2.3. 리눅스/유닉스 및 기타 시스템에 설치하는 방법	6
2.4. 설치할 때 발생하는 문제점	6
3. 시작하기 - 편집/컴파일/실행	7
3.1. BlueJ 시작하기	7
3.2. 프로젝트 열기	8
3.3. 객체 생성하기	8
3.4. 실행하기	10
3.5. 클래스 편집하기	12
3.6. 컴파일 하기	12
3.7. 컴파일러 에러들에 관한 도움말	13
4. 추가 기능	15
4.1. 객체 상태 검사	15
4.2. 매개변수를 통해서 객체 전달하기	18
5. 새 프로젝트 만들기	19
5.1. 프로젝트 디렉토리 만들기	19
5.2. 클래스 만들기	19
5.3. 의존관계 만들기	20
5.4. 요소들의 제거	20
6. 코드패드 사용하기	21
6.1. 코드패드 보기	21
6.2. 간단한 식의 표현	22
6.3. 객체 가져오기	22
6.4. 객체 검사하기	23
6.5. 명령문 실행하기	23
6.6. 멀티라인 명령문들과 일련의 명령문들	24
6.7. 변수를 이용한 명령문 실행시키기	24
6.8. 사용자 입력 이력기능	25

7. 디버깅	26
7.1. 중단점 설정하기	26
7.2. 코드의 단계별 실행	28
7.3. 변수들의 검사	28
7.4. 중지와 종료	29
8. 독립형 어플리케이션 생성	30
9. 애플릿 만들기	32
9.1. 애플릿 실행하기	32
9.2. 애플릿 만들기	33
9.3. 애플릿 테스트하기	33
10. 기타 기능들	34
10.1. BlueJ로 만들지 않은 패키지 오픈하기	34
10.2. 여러분의 프로젝트에 기존의 클래스들을 추가하기	34
10.3. main 메소드와 다른 정적 메소드들의 호출	34
10.4. 문서 생성하기	35
10.5. 라이브러리를 가지고 작업하기	35
10.6. 라이브러리 클래스로부터 객체생성하기	36
11. 요약 정리	37
11.1. 시작하기	37
11.2. 추가 기능	37
11.3. 새프로젝트 만들기	37
11.4. 디버깅	37
11.5. 독립형 어플리케이션 생성	38
11.6. 애플릿 만들기	38
11.7. 기타 기능들	38
12. 번역을 마치며	39

1. 서문

1.1. BlueJ의 소개

본 튜토리얼은 BlueJ 프로그래밍 환경을 사용하는 방법을 설명하고 있습니다. BlueJ는 초급 자바프로그래밍 교육을 위해 설계된 자바 개발 환경으로서, 호주 멜버른의 Monash대학 및 Southern Denmark대학의 BlueJ팀에 의해 설계되고 구현되었습니다.

BlueJ에 관한 보다 상세한 정보는 <http://www.bluej.org>를 참조하기 바랍니다.

1.2. 설명범위와 사용자

본 튜토리얼은 BlueJ 프로그래밍 환경에서 제공하는 제반 기능들에 익숙해지고자 하는 사람들을 위해 작성되었습니다. 따라서, BlueJ 자체의 설계 및 구축, 그리고 관련 기초연구에 관한 사항들은 설명하지 않습니다.

또한, 본 튜토리얼은 자바 언어를 교육하기 위한 목적으로 작성되지 않았습니다. 따라서, 자바 프로그래밍 초보자들은 자바 언어와 관련된 교재 또는 관련 강좌를 참조하여 학습하기 바랍니다.

본 튜토리얼은 BlueJ를 이해하기 위한 참고 메뉴얼이 아닙니다. 따라서, BlueJ의 모든 특징들을 상세히 설명하기 보다는, 기본적인 특징들만을 간략하게 설명하고 있습니다. 자세한 내용은 BlueJ 홈페이지(www.bluej.org)에 *The BlueJ Environment Reference Manual*을 참조하시기 바랍니다.

본 튜토리얼의 모든 절들은 한 줄 정도의 간략한 요약문장으로 시작됩니다. 따라서, 사용자들에게 이미 친숙한 내용은 생략하며 선택적으로 읽을 수 있도록 구성하였습니다. 제 11장에서는 빠른참조(quick reference)를 위하여 전체 내용을 요약 정리해 놓았습니다.

1.3. 복제, 저작권과 배포

BlueJ 시스템과 본 튜토리얼은 어떤 형태로든 누구나 무료로 사용할 수 있으며, 자유롭게 배포할 수 있습니다.

BlueJ 시스템 및 관련 문서들은 저작권자의 허락 없이 판매(패키지화한 형태)될 수 없습니다. BlueJ의 저작권은 M.Kölling 과 J.Rosenberg에게 있습니다.

1.4. 피드백

BlueJ 시스템 및 본 튜토리얼에 대한 코멘트, 질문, 수정, 평가 등을 환영하며, Michael Kölling (mik@mip.sdu.dk)에게 메일을 보내기 바라며, 번역본에 대한 문의는 shwang@sunmoon.ac.kr로 해 주시기 바랍니다.

2. 설치

BlueJ는 Windows, MacOS 그리고 기타 다른 시스템용이 있으며, 설치하는 매우 간단합니다.

<설치를 위한 필요환경>

BlueJ를 사용하기 위해서는 컴퓨터 시스템에 J2SE v1.4(일명, JDK 1.4)이후의 버전이 설치되어 있어야 합니다. JDK가 설치되어 있지 않을 경우 Sun Microsystem사의 웹사이트(<http://java.sun.com/j2se/>)에서 다운로드 받으면 됩니다. MacOS X의 경우, 최신판 J2SE 버전이 이미 설치되어 있습니다(이 경우 여러분이 직접 설치하지 않아도 됩니다). 만약, "JRE"(Java Runtime Environment)와 "SDK"(Software Development Kit)를 모두 제공하는 다운로드 페이지를 찾았다면, "SDK"를 다운로드 받아야 합니다(JRE는 적합하지 않습니다).

2.1. Windows에 설치하는 방법

Windows용 설치파일은 bluejsetup-xxx.exe이며, 여기서 xxx는 버전번호를 의미합니다. 예를 들어, BlueJ 버전2.0.0의 경우, 설치파일은 bluejsetup-200.exe입니다. 여러분은 이 파일을 디스크 형태로 구하거나, BlueJ 웹사이트(<http://www.bluej.org>)에서 다운로드 받아 실행시키면 됩니다.

설치파일을 실행하면, BlueJ를 설치할 디렉토리를 선택해야 합니다. 또한, 바탕화면 및 시작 메뉴에 단축키 설치여부에 관한 설정을 선택해야 합니다.

설치가 종료되면 BlueJ 설치 디렉토리 내에 bluej.exe실행 파일을 찾을 수 있습니다.

BlueJ를 처음 실행시키면 자바시스템(JDK)을 찾게 될 것입니다. 좀더 최신의 자바시스템을 찾으려면(예를 들어 JDK 1.4.2버전이 있다면 JDK 1.5.0를 설치), 여러분이 사용할 버전을 선택 하라는 대화상자가 나타날 것입니다. 찾지 못할 경우에는, 사용자가 스스로 파일의 위치를 찾아야 합니다(이런 경우는, JDK시스템이 설치되었지만, 해당 레지스트리 엔트리가 이미 제거된 경우에 발생합니다). 또한, BlueJ설치파일은 vmselect.exe라는 프로그램을 설치합니다. 이 프로그램을 사용하면 BlueJ가 사용하는 자바 버전을 나중에 변경할 수 있습니다. 즉, vmselect를 실행하여, 다른 자바 버전으로 BlueJ를 실행할 수 있습니다. 어떠한 JDK를 선택 하였는지에 대한 정보가 각각의 BlueJ 버전에 저장됩니다. 서로 다른 버전의 BlueJ를 설치한 경우, JDK 1.4.2 + BlueJ 와 JDK 1.5 + BlueJ와 같이 조합하여 사용할 수 있습니다. BlueJ의 Java버전을 변경하면, 동일한 사용자를 위한 동일 버전의 BlueJ설치가 모두 변경됩니다.

2.2. 맥킨토시에 설치하는 방법

BlueJ는 MacOS X에서만 동작한다는 사실을 명심하십시오.

MacOS용 설치파일은 BlueJ-xxx.sit입니다. 여기서 xxx는 버전번호를 말합니다. 예를 들어 BlueJ버전 2.0.0 설치파일은 BlueJ-200.sit입니다. 여러분은 이 파일을 디스크형태로 구하거나 또는 BlueJ웹사이트(<http://www.bluej.org>)에서 다운로드 받을 수 있습니다. 이 파일은 StuffIt Expander를 사용하여 압축을 풀 수 있습니다. 대부분의 브라우저들이 압축을 풀어 주지만, 그렇지 않은 경우에는 파일을 더블클릭해서 풀어야 합니다. 압축을 푼 후에는

BlueJ-xxx라는 폴더가 생길 것입니다. 이 폴더를 application폴더(또는, 저장하고 싶은 폴더 어디든지)로 이동하면 설치 과정이 완료됩니다.

2.3. 리눅스/유닉스 및 기타 시스템에 설치하는 방법

일반적인 설치파일은 실행 가능한 jar형태의 파일, 즉, bluej-xxx.jar입니다. 여기서, xxx는 버전번호입니다. 예를 들어 BlueJ 버전 2.0.0은 bluej-200.jar입니다. 이 파일은 디스크형태로 구하거나 BlueJ웹사이트(<http://www.bluej.org>)에서 다운로드 받을 수 있습니다.

다음 명령어에 의해 설치파일이 실행됩니다. (아래에서는 설치파일 bluej-120.jar를 사용하는 경우를 말합니다. 여러분들은 알맞은 버전번호가 적힌 파일의 이름을 사용하기 바랍니다.)

```
<j2se-path>/bin/java -jar bluej-200.jar
```

여기서, <j2se-path>는 J2SE SDK가 설치되어 있는 디렉토리를 의미합니다.

윈도우가 화면에 나타나면, BlueJ설치 디렉토리와 BlueJ를 동작시키기 위해 사용할 JDK버전을 선택해야 합니다. 주의 할 점은 BlueJ까지의 경로명(상위 디렉토리중에서 어느 하나)에는 공백을 포함해서는 안된다는 것입니다.

설치(Install)를 클릭하면 BlueJ가 설치됩니다.

2.4. 설치할 때 발생하는 문제점

설치할 때 문제가 발생할 경우,

BlueJ 웹사이트의 FAQ(<http://www.bluej.org/help/faq.html>)를 클릭하여 How To Ask For Help(<http://www.bluej.org/help/ask-help.html>)를 읽어보기 바랍니다.

3. 시작하기 - 편집/컴파일/실행

3.1. BlueJ 시작하기

Windows와 MacOS에서는 인스톨된 BlueJ 프로그램을 실행하십시오.

유닉스 시스템에서는 Installer가 설치된 디렉토리에 **bluej**라는 스크립트 파일을 인스톨합니다. GUI(Graphic User Interface) 환경에서는 **bluej** 파일을 더블 클릭하여 간단히 실행할 수 있습니다. 그리고 명령행에서는 매개변수 없이 **bluej**만을 입력하여 실행하거나 프로젝트 명을 매개변수화하여 실행하는 방법이 있습니다.

`$ bluej`

또는

`$ bluej example/people`

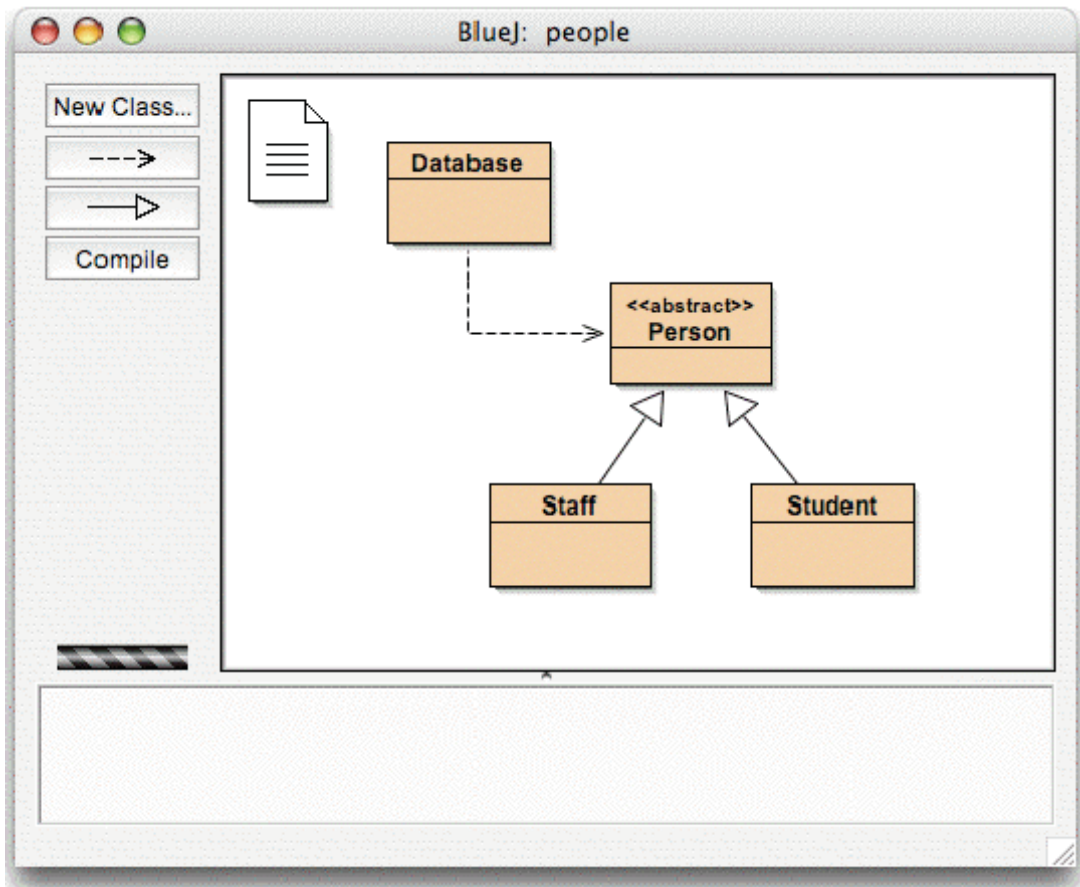


그림 1 : BlueJ 메인 윈도우

3.2. 프로젝트 열기

요약	프로젝트를 열기 위해 Project 메뉴에서 Open 을 선택하십시오.
-----------	-------------------------------------------------------

프로젝트란, 관련 있는 여러개의 파일들을 묶어서 관리하기 위한 하나의 작업 단위를 말합니다. BlueJ 프로젝트는 표준 자바 패키지들처럼 프로젝트에 관련된 파일들을 모아 놓은 디렉토리입니다.

우선 BlueJ를 시작한 후에 **Project - Open...** 메뉴를 선택하여 프로젝트를 여십시오. 표준 BlueJ 배포판의 **examples** 디렉토리에는 몇 개의 예제 프로젝트들이 있습니다.

이 예제들 중 이번 절에서는 **people** 프로젝트를 다룰 것입니다. **examples** 디렉토리에 들어있는 **people** 프로젝트를 선택해서 프로젝트를 오픈하십시오. **examples** 디렉토리는 BlueJ 홈 디렉토리에서 찾을 수 있습니다. 프로젝트를 열면 **그림 1**과 유사한 윈도우를 볼 수 있을 것입니다. 하지만 설치 환경에 따라서 보이는 모습이 차이가 날 수도 있습니다.

3.3. 객체 생성하기

요약	객체를 생성하기 위해서는 클래스 팝업 메뉴에서 생성자 를 선택하십시오.
-----------	-------------------------------------------------------

BlueJ의 기본적인 특징들 중 하나는, 완벽한 어플리케이션을 실행 할 수 있을 뿐만 아니라 하나의 클래스에서 생성된 객체도 public 메소드들을 실행함으로써 사용자와 단일 객체간의 상호작용을 할 수 있다는 것입니다. BlueJ에서의 실행은 객체를 생성하고, 생성된 객체의 메소드중 하나를 호출하는 것을 말합니다. 이러한 실행은 작성 완료된 클래스를 개별적으로 테스트해 볼 수 있기 때문에 어플리케이션 개발에 매우 큰 도움이 됩니다. 따라서, 처음부터 완벽한 어플리케이션을 만들 필요는 없습니다.

노트 : 정적 메소드는 객체를 만들지 않고도 직접 실행할 수 있습니다. 메인 메소드(main method)와 같은 정적 메소드(static method)는 자바 어플리케이션에서 일어나는 일반적인 일들을 (정적 메인 메소드를 실행함으로써 자바 어플리케이션을 실행하기) 할 수 있습니다. 위와 관련된 사항은 나중에 다시 살펴보도록 하겠습니다. 우선, 우리는 자바 환경에서 일반적으로 할 수 없는 보다 흥미로운 몇 가지 다른 것들을 다뤄보겠습니다.

메인 윈도우 중앙의 사각형들은(**Database, Person, Staff, Student**로 명명된 사각형들) 현재 작업중인 어플리케이션에 포함된 클래스들을 아이콘 형식으로 보여주는 것입니다.

클래스 아이콘에서 마우스 오른쪽 버튼을 클릭하면(매킨토시:ctrl+click¹⁾) 클래스에서 적용 가능한 **operation**들이 있는 메뉴를 볼 수 있습니다(**그림 2**). **그림 2**의 팝업 메뉴에는 두개의 생성자 함수들과 BlueJ 개발 환경에서 제공해주는 몇 가지 기능들을 보여줍니다.

1) 이 튜토리얼에서 마우스 오른쪽 버튼 클릭(right-click)은 매킨토시 사용자들에게는 컨트롤 버튼+클릭(ctrl-click)에 해당됩니다.

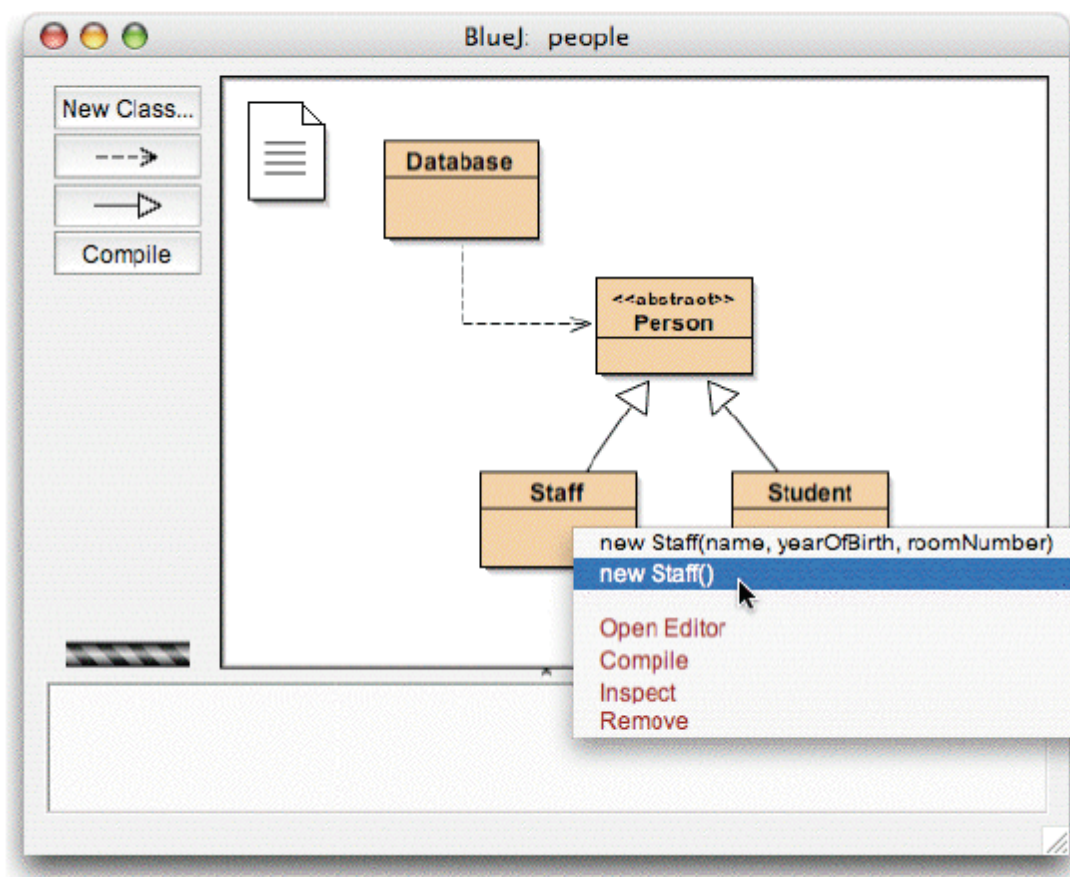


그림 2 : 클래스 기능들 (팝업 메뉴)

Staff 객체를 생성하기 위해서는 **Staff** 아이콘에서 마우스 오른쪽 버튼을 클릭합니다 (그림 2에서 보여지는 팝업 메뉴). 팝업 메뉴는 **Staff** 객체를 생성할 수 있는 두개의 생성자 함수들²⁾을 보여줍니다. 하나는 매개변수들이 없는 생성자 함수이고 다른 하나는 매개변수들을 가지고 있는 생성자 함수입니다. 우선 매개변수들이 없는 생성자 함수를 선택하십시오. 그러면 그림 3과 같은 대화상자가 나타날 것입니다.

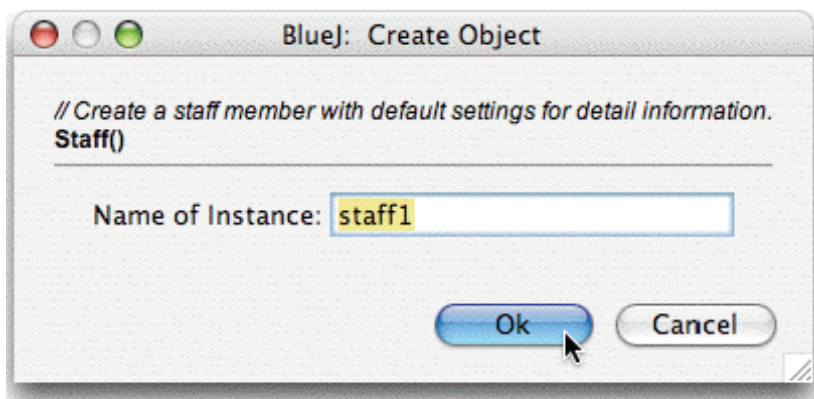


그림 3 : 매개변수 없는 생성자 함수로 객체 생성하기.

2) new Staff(), new Staff(name, yearOfBirth, roomNumber)

이 대화상자는 생성할 객체의 이름을 입력받습니다. 또한 동시에 기본적인 이름(**staff_1**)이 제공됩니다. 이 이름(**staff1**)은 지금 사용하기에 무리가 없기 때문에 바로 **OK** 버튼을 클릭하십시오. 그러면 **Staff** 객체가 생성될 것입니다.

생성된 객체는 오브젝트 벤치(object bench)에 보여지게 됩니다(그림 4). 여기까지가 객체 생성에 관련된 모든 사항입니다. 다시 정리하면, 클래스 팝업 메뉴에서 생성자 함수를 선택하여 실행하면 오브젝트 벤치에 생성된 객체가 보여지게 됩니다.



그림 4 : 오브젝트 벤치에 생성된 객체

Person 클래스 아이콘에는 추상 클래스를 나타내는 <<abstract>> 표기가 있습니다. 자바 언어 명세서에서 정의되었듯이 추상 클래스는 객체를 생성할 수 없습니다. 정말로, 추상 클래스는 객체를 생성할 수 없는지 직접 확인해 보기 바랍니다.

3.4. 실행하기

요약 메소드를 실행시키기 위해서는 객체 팝업메뉴에서 메소드를 선택하십시오.

객체가 생성되면 public 메소드들을 실행시킬 수 있습니다. 오브젝트 벤치에 있는 객체에 대하여 마우스 오른쪽 버튼을 클릭하면 객체 메소드들이 포함된 팝업 메뉴를 볼 수 있습니다(그림 5). 이 메뉴는 클릭한 객체에서 사용할 수 있는 메소드들과 **BlueJ** 환경에서 제공하는 두개의 특별한 기능들(**Inspect**, **Remove**)을 보여줍니다. 이 기능들에 대해서는 나중에 다시 살펴보도록 하고, 우선 메소드에 대해 자세히 살펴보겠습니다.

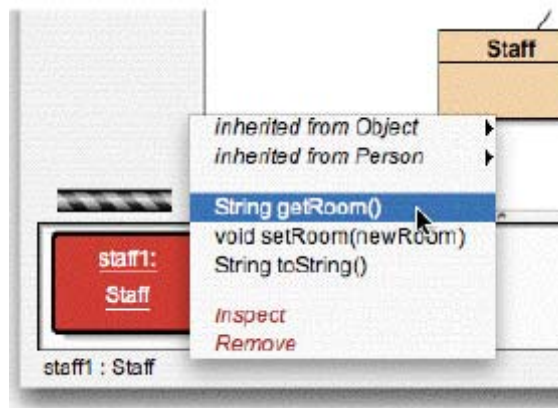


그림 5 : 객체 메뉴

그림 5와 같이 `getRoom`과 `setRoom` 메소드들은 각각 `staff` 멤버의 방번호(room number)를 설정하고 반환하는 동작을 합니다. `getRoom` 메소드를 호출해 봅시다. 객체 메뉴의 `getRoom` 메소드를 선택하여 실행합니다. 그러면 대화상자에서 실행 결과를 볼 수 있을 것입니다(그림 6). 그림 6과 같이 결과의 내용이 "(unknown room)"이 됩니다. 왜냐하면, `Staff` 객체에 대한 방번호를 지정하지 않았기 때문입니다.

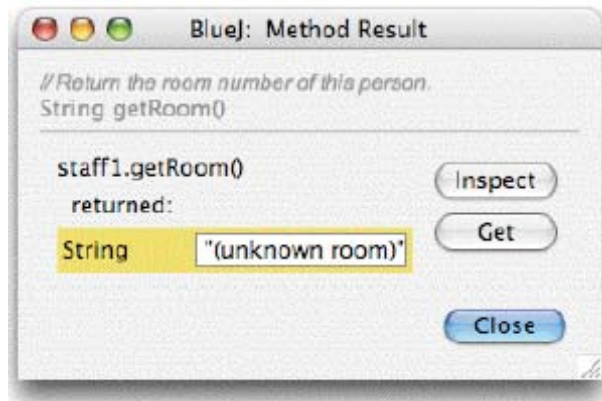


그림 6 : 메소드 호출 결과

슈퍼 클래스에서 상속된 메소드들은 서브 메뉴(inherited from Person)에서 선택하여 사용할 수 있습니다. 객체 팝업메뉴의 상단에는 두개의 서브메뉴³⁾가 있을 것입니다. 하나는 `Object` 클래스로부터 상속 받은 메소드들이고 다른 하나는 `Person` 클래스로부터 상속 받은 메소드들입니다(그림 5). 따라서, 서브 메뉴를 선택하면 `getName`과 같은 `Person` 클래스의 메소드들을 호출할 수 있습니다. `getName`을 호출해 보세요. 그러면 "(unknown name)"이라는 불분명한 결과를 보게 될 것입니다. 왜냐하면 `Person` 클래스의 `name` 속성을 지정하지 않았기 때문입니다.

이제 방번호(를 지정해 봅시다. 매개변수를 갖는 함수 호출 방법을 알게 될 것입니다. (`getRoom`과 `getName` 메소드는 반환값들을 가지지만, 매개변수들은 가지지 않습니다.) 객체 팝업메뉴의 `setRoom` 메소드를 호출하면 매개변수들을 입력받기 위한 대화상자가 나타납니다. 대화상자에 나타나는 프롬프트 위치에 매개변수를 입력하십시오(그림 7).

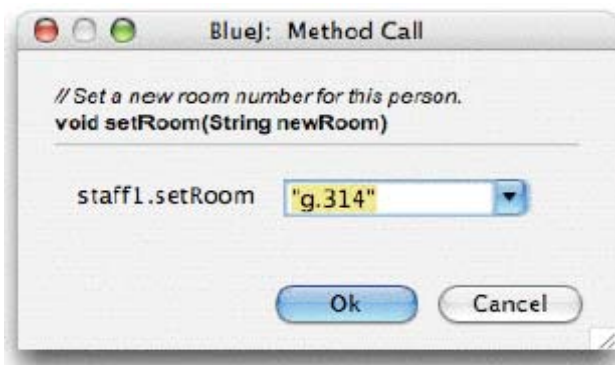


그림 7 : 매개변수들을 입력받는 메소드 호출 대화상자

3) *inherited from Object* ▶
inherited from Person ▶

이 대화상자에서는 호출되어지는 메소드의 인터페이스(주석과 시그니처 포함)를 보여줍니다. 그 아래에는 매개변수 값을 입력할 수 있는 텍스트 필드가 있습니다. 또한, 윗부분의 시그니처에는 매개변수의 데이터형이 **String**형임을 알려줍니다. 이제 텍스트 필드에 새로운 방번호(이중 인용부호를 가진 문자열)를 입력하시고 **OK**버튼을 누르십시오.

위 메소드는 반환 값이 없기 때문에 더 이상의 결과 창이 나타나지 않습니다. 그럼, 정말로 방번호가 변했는지 **getRoom** 메소드를 호출하여 확인해 보십시오.

이밖에 다른 객체 생성과 메소드 호출을 연습해 보기 바랍니다. 익숙해질 때까지, 매개변수들을 가진 생성자 함수를 호출해 보고, 좀 더 다양한 메소드들을 실행해 보십시오.

3.5. 클래스 편집하기

요약	<i>클래스의 소스 코드를 편집하기 위해서는 클래스 아이콘을 더블 클릭하세요.</i>
-----------	-------------------------------------------------

지금까지는 단지 객체의 인터페이스만을 다루어 보았습니다. 지금부터는 클래스 내부 구현 부분을 다루어 보겠습니다. 클래스 기능들 중(클래스 아이콘에서 마우스 오른쪽 버튼을 클릭하면 나타나는 기능들) **Open Editor**를 선택하면 클래스 구현부분을 볼 수 있습니다. 또한 클래스 아이콘을 더블 클릭하여 똑같은 기능을 수행할 수 있습니다.

에디터에 대한 사용방법이 간단함으로 이 튜토리얼에서는 자세히 다루지 않겠습니다. 에디터에 대한 자세한 내용은 후에 별도로 다루어 보겠습니다. **Staff** 클래스의 구현부분을 에디터를 사용하여 오픈한 후 **getRoom**메소드의 구현부분을 찾아보십시오. **getRoom**메소드의 이름에서 알 수 있듯이 **staff** 멤버의 방번호를 반환합니다. 메소드 반환값 앞부분에 **"room"**을 추가하여 반환값을 **"M.3.18"**에서 **"room M.3.18"**로 변경해 보십시오. 즉,

```
return room;
```

을

```
return "room" + room;
```

으로 변경합니다.

BlueJ는 자바를 완벽히 지원하므로 클래스를 구현하는데 있어서 특별한 제한 사항은 없습니다.

3.6. 컴파일 하기

요약	<i>클래스를 컴파일하기 위해서는 에디터에 있는 Compile 버튼을 클릭하십시오. 프로젝트를 컴파일하기 위해서는 프로젝트 윈도우의 Compile 버튼을 클릭하십시오.</i>
-----------	---------------------------------------------------------------------------------------------------------------------

에디터를 수정한 후 메인 윈도우를 확인해 보십시오. 그러면 **Staff** 클래스의 아이콘에 체크 표시가 생겼다는 것을 알 수 있습니다. 체크 표시가 생긴 클래스는 변경된 후에 컴파일 되지 않았음을 나타냅니다. 다시 에디터를 살펴보겠습니다.

노트 : 프로젝트를 처음 열었을 때 왜 클래스 아이콘에 체크표시가 되어있지 않은지 궁금할 수 있습니다. **people** 프로젝트 내의 클래스들은 이미 컴파일 되어있기 때문에 체크표시가 되어 있지 않습니다. 종종 BlueJ 프로젝트들은 컴파일 되지 않은 상태로 배포 되므로, 프로젝트를 처음으로 오픈하면 대부분의 클래스 아이콘들은 체크 표시가 되어있을 것입니다.

에디터의 상단에 있는 툴바에는 자주 사용되는 기능을 수행하는 몇 개의 버튼들이 있습니다. 그중의 하나가 **Compile** 버튼입니다. 컴파일 기능은 에디터에서 해당 클래스를 직접 컴파일 할 수 있게 해줍니다. **Compile** 버튼을 눌러 보십시오. 예러가 없다면 에디터의 아래 부분에 있는 정보창(information area)에 클래스가 컴파일 되었다는 메시지를 출력 할 것입니다. 만약 구문 에러(syntax error)가 생겼다면 에러가 발생한 라인은 반전되고, 정보창에 에러 메시지가 출력될 것입니다. (컴파일을 처음 해보는 경우에는, 세미콜론 빼먹기등과 같은 구문 에러를 직접 만들어 보고 적절한 에러 메시지가 표시되는지 다시 한번 컴파일 하여 확인해 보십시오.)

노트 : 클래스 소스 코드를 반드시 저장할 필요는 없습니다. 소스들은 사용되어질 때마다(예를 들어, 클래스가 컴파일 되기 전이나 에디터가 닫힐 경우) 자동으로 저장됩니다. 단, 사용하는 시스템이 **매우 불안정하고**, 자주 충돌을 일으킨다거나, 작업 내용이 사라질 것이 걱정된다면 **Class** 메뉴에 있는 기능을 이용하여 확실히 저장할 수 있습니다.

클래스를 성공적으로 컴파일 하였다면 에디터를 닫으십시오.

프로젝트 윈도우에 툴바 역시 **Compile** 버튼을 가지고 있습니다. 이 컴파일 기능은 전체 프로젝트를 컴파일합니다. (컴파일이 필요한 클래스들만 올바른 순서대로 재컴파일 합니다.) 두개 이상의 클래스들을 수정한 후에(수정된 클래스 아이콘에 체크 표시가 나타납니다.) **Compile** 버튼을 눌러보십시오. 만약 컴파일 되는 클래스들 중에서 에러가 발생한다면, 에디터 창이 열려 에러가 발생한 위치와 에러 메시지가 표시될 것입니다.

다시 오브젝트 벤치를 보면 객체가 없을 것입니다. 오브젝트 벤치의 객체는 클래스 구현부분이 변경 될 때마다 사라집니다.

3.7. 컴파일러 에러들에 관한 도움말

요약	<i>컴파일러 에러 메시지에 대한 도움말이 필요하다면, 에러 메시지 오른쪽에 있는 물음표를 클릭하십시오.</i>
-----------	----------------------------------------------------------------

자바를 처음 배우는 대부분의 학생들은 자주 컴파일러 에러 메시지를 이해하는데 어려워 합니다. 따라서 몇 가지 도움을 주고자 합니다.

에디터를 다시 오픈하십시오. 소스 코드 파일에 에러를 만들고 컴파일 하면 에러 메시지가 에디터 정보 창에 보여 질 것입니다. 정보창의 오른쪽에 보여지는 물음표를 클릭하면 지금 발생한 에러 메시지에 대한 추가 정보를 볼 수 있습니다(그림 8).

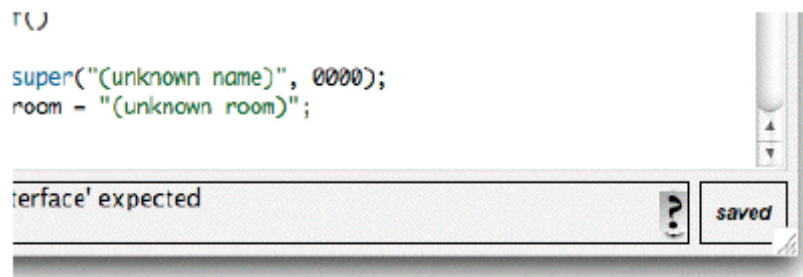


그림 8 : 컴파일러 에러와 Help 버튼

여기서 모든 에러 메시지에 대한 도움말이 제공되는 것은 아닙니다. 일부 도움말은 아직 작성 중입니다. 하지만 이미 많은 에러들이 설명되어 있기 때문에 시도해 볼 가치가 있습니다. 아직 작성되지 않은 도움말들은 추후에 배포될 **BlueJ** 버전에 포함시킬 것입니다.

4. 추가 기능

이번 절에서는 자바 개발 환경에서 사용가능한 몇 가지 추가 기능들에 대해서 살펴보도록 하겠습니다. 여기서 소개할 추가 기능들은 필수적인 것은 아니지만, 매우 빈번하게 사용됩니다.

4.1. 객체 상태 검사

요약	객체 상태 검사는 객체의 내부 상태를 보여줌으로써 디버깅을 도와줍니다.
-----------	-----------------------------------------

객체의 메소드를 실행시킬 때, 그림 5와 같이, 사용자가 정의한 메소드 외에 객체들에 사용가능한 객체 상태 검사 기능을 사용할 수 있습니다. 이 기능은 객체의 인스턴스 변수들(fields)의 상태를 체크해 줍니다. 사용자로부터 값을 입력받아서 객체를 생성해 보십시오(예, 사용자로부터 매개변수의 값을 입력받아서 실행되는 생성자 함수를 이용하여 **Staff** 클래스로부터 객체를 생성). 그런 후에 생성된 객체에 마우스를 올려놓고 오른쪽 버튼을 클릭하여 팝업메뉴에서 **Inspect**를 선택합니다. 그러면, 객체의 필드들과 그들의 데이터 타입, 그리고 필드값들이 표시되는 대화상자가 나타납니다.(그림 9)

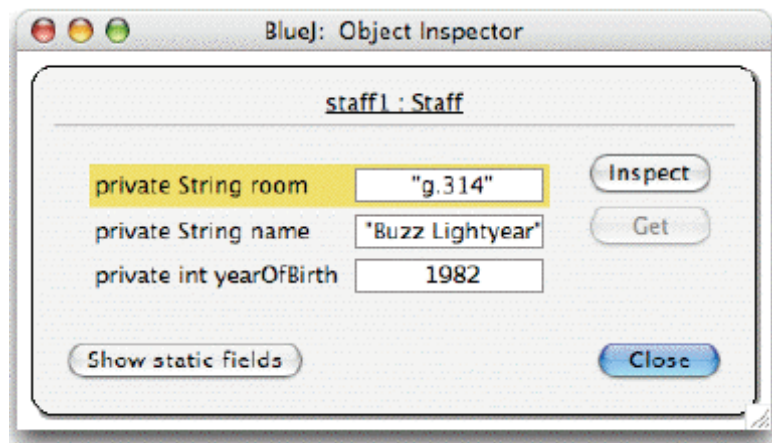


그림 9 : 객체 상태 검사 대화상자

객체 상태 검사는 객체의 상태를 변화시키는 기능이 올바르게 실행되었는지 아닌지를 즉시 체크할 수 있습니다. 따라서, 객체 상태 검사는 간단한 디버깅 도구라고 할 수 있습니다.

Staff 예제에서, 모든 필드들은 간단한 데이터 형들(기본 자료형, 또는 문자열형)로 구성되어 있습니다. 이 데이터 타입들의 값은 직접 볼 수 있어서, 생성자 함수가 올바르게 수행되었는지를 즉시 볼 수 있습니다.

좀 더 복잡한 경우를 살펴보면, 필드들의 값들은 사용자가 정의한 객체들을 참조할 수도 있습니다. 이와 같은 예를 살펴보기위해 여기에서는 다른 프로젝트를 사용해 보겠습니다. 기본적인 BlueJ 배포판에 포함되어있는 **people2** 프로젝트를 엽니다. **People2** 는 그림 10 에서 보여 지고 있습니다. 보다 시피, 이 두 번째 예제에는 이전에 살펴보았던 프로젝트 (**People**)의 클래스들에 더하여 **Address** 클래스를 가지고 있습니다. **Person** 클래스에서 필드 중 하나는 사용자 정의형(**Address**)입니다.

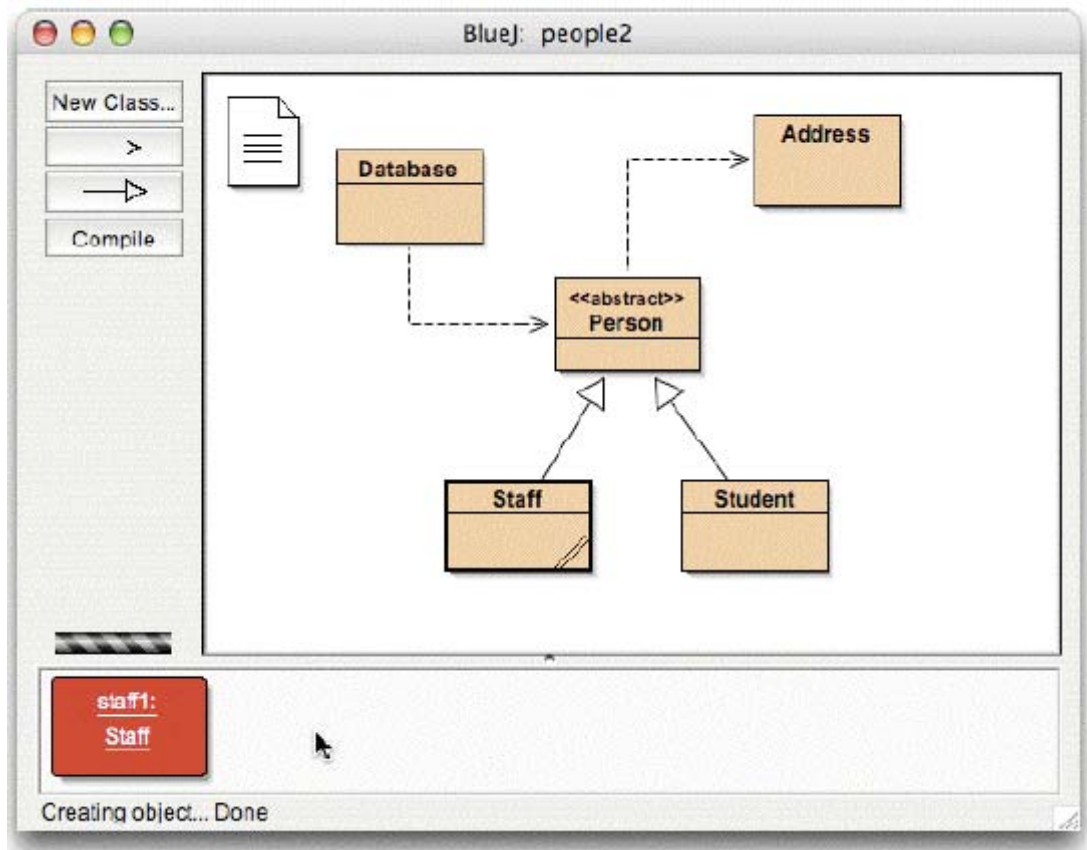


그림 10 : People2 프로젝트 윈도우

객체 필드들을 객체 상태 검사(Inspection)하기 위해서, **Staff** 클래스로부터 객체를 생성하고 이 객체의 **setAddress** 메소드를 호출하십시오(Person의 하위메뉴⁴)에서 **setAddress** 메소드를 발견하게 될 것입니다. 주소를 입력하고나면 내부적으로, **Staff** 클래스에서 생성된 객체가 갖고 있는 코드는 **Address** 클래스의 객체를 생성하고, 그것의 **address** 필드에 **Address** 객체를 저장합니다.

이제, **Staff** 객체를 객체 상태 검사 합시다. 객체 상태 검사 결과가 나타나는 대화상자는 **그림 11**과 같습니다. **Staff** 객체의 필드들 속에는 **address**가 포함되어 있습니다. 보시다시피 **address** 필드의 값은 다른 객체를 참조하고 있음을 화살표로 보여줍니다. 이 필드의 값은 매우 복잡한 사용자 정의형 객체이므로, **그림 11**의 대화상자에 직접 나타낼 수 없습니다. **address** 필드의 값을 좀 더 살펴보기 위해서, **object fields list**에 나타나는 **address** 필드를 선택하고, 객체 상태 검사 버튼을 클릭하십시오. (**address** 필드를 더블 클릭해도 됩니다.) **그림 12**와 같이 또 다른 객체 상태 검사 대화상자가 오픈되면서, **Address** 객체의 세부 사항들이 표시됩니다.

4) **Person** 하위메뉴 : **staff** 객체 생성 후, 마우스 오른쪽 버튼을 클릭하여 나타나는 팝업메뉴에는 **staff** 클래스의 슈퍼클래스(Person)로부터 계승된 메소드들이 나열되어 있습니다.



그림 11 : 객체 참조를 가지는 상태검사

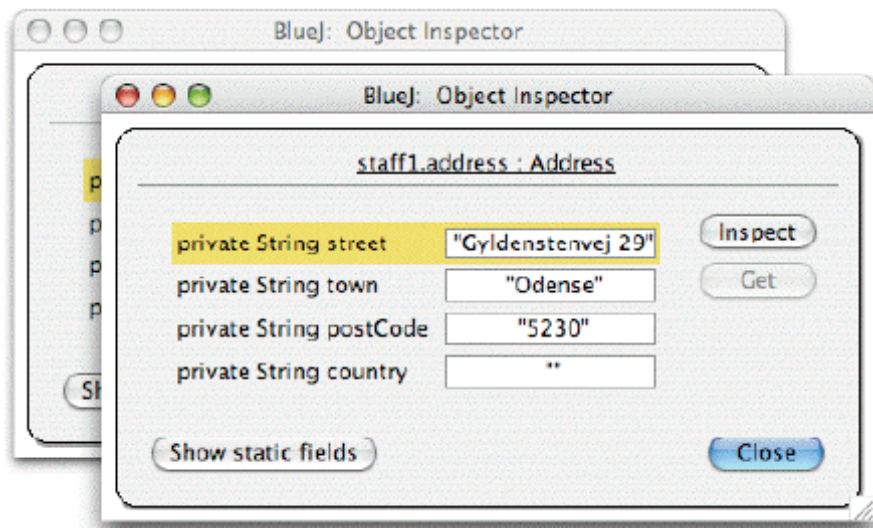


그림 12 : 내부 객체의 상태검사

선택한 필드가 **public**인 경우에는 객체 상태 검사 버튼을 클릭하는 대신에, **address** 필드를 선택하고 **Get** 버튼을 클릭해도 됩니다. 이와 같은 조작을 하게 되면, 선택된 객체가 오브젝트 벤치에 보여지게 됩니다. 오브젝트 벤치에 나타난 객체의 메소드를 호출함으로써, 좀 더 상세한 사항들을 시험해 볼 수 있습니다.

4.2. 매개변수를 통해서 객체 전달하기

요약	<i>객체 아이콘을 클릭하면 메소드 호출의 매개변수를 통하여 객체를 전달할 수 있습니다.</i>
-----------	-------------------------------------------------------

매개변수를 통해서 다른 객체의 메소드로 객체를 전달할 수 있습니다. 예제를 살펴봅시다. **Database** 클래스의 객체를 생성합니다. (**Database** 클래스는 매개변수를 갖지 않는 생성자 함수를 한개 가지고 있으므로, 객체의 생성은 매우 간단합니다.) **Database** 객체는 사람들 (**Person**)의 목록을 보유하는 기능을 갖고 있습니다. **Database** 객체는 **person** 객체들을 추가하는 기능과, 현재 목록에 저장된 모든 사람들을 보여주는 기능을 가지고 있습니다. (**Database**라고 부르는 것은 사실 과장된 표현입니다.)

만약에 오브젝트 벤치에 **Staff**나 **Student** 객체가 없다면, **Staff**나 **Student**의 객체를 생성하십시오. 그렇게 하기 위해, **Database** 객체와 **Staff**나 **Student** 객체가 오브젝트 벤치에서 동시에 필요합니다.

이제 **Database** 객체의 **addPerson** 메소드를 호출합니다. 시그니처에는 **Person**형의 매개변수가 필요합니다. **Person** 클래스는 추상클래스 이므로, **Person**형의 객체를 직접 만들 수 없다는 것을 기억하십시오. 그러나, **Student** 객체나 **Staff** 객체는 **Person** 객체의 서브타이핑(subtyping)이므로, **Person** 객체 대신 사용할 수 있습니다. 그래서, **Person** 객체가 필요한 곳에 **Student** 객체나 **Staff** 객체를 대신해서 사용할 수 있습니다. 오브젝트 벤치에 있는 객체를 매개변수를 통해서, 기동시킨 메소드로 전달하기 위해서는, 매개변수 필드 안에 전달시킬 객체 이름을 입력하는 방법과 전달시킬 객체를 클릭하는 방법이 있습니다. 후자의 방법을 사용할 경우, 메소드 호출 대화상자에 객체 이름이 입력됩니다. **OK**버튼을 클릭하면 함수가 호출됩니다. 이 메소드는 리턴 값이 없어서, 결과를 직접 볼 수는 없습니다. **Database**의 **listAll** 메소드를 호출하여 **addPerson** 메소드가 올바르게 수행 되었는지 확인할 수 있습니다. **listAll** 메소드는 **Person** 객체의 정보를 표준 출력합니다. **Person** 객체의 정보를 보여주기 위한 텍스트 터미널이 자동적으로 오픈되는 것을 알 수 있습니다.

한 개 이상의 **Person** 객체들을 **Database** 객체에 삽입해 보십시오.

5. 새 프로젝트 만들기

이장에서는 새로운 프로젝트를 만드는 방법에 대해서 살펴보겠습니다.

5.1. 프로젝트 디렉토리 만들기

요약	<i>프로젝트를 만들기 위해서는, Project 메뉴에서 New Project를 선택합니다.</i>
-----------	---------------------------------------------------------

프로젝트를 만들기 위해서는, 메뉴에서 **Project - New Project...** 를 선택합니다. 파일입력 대화상자가 열리게 되면 이곳에 프로젝트 이름을 입력하고, 저장할 위치를 선택합니다. 지금 한번 확인해 보십시오. 새 프로젝트의 이름은 어떠한 것을 사용해도 상관없습니다. **OK**버튼을 클릭하시면 앞서 입력했던 이름의 디렉토리가 생성되고, 메인 윈도우에는 새로운 프로젝트가 생성될 것입니다.

5.2. 클래스 만들기

요약	<i>클래스를 새로 만들 때는 New Class 버튼을 클릭하고 클래스 이름을 입력합니다.</i>
-----------	--------------------------------------------------------

또 다른 방법으로서, 프로젝트 툴바에 있는 **New Class** 버튼을 클릭 하여 클래스 이름을 지정하면 새로운 클래스를 만들 수 있습니다. 여기서 클래스 이름은 반드시 적합한 자바 식별자를 입력해야 합니다.

클래스는 4가지 유형(추상클래스, 인터페이스, 애플릿 또는 기본적인 구상클래스) 중에서 선택하여 만들 수 있습니다. 어떠한 유형을 선택하느냐에 따라서, 클래스 내부에 자동적으로 만들어지는 코드들이 결정됩니다. 나중에 클래스내의 소스코드를 편집/변경함으로써 클래스의 유형을 변경할 수 있습니다(예를 들어, **abstract**란 키워드를 코드 안에 삽입하여 추상클래스로 변경).

클래스가 만들어지면 클래스 다이어그램 영역에 아이콘으로 표시됩니다. 만약 그것이 기본 구상클래스가 아닌 경우, 해당 클래스의 유형(인터페이스, 추상 또는 애플릿)이 아이콘에 표시될 것입니다. 새로운 클래스에 대하여 에디터를 오픈해 보면, 기본적인 코드가 자동으로 작성되어 있음을 알 수 있습니다. 이러한 기본적인 코드는 코딩을 쉽게 시작할 수 있도록 도와줍니다. 이러한 기본 코드는 문법적으로 정확하기 때문에 컴파일 될 수 있습니다(그러나 많은 일을 하지는 않습니다). 몇 개의 클래스를 만들어보고 컴파일을 해봅시다.

5.3. 의존관계 만들기

요약	<i>화살표를 만들기 위해서는, 화살표 버튼을 클릭하고 다이어그램영역에서 화살표를 드래그 하거나 에디터를 이용하여 소스코드를 작성합니다.</i>
-----------	----------------------------------------------------------------------------------

클래스 다이어그램에서 클래스간의 의존관계는 화살표로 나타내는데, 상속관계(**extends** 또는 **implements**)는 **더블화살표**($\text{---}\triangleright$)로 보여 주고 **uses**관계는 **싱글화살표**($\text{--}\triangleright$)로 보여 줍니다.

의존관계는 다이어그램영역에서 직접 화살표로 추가하거나 소스코드에서 문자로 추가 할 수 있습니다. 만약 다이어그램영역에서 화살표를 추가 했다면 소스코드에도 자동적으로 의존관계가 입력되고, 소스코드에서 의존관계를 입력했다면 다이어그램영역에서도 화살표가 추가됩니다.

다이어그램영역에서 화살표를 추가하려면, 원하는 **화살표** 버튼(**extends** 또는 **implements**는 더블화살표, **uses**관계는 싱글 화살표)을 클릭하고 하나의 클래스에서 다른 클래스로 화살표를 드래그 합니다.

클래스나 인터페이스의 소스코드 내에 **extends** 또는 **implements** 정의를 입력함으로써 상속 관계 화살표를 추가할 수 있습니다.

uses관계 화살표를 추가할 경우, 화살표가 가리키는 대상이 다른 패키지의 클래스가 아닌 경우, 소스 코드는 바로 변경되지 않습니다. (다만 화살표가 가리키는 대상이 다른 패키지의 클래스인 경우는 **import**문장이 생성됩니다. 그러나 지금까지의 예에서는 다루지 않았습니다). 소스 코드에서 실제로 사용하지 않고 있는 클래스를 가리키는 **uses**관계 화살표를 다이어그램에 추가하면, 「해당클래스가 사용되지 않으면서 **uses**관계가 선언되었음」이라는 경고메시지가 발생합니다.

텍스트로 화살표를 추가 하는 것은 어렵지 않습니다: 보통 때처럼 소스코드를 입력하십시오. 그러면 클래스를 저장하자마자, 다이어그램에도 추가 될 것입니다.

5.4. 요소들의 제거

요약	<i>클래스나 화살표를 삭제하려면, 대상클래스 또는 화살표의 팝업 메뉴에서 Remove를 선택하십시오.</i>
-----------	----------------------------------------------------------------------

다이어그램으로부터 클래스를 삭제 하려면, 삭제하려는 클래스를 선택하고 **Edit** 메뉴로부터 **Remove Class**를 선택하거나, 삭제대상 클래스의 팝업 메뉴에서 **Remove**를 선택하여 삭제할 수 있습니다. 화살표를 삭제할 때는 우선 화살표를 선택한 후 **Edit** 메뉴에서 **Remove** 를 선택하거나 팝업메뉴에서 삭제할 수 있습니다.

6. 코드패드의 사용

BlueJ 코드패드는 자바 코드(표현식과 명령문)의 일부분을 쉽고 빠르게 평가할 수 있는 기능을 제공합니다. 따라서, 코드패드는 자바언어로 작성된 프로그램 코드의 의미를 상세히 조사하거나 구문을 예증하고 시험하는데 사용할 수 있습니다.

6.1. 코드패드 나타내기

요약	코드패드를 사용하기 위해 보기메뉴에서 코드패드보기(Show Code Pad)를 선택하십시오
-----------	----------------------------------------------------

코드패드는 BlueJ 초기실행화면 상에 보여지지 않습니다. 코드패드를 나타내기 위해서는 보기메뉴에서 코드패드보기 메뉴를 선택하십시오. 그러면, BlueJ 메인화면의 오른쪽 아랫편, 즉, 오브젝트 벤치의 오른쪽에 나타납니다(그림 13). 코드패드와 오브젝트 벤치사이에 있는 가로선과 세로선을 조정하여 크기를 바꿀 수 있습니다.

코드패드 영역에는 표현식 또는 문장들을 입력할 수 있습니다. 키보드의 Enter를 누르면 입력된 표현식 또는 문장이 라인단위로 평가되어 그 결과가 화면에 나타납니다.

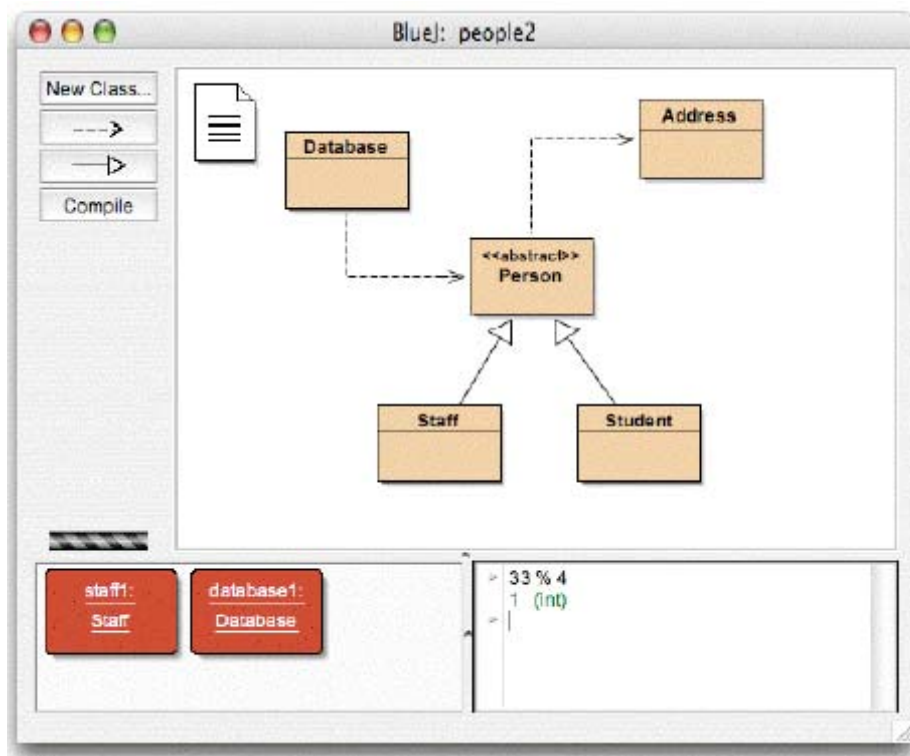


그림 13 : 코드패드를 나타낸 메인화면

6.2. 간단한 표현식의 평가

요약	코드패드에 자바 표현식을 간단히 입력함으로써 평가할 수 있습니다.
-----------	--------------------------------------

코드패드는 간단한 표현식을 평가하는데 사용할 수 있습니다. 다음의 예를 입력해봅시다.

```
4 + 45
```

```
"hello".length()
```

```
Math.max(33, 4)
```

```
(int) 33.7
```

```
javax.swing.JOptionPane.showInputDialog(null, "Name:")
```

표현식은 표준 자바 값과 객체 그리고 현재 프로젝트 내에 작성된 클래스까지도 참조할 수 있습니다. 코드패드는 결과값과 함께 해당 결과값의 타입을 괄호 안에 출력할 것입니다. 만약 잘못된 표현식이 입력된 경우에는 에러 메시지를 출력해줄 것입니다.

또한 오브젝트 벤치위에 생성해 놓은 객체들도 코드패드에서 사용할 수 있습니다. 예를 들면, Student 클래스로부터 한개의 객체를 생성하여 오브젝트 벤치에 올려놓아봅시다(클래스 팝업 메뉴를 이용해 좀더 빠르게 수행할 수 있습니다). 생성된 객체의 이름은 *student1*입니다.

코드패드 안에 다음과 같이 입력할 수 있습니다.

```
student1.getName()
```

또한, 여러분이 작성한 프로젝트에 있는 클래스들이 제공하는 모든 가용한 메소드들을 참조할 수 있습니다.

6.3. 객체 가져오기

요약	코드패드로부터 객체를 오브젝트 벤치로 이동하기 위해서는 객체아이콘을 드래그하십시오
-----------	-----------------------------------------------

코드패드에서는 표현식의 결과값이 단순한 값이 아닌 객체인 경우가 있습니다. 이와 같은 경우, 결과값은 <object reference>에 이어서 객체의 타입(클래스)이 표시됩니다. 또한, 결과값이 표시되는 라인의 선두부분에 작은 객체 아이콘이 나타납니다(그림 14)

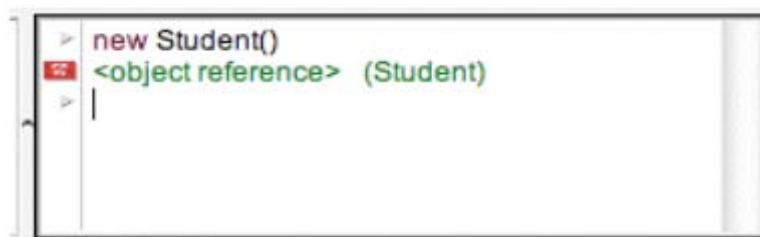


그림 14 : 코드패드에서 객체의 결과를 표시

만약 코드패드의 표현식 평가 결과가 문자열인 경우, 결과값이 출력되어지는 라인에는 문자열의 값과 더불어 작은 객체 아이콘이 출력됨을 알 수 있습니다(왜냐하면, 문자열은 객체이므로).

객체를 생성하기 위한 표현식 몇 가지의 예는 다음과 같습니다.

```
new Student()  
"marmelade".substring(3,8)  
new java.util.Random()  
"hello" + "world"
```

위 표현식을 실행하여 얻어진 결과값(객체)들을 이후의 작업에서 지속적으로 사용하기 위해서는 객체 아이콘을 이용하면 됩니다. 즉, 코드패드에 있는 객체 아이콘을 선택하여 오브젝트 벤치 쪽으로 드래그합니다(그림 15). 이와같이 함으로써 해당객체를 오브젝트 벤치에 위치시킬 수 있게 되고, 오브젝트 벤치에 들어간 객체에 대해서는 팝업메뉴 또는 코드패드를 이용하여 보다 많은 메소드 호출을 실행시킬 수 있습니다.

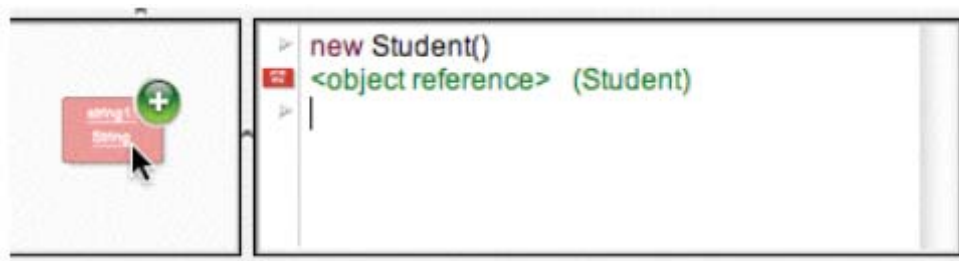


그림 15 : 코드패드에서 생성된 객체를 오브젝트 벤치로 가져오기

6.4. 객체 검사하기

요약	<i>코드패드에서 생성된 객체를 검사하기 위해서는 객체 아이콘을 더블클릭 하십시오.</i>
-----------	----------------------------------------------------

코드패드를 이용하여 표현식을 평가한 결과로써 얻어진 객체에 대하여, 검사를 수행하려는 경우, 검사대상 객체를 오브젝트 벤치에 위치시키지 않고도 검사를 수행할 수 있습니다. 즉, 코드패드에 나타난 객체 아이콘을 더블클릭 함으로써 앞서 설명했던 객체상태검사 화면이 나타납니다.

6.5. 명령문 실행하기

요약	<i>코드패드에 명령문들을 입력하여 실행시킬 수 있습니다.</i>
-----------	--------------------------------------

자바명령문들(값을 리턴하지 않는 명령문들)을 코드패드를 이용하여 실행시킬 수 있습니다. 예를 들어 다음과 같이 입력해 보십시오.

```
System.out.println("Gurkensalat");  
System.out.println(new java.util.Random().nextInt(10));
```

위와 같은 명령문들은 문장의 마지막 부분에 세미콜론을 포함하거나, 포함하지 않은 형태로 평가되어 올바르게 실행됩니다.

6.6. 멀티라인 명령문(여러개의 라인에 이어진 명령문)들과 일련의 명령문들

요약	<i>멀티라인 명령문을 입력하기 위해서는 해당라인의 끝 부분에서 shift-Enter를 사용하십시오.</i>
-----------	--------------------------------------------------------------

일련의 명령문들을 입력하거나 여러개의 라인에 이어진 명령문들(멀티라인 명령문)을 입력하기 위해서는 입력라인의 끝부분에서 *shift-Enter*를 사용하십시오. *shift-Enter*를 사용함으로써, 입력된 내용을 실행시키지 않은 상태로 커서는 다음라인에 시작부분으로 이동하게 됩니다. 마지막 입력라인에 끝부분에서 *Enter*를 입력하게 되면 지금까지 입력한 라인들을 종합하여 평가합니다. 예를 들어 다음과 같은 for문을 실행시켜 봅시다.

```
for (int i=0; i<5; i++){  
    System.out.println("number: " + i);  
}
```

6.7. 변수를 이용한 명령문 실행시키기

요약	<i>(멀티라인)명령문들에는 지역변수가 사용될 수 있습니다. 오브젝트 벤치에 있는 객체들에 이름은 인스턴스 필드의 역할을 합니다.</i>
-----------	------------------------------------------------------------------------------

코드패드에서는 다소 제한된 방법으로 변수(인스턴스 필드와 지역변수)들이 사용될 수 있습니다. 코드패드에서는 지역변수들을 선언할 수 있지만 멀티라인 명령문들에서만 사용할 수 있습니다. 왜냐하면 분리된 입력라인들 사이에서 변수들은 무시되기 때문입니다. 예를 들면, 다음과 같은 문장들을 멀티라인 방식으로 입력하고 실행결과를 예상해 봅시다.

```
int sum:  
sum = 0;  
for (int i=0; i<100; i++){  
    sum+=i;  
}  
System.out.println("The sum is : " + sum);
```


한편, 위의 명령어들을 개별적인 라인입력 방식으로 코드패드에 입력할 경우, 실행결과는 실패합니다. 왜냐하면, 지역변수 *sum*은 각 라인을 처리할 때 마다 기억되지 않기 때문입니다.

사용자가 입력한 내용들은 메소드의 보디부분 내에 있는 코드들로 간주될 수 있습니다. 자바 프로그램의 메소드 보디에 유효하게 작성된 모든 코드들은 코드패드에서도 또한 유효합니다. 그러나, 사용자가 입력한 내용이 서로 다른 메소드의 일부분인 경우 어느 한 입력라인으로부터 다른 입력라인에 선언된 변수를 참조할 수 없습니다.

또한, 오브젝트 벤치에 있는 객체들을 인스턴스 필드로 간주할 수도 있습니다. 사용자들은 어떤 메소드의 보디 내(또는 코드패드 내)에서 새로운 인스턴스 필드들을 정의할 수 없으며, 다만 오브젝트 벤치에 있는 객체들의 인스턴스 필드들을 참조하거나, 메시지를 보낼 수 있습니다.

생성된 객체에 새로운 인스턴스 필드를 추가하기 위해서는 코드패드로부터 해당객체를 드래그하여 오브젝트 벤치에 옮겨놓아야 합니다.

6.8. 사용자 입력 이력기능

요약	<i>입력이력을 사용하기 위해서는 상하 화살표를 사용하십시오.</i>
-----------	----------------------------------------

코드패드에서는 사용자가 이전에 입력한 내용들에 대한 이력을 저장하고 있습니다. 상하 화살표를 이용하여 이전에 입력한 내용을 손쉽게 다시 입력시킬 수 있을 뿐만 아니라, 이전에 입력했던 내용을 수정하여 재입력시킬 수도 있습니다.

7. 디버깅

이 절에서는, BlueJ의 디버깅에 관한 기능들 중에서 가장 중요한 측면을 소개합니다. 컴퓨터 프로그래밍 담당 선생님들은 첫 수업에서 디버거를 사용하는 것이 매우 중요하다고 얘기하곤 합니다. 그러나 정작 디버거에 대해 소개할만한 시간은 없습니다. 학생들은 편집기와 컴파일러, 실행을 하는 것에도 힘겨워합니다. 그래서 다른 복잡한 도구를 소개 할 시간이 없습니다.

위와 같은 이유로 인해서 우리는 가능한 간단한 디버거를 만들 결심을 하였습니다. 선생님이 15분 안에 설명을 할 수 있고, 학생들은 추가적인 다른 설명 없이 바로 사용할 수 있도록 하는 것이 우리의 목표 입니다.

우선, 우리는 전통적인 디버거의 기능들을 3가지로 줄였습니다.

- 중단점(breakpoints) 설정
- 단계별 코드 실행
- 변수 검사

위 3가지 작업들은 아주 간단합니다. 지금부터 하나씩 살펴 보도록 하겠습니다.

시작하기에 앞서, examples 디렉토리 안에 포함된 debugdemo 프로젝트를 열어보십시오. debugdemo 프로젝트는 디버거의 기능을 시연하기 위한 목적으로 만들어진 몇 개의 클래스들을 포함하고 있습니다.

7.1. 중단점 설정하기

요약	<i>중단점을 설정하기 위해서는, 에디터에서 소스코드의 왼쪽부분에 위치한 중단점 영역을 마우스의 왼쪽버튼으로 클릭하면 됩니다.</i>
-----------	----------------------------------------------------------------------------

중단점을 설정함으로써 소스코드의 특정 지점에서 프로그램 실행을 일시 중단 시킬 수 있습니다. 프로그램 실행이 일시 중단 되면, 프로그램 실행에 의해 생성되어 실행중인 객체들의 현재 상태를 조사할 수 있습니다. 따라서, 프로그램 소스코드 내에서 어떤 일이 일어나고 있는지를 이해하는데 도움이 됩니다.

편집기의 소스코드가 보여지는 곳의 왼쪽 부분이 중단점 영역입니다(그림 13). 이 영역을 마우스 왼쪽 버튼으로 클릭함으로써 중단점을 설정할 수 있습니다. 작은 stop 표시가 중단점을 의미합니다. 바로 시도해 봅시다. Demo 클래스를 열어 보십시오. loop 메소드를 찾은 후 loop 메소드 코드 중에서 for 반복문 내의 임의의 라인에 중단점을 설정 하십시오. stop 표시가 에디터에 나타날 것입니다.

```

public int loop(int count)
{
    int sum = 17;

    for (int i=0; i<count; i++) {
        sum = sum + i;
        sum = sum - 2;
    }
    return sum;
}

```

그림 13 : A breakpoint

프로그램 실행 중 중단점에 도달하게 되면 프로그램은 일시 중단 됩니다. 바로 시도해 봅시다.

Demo 클래스에서 객체를 생성하여 매개변수를 10으로 하여 **loop** 메소드를 호출하십시오. 실행 후 중단점에 도달 하면, 소스코드의 현재 라인을 보여주는 에디터 윈도우와 디버거 윈도우가 나타납니다(그림 14).

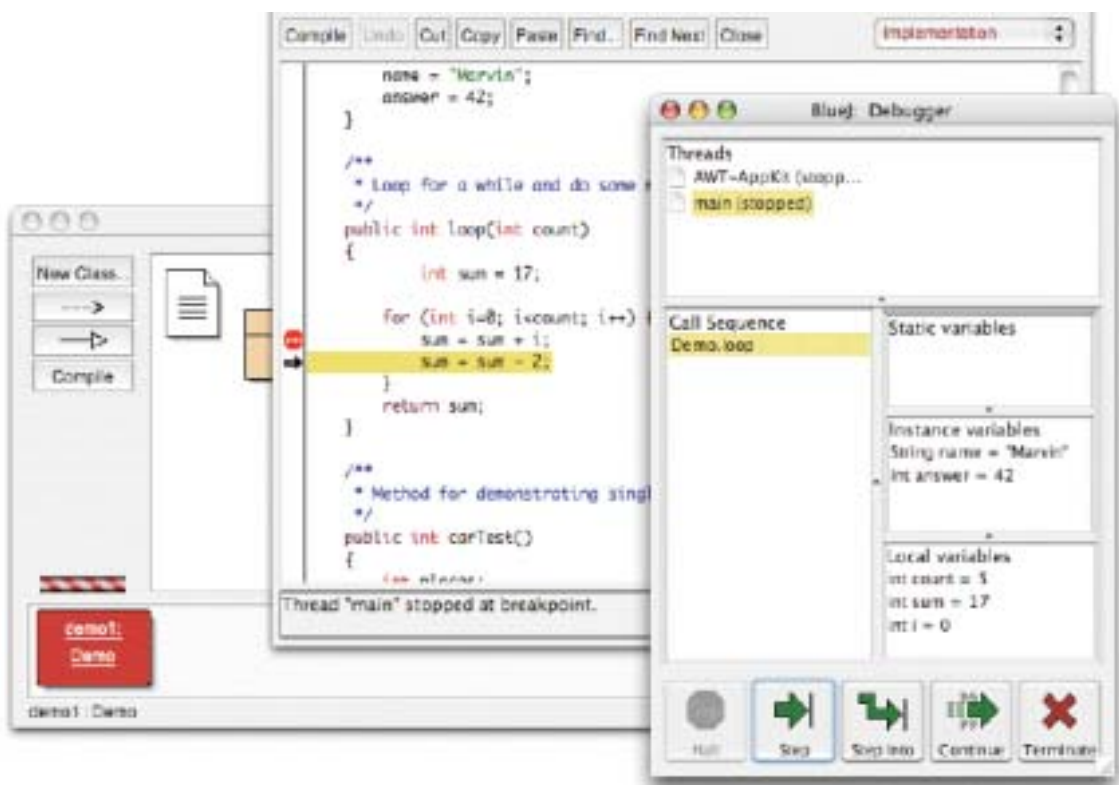


그림 14 : 디버거 윈도우

편집기에서 반전된 부분은 다음 차례에 실행될 라인입니다. (프로그램의 실행은 반전된 라인의 명령문을 실행하기 바로 앞부분에서 정지됩니다.)

7.2. 코드의 단계별 실행

요약	디버거에 있는 Step 버튼과 Step Into 버튼을 사용하여 코드를 단계별로 실행시킬 수 있습니다.
-----------	-------------------------------------------------------------------------

현재 실행을 일시 중지한 상태이며(메소드는 실제로 실행이 되었고 중단점까지 도달했음을 확인할 수 있습니다), 우리는 한 단계씩 코드를 실행시키면서 어떻게 실행이 진행 되는지 살펴볼 수 있습니다. 이와 같이 하기 위해서는 디버거 윈도우의 **Step** 버튼을 반복해서 클릭합니다. 이때 소스코드의 라인이 변화하는 것을 볼 수가 있습니다. (실행이 완료된 라인을 지나서 다음 라인이 반전됩니다.) **Step** 버튼을 누를 때마다 코드의 한 라인이 실행되고 프로그램의 실행이 다시 멈추게 됩니다. 디버거 윈도우에 나타난 변수들의 값들(예를 들면, `sum`의 값)이 변하는 것을 유념하십시오. 단계별로 실행을 시키면서 일어나는 일들을 관찰할 수 있게 됩니다. 위와 같은 단계별 실행을 중지하고 싶을 때는 중단점을 다시 클릭해서 **stop** 표시를 제거하면 되고, 디버거의 **Continue** 버튼을 클릭하여 다시 이어서 실행을 시작할 수 있습니다.

다른 메소드를 가지고 다시 한번 시도해 보겠습니다. 현재 실행하고 있는 `Demo` 클래스의 `carTest()` 메소드 내의 아래와 같은 라인에 중단점을 설정하십시오.

```
places = myCar.seats();
```

메소드를 호출하십시오. 위에서 설정한 중단점에 도달했을 때 실행하려는 라인에는 `Car` 클래스의 `seats()` 메소드를 호출하는 명령어를 포함하고 있습니다. 이때 **step** 버튼을 클릭함으로써 해당 메소드의 모든 라인을 단번에 실행합니다. 이번에는 **Step Into** 버튼을 클릭해 봅시다. 임의의 메소드 호출에 대해서 **Step Into** 기능을 사용하면, 해당 메소드의 구현부분으로 들어가서 한 라인씩 명령어를 실행합니다. 위의 경우, `Car` 클래스에 정의된 `seat` 메소드가 해당 됩니다. 한 라인씩 명령어를 실행하여 끝에 도달하게 되면 메소드를 호출한 `carTest` 메소드쪽으로 돌아가게 됩니다. 디버거가 변화되는 상태를 어떻게 표시하는지 주의해서 살펴봐 주십시오.

현재 라인에 메소드 호출문이 포함되어 있지 않을 경우, **Step** 버튼과 **Step Into** 버튼은 동일한 기능을 수행합니다.

7.3. 변수들의 검사

요약	디버거 윈도우에는 변수들의 상태가 자동으로 변경되므로 변수들을 검사하는 것이 쉽습니다.
-----------	--------------------------------------------------

소스 코드를 디버그 할 때 객체들(지역 변수들과 인스턴스 변수들)의 상태를 검사하는 것은 중요합니다.

객체들의 상태를 검사하는 것은 대부분 계속 보아온 평범한 일입니다. 변수들을 검사하기 위해서 특별한 명령들을 필요로 하지는 않습니다. 현재 객체의 정적 변수들과 인스턴스 변수들과 현재 메소드의 지역 변수들은 항상 자동적으로 화면에 갱신되어 보여집니다.

일련의 함수호출들 중에서 메소드들을 선택하므로써, 현재 활성화되어 있는 다른 객체들 및 메소드들의 변수를 볼 수 있습니다. 예를 들면, `carTest()` 메소드에 중단점을 설정하고 다시 실행을 해 보십시오. 디버거 윈도우의 왼쪽에 순차 호출을 아래와 같이 볼 수 있습니다.

```
Car.seats  
Demo.carTest
```

이는 `Demo.carTest`에 의해 `Car.seats`가 호출된 것을 의미 합니다. 메소드의 현재 변수 값과 소스를 검사하는 리스트에서 `Demo.carTest`를 선택할 수 있습니다.

만약 `new Car(...)` 명령이 포함된 라인을 지나게 되면, <object reference>에서 지역변수 `myCar`의 값을 확인 할 수 있습니다. 문자열을 제외한 모든 객체 형의 값들은 이와 같은 방식으로 보여지게 됩니다. 객체 형의 값을 더블클릭 함으로써 변수를 검사할 수 있습니다. 이렇게 객체 검사 윈도우를 여는 것은 4.1절에서 말했던 내용과 동일합니다. 지금 객체들을 검사하는 것과 오브젝트 벤치의 객체들을 검사하는 것 사이에는 사실 차이가 없습니다.

7.4. 중지와 종료

요 약	<i>중지(Halt)와 종료(Terminate)는 실행을 일시적으로 중지하거나 완전 종료하는데 사용됩니다.</i>
------------	-----------------------------------------------------------------

간혹 프로그램이 오랜 시간동안 실행하고 있으면, 모든 것이 정상인지 의구심이 들게 됩니다. 아마도 무한 루프에 빠져 있거나 아니면 오랜 실행시간을 필요로 하는 프로그램일 것 입니다. 이런 경우에 우리는 프로그램을 점검 할 수 있습니다. `Demo` 클래스의 `longloop()` 메소드를 실행시켜 보십시오. `longloop()` 메소드는 오랫동안 실행됩니다.

무엇이 어떻게 진행되고 있는지 알고 싶어 질 것입니다. 만약 디버거 윈도우가 화면위에 없으면 디버거 윈도우를 오픈하십시오. (한편, 프로그램이 실행되는 동안 컴퓨터가 작동하고 있음을 알려주는 상태 표시바를 클릭하므로써 간단히 디버거를 활성화 할 수 있습니다.)

Halt 버튼을 클릭하세요. **Halt** 버튼의 클릭으로 프로그램의 실행은 중단점을 설정한 것과 같이 일시중단 됩니다. 몇 개의 스텝을 차례로 진행시켜 볼 수 있게 되고 그러면서 변수들을 조사해서 제대로 되어있는지 살펴볼 수 있습니다. 이와 같은 방법을 사용하면 프로그램 전체를 완료 시키는데 시간이 좀 필요합니다. **Continue** 와 **Halt** 버튼을 여러번 클릭함으로써 얼마나 빨리 반복되는지 알 수 있습니다. 만약 계속 진행하기 싫으면(예를 들면, 무한 루프에 빠져 있는 것을 발견하는 경우) **Terminate** 버튼을 눌러서 프로그램 전체 실행을 종료시킬 수 있습니다. 전체 실행 종료는 너무 자주 사용하면 좋지 않습니다. 컴퓨터를 강제 종료 시키게 되면 완벽하게 잘 쓰여진 객체들이 비정상적인 상태로 종료될 수 있으므로, 긴급 상황에서만 전체 실행 종료를 사용할 것을 권장합니다.

8. 독립형 어플리케이션 생성

요약	독립형 어플리케이션을 만들기 위해서는 Project 메뉴에서 Export... 를 사용합니다.
-----------	--------------------------------------------------------------------

BlueJ는 실행가능한 **jar**파일을 생성할 수 있습니다. 실행가능한 **jar**파일은 더블 클릭을 함으로써 Windows나 MacOS X 등에서 실행시킬 수 있으며, Unix나 DOS프롬프트 상에서는 다음과 같은 명령어를 사용하여 실행시킬 수 있습니다.

```
java -jar <파일명>.jar
```

예제 프로젝트 hello를 살펴봅시다. 먼저 예제 디렉토리를 열어서 프로젝트를 컴파일 하고, **Project** 메뉴에서 **Export...**를 선택합니다.

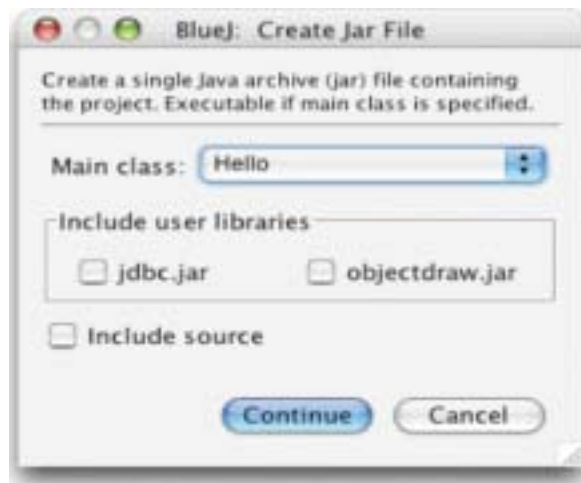


그림 15: Export 대화상자

그림 15와 같이 형식을 지정할 수 있는 대화상자가 나옵니다. **jar file**을 선택하여 실행 가능한 **jar**파일을 생성합니다. **jar**파일이 실행 가능하도록 하기 위해서는 메인 클래스를 지정해야 합니다. 이 클래스에는 다음과 같이 올바르게 정의된 메인 메소드가 있어야 합니다.

```
public static void main(String[] args)
```

위의 예제에서는 단하나의 클래스만 존재하기 때문에 메인 클래스를 선택하는 것은 간단합니다. 팝업 메뉴에서 **Hello**를 선택하십시오. 만약 다른 프로젝트를 실행하고 싶다면 메인 메소드가 있는 원하는 클래스를 선택하십시오.

일반적으로 실행 파일에는 소스코드가 포함되어 있지 않지만, 소스코드를 배포하고 싶다면 사용자가 포함시킬 수 있습니다.(예를 들어, 전자우편을 통해 다른 누군가에게 한개 파일 형태로 프로젝트 내의 모든 파일을 보내려고 할 경우, jar 형식을 사용할 수 있습니다.)

사용자 라이브러리를 사용하기 위하여 BlueJ의 환경을 설정했을 경우(Preferences/Libraries를 설정하거나 lib/userlib 디렉토리를 사용), 대화상자의 중간부분에 Include User libraries 라는 영역이 나타납니다.(어떠한 라이브러리도 사용하지 않을 경우에는 이와 같은 영역은 나타나지 않습니다.) 사용자들은 반드시 현재 작성중인 프로젝트에서 사용하고 있는 모든 라이브러리들을 체크해야 합니다.

Continue를 클릭하십시오. 새로 생성할 jar파일의 이름을 기입할 수 있는 파일 대화상자가 나타납니다. `hello`를 입력하고 **Create**버튼을 클릭합니다.

포함될 라이브러리를 갖고 있지 않을 경우에는, `hello.jar` 파일이 생성될 것 입니다. 만약, 갖고 있을 경우에는 `hello.jar`파일이 포함된 `hello`라는 디렉토리가 생성될 것입니다. 또한, 이와 같이 만들어진 디렉토리에는 모든 필요한 라이브러리들이 포함됩니다. 이와 같이 만들어진 jar파일은 같은 디렉토리 내에서 참조하고 있는 라이브러리들을 찾아내고자 할 것이므로, 다른 곳으로 이동시킬 때에는 관련된 jar파일들을 함께 유지할 수 있도록 주의를 기울여야 합니다.

GUI 인터페이스를 사용하는 어플리케이션의 경우에는 jar파일을 더블클릭 할 수 있습니다. 우리가 살펴보고 있는 예제프로그램은 텍스트 입출력을 사용하므로 텍스트 터미널에서 **jar** 파일을 시작하여야 합니다. **jar**파일을 실행해 봅시다.

터미널이나 DOS 윈도우를 열어 **jar**파일이 저장된 디렉토리로 이동합니다(**hello.jar**파일을 볼 수 있을 것입니다). 자바가 올바르게 설치되어 있다면 다음과 같이 입력함으로써 파일을 실행할 수 있습니다.

```
java -jar hello.jar
```

9. 애플릿 만들기

9.1. 애플릿 실행하기

요약 애플릿을 실행하기 위해 애플릿의 팝업메뉴에서 **Run Applet**을 선택하십시오.

BlueJ는 어플리케이션 뿐만 아니라 애플릿을 만들고, 실행할 수 있습니다. 예제(examples) 디렉토리 안에는 애플릿 프로젝트가 있습니다. 우선 애플릿을 실행하기 위하여, 예제 디렉토리에서 appletdemo 프로젝트를 엽니다.

appletdemo 프로젝트 안에는 클래스(CaseConveter)가 하나 존재합니다. CaseConveter 클래스 아이콘에는 애플릿임을 나타내는 <<applet>>태그가 표시되어 있습니다. 컴파일을 한 후에, 클래스 팝업메뉴에서 **Run Applet**명령어를 선택하십시오.

그러면 몇 가지 선택을 하기 위한 대화상자가 나타납니다(그림 16).

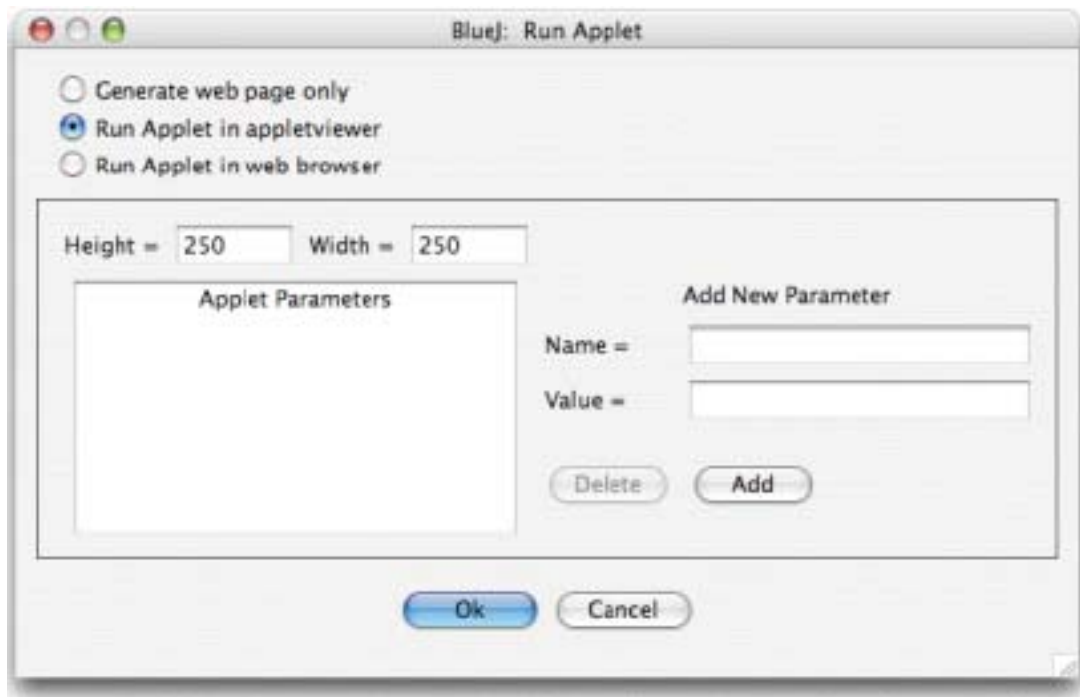


그림 16. Run Applet 대화상자

애플릿을 브라우저에서 실행시킬지 애플릿에서 실행시킬지(또는 애플릿을 실행하지 않고 웹페이지만을 생성시킬지) 선택해야 합니다. 기타 부분은 디폴트 세팅으로 하고, OK 버튼을 클릭합니다. 잠시 후, 애플릿뷰어가 CaseConverter 애플릿을 화면상에 보여줍니다.

애플릿뷰어는 JDK와 함께 설치되어 있기 때문에 여러분이 사용하는 자바 컴파일러와 항상 같은 버전입니다. 따라서, 일반적으로 브라우저에서 실행하는 것보다 문제 발생률이 낮습니다.

여러분이 사용하는 브라우저의 버전에 따라서, 웹브라우저가 다른 버전의 자바환경에 대응할 수도 있으므로, 문제점이 발생할 수 있습니다. 그러나, 대부분의 브라우저에서는 잘 작동됩니다.

마이크로소프트 윈도우즈나 MAC OS시스템에서, **BlueJ**는 여러분의 기본 브라우저를 사용 합니다. 유닉스 시스템의 경우는, **BlueJ**설정 파일 내에 사용할 브라우저가 정의되어 있습니다.

9.2. 애플릿 만들기

요 약	애플릿을 만들기 위해서는 New Class 버튼을 클릭하고, 클래스형으로 Applet 을 선택하십시오.
------------	-------------------------------------------------------------------------

애플릿이 어떻게 실행되는지 알아본 후, 직접 만들어 보기 바랍니다.

일반 클래스처럼 애플릿이 포함된 새로운 클래스를 만드십시오. **New Class** 대화상자에서 타입을 선택할 수 있습니다. 컴파일을 하고 나서 애플릿을 실행시킵니다.

다른 클래스처럼 애플릿은 몇 가지 코드가 포함된 기본 클래스 골격으로 만들어집니다. 이 코드는 2줄의 텍스트로 이루어진 간단한 애플릿을 보여줍니다. 여러분은 이제 에디터를 열고, 여러분의 코드를 삽입하여 애플릿을 편집할 수 있습니다.

일반적으로 애플릿 메소드들은 각각 그 목적을 설명해놓은 주석을 가지고 있습니다. 샘플 코드는 모두 **paint** 메소드 안에 있습니다.

9.3. 애플릿 테스트하기

어떤 경우에는 오브젝트 벤치에서 애플릿 객체를 다른 보통의 클래스처럼 생성하는 것은 매우 유용합니다. 생성자는 애플릿의 팝업메뉴에서 볼 수 있습니다.

오브젝트 벤치에서 여러분은 애플릿을 완벽히 실행할 수 없을 것입니다. 그러나 몇 가지 메소드는 실행할 수 있습니다. 이것은 여러분의 애플릿 구현 부분에서 작성한 한 개의 메소드를 테스트하는 데 사용할 수 있습니다.

만약 여러분이 애플릿에 중단점을 설정했다라도, 애플릿 뷰어나 웹 브라우저에서 애플릿을 실행시켰을 때 어떤 영향도 미치지 않을 것입니다. 왜냐하면, 애플릿 뷰어와 웹 브라우저는 애플릿을 실행시키기 위한 자체 가상머신을 사용하기 때문에, 어떤 BlueJ의 중단점도 인식하지 못합니다.

만약 여러분이 애플릿 내에서 중단점과 한 단계씩 실행 기능을 사용하려면, **Michael Trigoboff**가 만든 *AppletWinodw* 클래스를 이용해야 합니다. 이와 같은 *AppletWindow* 클래스는 BlueJ환경 하에서 애플릿을 직접 실행시킬 수 있게 해줍니다. 따라서, 애플릿에 대한 일반적인 디버깅 작업이 가능합니다. *AppletWinodw* 클래스는 BlueJ 홈페이지에 있는 Resources 메뉴에서 데모와 함께 입수할 수 있습니다.

10.기타 기능들

10.1.BlueJ로 만들지 않은 패키지 오픈하기

요약	<i>BlueJ로 만들지 않은 패키지들도 Open Non BlueJ... 명령으로 오픈할 수 있습니다.</i>
-----------	---------------------------------------------------------------

BlueJ는 BlueJ외부에서 만들어진 패키지들도 오픈할 수 있게 해줍니다. 이것을 하기 위해서는, Project - Open Non BlueJ...메뉴를 선택합니다. 자바 소스파일들이 포함되어 있는 디렉토리를 선택하고, 메뉴에서 Open in BlueJ 버튼을 선택합니다. BlueJ는 이 디렉토리를 열기를 원하는지 다시 한번 물어볼 것입니다.

10.2.여러분의 프로젝트에 기존의 클래스들을 추가하기

요약	<i>Add Class from File... 메뉴를 사용하여 외부에 있는 클래스들을 프로젝트에 복사할 수 있습니다.</i>
-----------	-----------------------------------------------------------------------

BlueJ 프로젝트를 진행하면서 BlueJ프로젝트 외부에 존재하는 클래스를 사용하려는 경우가 있습니다. 예를 들면, 선생님이 프로젝트에서 사용될 자바클래스를 학생들에게 주는 경우가 있습니다. 여러분은 여러분의 프로젝트에서 Edit - Add Class from File... 메뉴를 선택함으로써 받은 클래스를 쉽게 끼워 넣을 수 있습니다. 즉, 이처럼 하게 되면 import시킬 자바 소스 파일(확장자가 .java인 파일들)을 선택할 수 있는 것입니다.

클래스가 프로젝트에 import되면, import된 클래스의 복사본이 만들어져서 현재 프로젝트 디렉토리로 옮겨지고 저장됩니다. 이와 같은 방법은, 여러분이 직접 클래스를 만들어서 모든 소스코드를 작성하는 것과 동일한 결과를 가져옵니다.

또 다른 방법으로써, BlueJ 외부로부터 프로젝트 디렉토리에 새로운 클래스에 해당하는 소스파일을 추가한 후에, 다시 프로젝트를 오픈하면, 그 클래스는 클래스 다이어그램 영역에 포함되어 표시됩니다. 위의 방법은 클래스를 만들어 import할 경우에는 소스파일이, 소스파일을 만들어 추가할 경우에는 클래스가 만들어지는 두 가지 방법 모두를 사용할 수 있다는 것을 말합니다.

10.3.main 메소드와 다른 정적 메소드들의 호출

요약	<i>정적인 메소드들은 클래스의 팝업 메뉴로부터 호출됩니다.</i>
-----------	---------------------------------------

example 디렉토리에서 hello 프로젝트를 여십시오. 프로젝트에 있는 클래스는(클래스 Hello)표준 main 메소드로 정의합니다.

클래스에서 마우스 오른쪽버튼을 클릭하면 클래스 메뉴에서 클래스의 생성자 뿐만 아니라 정적 **main** 메소드가 포함된 메뉴를 볼 수 있습니다. 이제 이 메뉴로부터 처음에 객체를 생성하지 않고 바로 정적 **main** 메소드를 호출할 수 있습니다.

모든 정적 메소드들은 이렇게 호출됩니다. 표준 **main**메소드는 매개변수(입력값)로서 **String**형 배열이 필요합니다.

배열상수라는 조건에 적합하게 표준자바 구문을 사용한 **String**배열을 넘겨줄 수 있습니다. 예를 들면, 아래와 같이(중괄호포함) 메소드에 넘겨줄 수 있습니다.

```
 {"one","two","three"}
```

해보십시오!

노트 : 표준자바에서 배열상수들은 메소드 호출을 위한 실인자(arguments)로서 사용될 수 없습니다. 배열상수들은 초기화에만 사용될 수 있습니다. BlueJ에서는 표준 main메소드의 상호 호출이 가능하도록 하기 위해 매개변수[입력값]로서 배열상수의 사용을 허용합니다.

10.4. 문서 생성하기

요약	<i>프로젝트와 관련된 문서를 생성하기 위해 Tools 메뉴에서 Project Documentation을 선택합니다.</i>
-----------	-----------------------------------------------------------------------

BlueJ를 사용하여 표준 **javadoc**형식으로 프로젝트에 관련된 문서를 생성할 수 있습니다. **Tools - Project Documentation**메뉴를 선택합니다. 이 기능은 프로젝트에 포함되어 있는 클래스들의 소스코드로부터 정보를 채취하여 프로젝트에 사용된 모든 클래스들에 대한 문서를 생성하고, 생성된 문서들을 보여주기 위하여 웹브라우저(web browser)를 엽니다. 또한 BlueJ 에디터를 이용하여 한 개의 클래스에 대한 문서를 생성하여 볼 수도 있습니다.

에디터를 오픈하여 에디터의 툴바(toolbar)에 있는 팝업메뉴를 사용합니다. 팝업메뉴 상단 우측부분에서 **implementation**을 **Interface**로 바꿉니다. 이와 같이 함으로서 에디터 상에서 **javadoc**스타일의 문서(클래스인터페이스)를 볼 수 있습니다.

10.5. 라이브러리를 가지고 작업하기

요약	<i>자바 표준 클래스 API는 Help - Java Standard Libraries를 선택하면 볼 수 있습니다.</i>
-----------	----------------------------------------------------------------------

자바프로그램을 작성할 때에는, 자주 자바표준 라이브러리를 참조해야 합니다. 만약 온라인 연결 상태라면, **Help - Java Standard Classes**메뉴를 선택함으로써 **JDK API** 문서를 웹브라우저에서 열 수 있습니다.

JDK 문서는 또한 오프라인 상태에서 설치하고 사용할 수 있습니다.

자세한 사항은 **BlueJ** 웹사이트의 help부분에 설명되어 있습니다.

10.6. 라이브러리 클래스로부터 객체생성하기

요약	<i>라이브러리 클래스로부터 객체를 생성하기 위해서는, Tools - Use Libraries Class 메뉴를 사용하십시오.</i>
-----------	----------------------------------------------------------------------------

BlueJ는 JDK에서 제공하는 라이브러리의 클래스들로부터 객체를 생성하기 위한 기능을 제공합니다. 예를 들면 여러분은 **ArrayList** 클래스 혹은 **String** 클래스의 객체를 생성할 수 있습니다. 이와 같은 방법을 이용하면, 라이브러리로부터 객체를 생성하여 실습하는데 큰 도움이 됩니다.

Tools - Use Libraries Class 메뉴를 선택함으로써 라이브러리 객체를 생성할 수 있습니다.

위의 메뉴를 선택하면 대화상자가 팝업됩니다. 팝업된 대화상자에는 **java.lang.string**과 같이 완벽하게 인증된 클래스명을 입력해야 합니다. 즉, 패키지 이름까지 포함된 완벽한 클래스 이름을 입력해야 합니다.

팝업메뉴에 표시되는 텍스트 필드값들은 최근에 사용된 클래스들을 보여줍니다.

클래스명을 입력하고, 엔터버튼을 누르십시오. 그러면 대화상자에는 입력된 클래스의 모든 생성자 함수와 정적 메소드들의 리스트가 표시됩니다. 리스트에 표시된 모든 생성자 함수와 정적 메소드들은 단순히 선택함으로써 호출할 수 있습니다. 위와 같이 호출된 생성자 함수와 정적 메소드들은 다른 생성자 함수 또는 메소드 호출보다 우선적으로 실행됩니다.

11. 요약 정리

시작하기

1. 프로젝트를 열기 위해 **Project** 메뉴에서 **Open** 을 선택하십시오.
2. 객체를 생성하기 위해서는 클래스 팝업 메뉴에서 생성자를 선택하십시오.
3. 메소드를 실행시키기 위해서는 객체 팝업메뉴에서 메소드를 선택하십시오.
4. 클래스의 소스 코드를 편집하기 위해서는 클래스 아이콘을 더블 클릭하십시오.
5. 클래스를 컴파일 하기 위해서는 에디터에 있는 **Compile** 버튼을 클릭하십시오.
프로젝트를 컴파일 하기 위해서는 프로젝트 윈도우의 **Compile** 버튼을 클릭하십시오.
6. 컴파일러 에러 메시지에 대한 도움말이 필요하다면, 에러 메시지 오른쪽에 있는 물음표를 클릭하십시오

추가 기능

7. 객체 아이콘을 클릭 함으로서 메소드 호출에 대하여 매개변수를 통하여 객체를 전달할 수 있습니다.
8. 객체 상태검사는 객체의 내부 상태를 보여줌으로써 디버깅을 도와줍니다.

새 프로젝트 만들기

9. 프로젝트를 만들기 위해서는, **Project** 메뉴에서 **New Project**를 선택합니다.
10. 클래스를 만들기 위해서는, **New Class** 버튼을 클릭하고 클래스 이름을 지정합니다.
11. 화살표를 만들기 위해서는, 화살표 버튼을 클릭하고 다이어그램에서 화살표를 드래그 하거나 에디터를 이용하여 소스코드를 작성합니다.
12. 클래스 또는 화살표를 삭제하려면, 팝업메뉴에서 삭제 기능을 선택합니다.

코드패드 사용하기

13. 코드패드를 사용하기 위해 보기메뉴에서 코드패드보기(Show Code Pad)를 선택하십시오.
14. 코드패드에 자바 표현식을 간단히 입력함으로써 평가할 수 있습니다.
15. 코드패드로부터 객체를 오브젝트 벤치로 이동하기 위해서는 객체아이콘을 드래그 하십시오.
16. 코드패드에서 생성된 객체를 검사하기 위해서는 객체 아이콘을 더블클릭 하십시오.
17. 코드패드에 명령문들을 입력하여 실행 시킬 수 있습니다.
18. 멀티라인 명령문을 입력하기 위해서는 해당라인의 끝 부분에서 *Shift-Enter*를 사용하십시오.
19. (멀티라인)명령문들에는 지역변수가 사용될 수 있습니다. 오브젝트 벤치에 있는 객체들에 이름은 인스턴스 필드의 역할을 합니다.
20. 입력이력을 사용하기 위해서는 상하 화살표를 사용하십시오.

디버깅

21. 중단점(**breakpoint**)을 설정하기 위해서는, 편집기에서 소스코드의 왼쪽부분에 위치한 중단점(**breakpoint**) 영역을 마우스의 왼쪽버튼으로 클릭하면 됩니다.
22. 디버거에 있는 **Step** 버튼과 **Step Into** 버튼을 사용하여 코드를 단계별로 실행시킬 수 있습니다.
23. 디버거 윈도우에는 변수들의 상태가 자동으로 변경되므로 변수들을 검사하는 것이 쉽습니다.
24. 중지(**Halt**)와 종료(**Terminate**)는 실행을 일시적으로 중지하거나 완전 종료하는데 사용 됩니다.

독립형 어플리케이션 생성

25. 독립형 어플리케이션을 만들기 위해서는 **Project** 메뉴에서 **Export...**를 사용합니다.

애플릿 만들기

26. 애플릿을 실행하기 위해 애플릿의 팝업메뉴에서 **Run Applet**을 선택합니다
27. 애플릿을 만들기 위해서는 **New Class**버튼을 클릭하고, 클래스형으로 **Applet**을 선택 하십시오.

기타 기능들

28. **BlueJ**로 만들지 않은 패키지 오픈하기 : **BlueJ**로 만들지 않은 패키지들도 **Open Non BlueJ...**명령으로 프로젝트에서 오픈할 수 있습니다.
29. **Add Class from File...**메뉴를 사용하여 외부에 있는 클래스들을 프로젝트에 복사할 수 있습니다.
30. 정적인 메소드들은 클래스의 팝업 메뉴로부터 호출됩니다.
31. 프로젝트와 관련된 문서를 생성하기 위해 **Tools**메뉴에서 **Project Documentation**를 선택 합니다.
32. 자바 표준 클래스 **API**는 **Help - Java Standard Libraries**를 선택하면 볼 수 있습니다.
33. 라이브러리 클래스로부터 객체를 생성하기 위해서는, **Tools - Use Libraries Class** 메뉴를 사용하십시오.

12. 번역을 마치며

© Thanks God. It's a BlueJ.

- 황석형 : shwang@sunmoon.ac.kr