

# Data Structure

**Fall 2021**

**<http://smartlead.hallym.ac.kr>**

**Instructor: Jin Kim**  
**010-6267-8189(033-248-2318)**  
**[jinkim@hallym.ac.kr](mailto:jinkim@hallym.ac.kr)**

**Office Hours:**

# 별을 내 가슴에 (Starry Night)



# Java Style Guide by Google

# 자바 스타일 가이드(Java Style Guide)

- ◆ 자바 작성시 표준화된 스타일로 프로그래밍하면 보다 이해하기 쉬울 것임
- ◆ 구글에서 제공하는 스타일 가이드를 공부하고, 이를 자바 프로그램 제작시 사용하자.

# 명칭부여(Naming)

- ◆ 일반적으로 모든 식별자(identifiers)들은 문자와 숫자로만 이루어져야 한다.
- ◆ 상수는 언더스코어(\_)를 추가하여 사용할 수도 있다.
- ◆ 패키지(package)이름은 모두 소문자.  
연속적인 단어. Ex) mypackage
- ◆ 클래스(class)와 인터페이스(interface) :  
클래스와 인터페이스이름은 대문자로 시작되며 단어의 시작은 대문자.
  - ◆ Ex) ArrayList, Comparable, UpperCamelCase

# Constants(상수)

- ◆ 상수는 **CONSTANT\_CASE** : 단어들은 모두 대문자, 단어의 시작앞에 **\_** 를 사용한다.
  - ◆ Ex) **MAX\_NUMBER\_OF\_PIZZAS**

# Type parameters (형 변수)

- ◆ Type parameter(예를 들면 generic)는 한 문자.

Name	Meaning
E	element
N	number
V	value
K	key
T	type
S, U, V	other

# Everything else(기타)

- ◆ 기타(메소드 이름, 비상수, 매개변수, 변수 이름)은 소문자로 시작하고 새로운 단어의 시작은 대문자로 함.
  - ◆ Ex) lowerCamelCase



# Formatting(포맷팅)

## ◆ Indentation(들여쓰기)

- ◆ 들여쓰기 할 때는 절대 **tab**(탭)을 사용하지 말고, 스페이스를 사용하라.
- ◆ 새로운 블록이 시작될 때는 스페이스를 두 개 사용하라.(우리는 네개. 이클립스는 자동처리)

// Good style

```
public void myMethod(String myFavColor) {  
    if (myFavColor == "Wisteria") {  
        System.out.println("Your favorite color is Wisteria."); // Indented two  
spaces  
    } else { // Indentation level returned to previous one  
        System.out.println("Your favorite color is not Wisteria.");  
    }  
}
```

# Formatting

## ◆ Line Wrapping



- ◆ 한 라인 내의 글자는 80개를 넘으면 안된다.
- ◆ 라인을 넘어 두 줄로 하게되면, 두번째 라인은 네 개의 스페이스 이후 시작한다.

```
aVeryLongArgumentName = anotherEvenLongerArgumentName +  
    anotherStillLongerArgumentNameByJustOneCharacter;  
}
```

# Formatting

## ◆ Vertical Whitespace


- ◆ 다음과 같은 경우 한 라인을 삽입해주어야 한다.
  - 클래스의 선언과 클래스의 끝
  - 클래스내의 멤버들 사이 (필드, 생성자, 메소드 등)
  - 일반적으로 오른쪽괄호())이후 한 라인 삽입.  
절대적아님. If문에서 } 다음에는 한 라인 삽입하지 않음

```
public class MyClass {  
      
    public static void main(String[] args) {  
        System.out.println("Hello world");  
    } // A closing curly brace that ends a method  
      
} // A closing curly brace that ends a class
```

# Formatting

## ◆ Vertical Whitespace

- ◆ 다음과 같은 경우 한 라인을 삽입해주어야 한다.  
Return 문 앞에 **한라인** 삽입. 만일 method 내에 세 개 이내의 문장으로 마감하면 삽입하지 않는다(예외).

```
// Good style: wow that space looks beautiful!  
myVar++;  
  
return myVar;
```

# Formatting

## ◆ Horizontal Whitespace(라인 내에 하나의 공백(space)삽입)

### ◆ 괄호와 키워드 구분하기 위한 공백

```
// Good style   좋은 스타일
for (int i = 1; i < 10; i++)
```

```
// Bad style: no space between opening curly brace and
keyword "for"   나쁜 스타일
for(int i = 1; i < 10; i++)
```

```
// Good style   좋은 스타일
if (val < 10)
```

```
// Bad style   나쁜 스타일
if(val < 10)
```

# Formatting

## ◆ Horizontal Whitespace(라인 내에 하나의 공백(space)삽입)

- ◆ 오른쪽 중괄호(**)** 다음에 한 스페이스 띄움

- ◆ 왼쪽 중괄호(**{**) 시작하기 전 한 스페이스 띄움

```
// Good style
```

```
if (val == 10) {
```

```
    // Do a thing.
```

```
} else {
```

```
    // Do a different thing.
```

```
}
```

```
// Bad style: no space between closing curly brace and  
keyword "else"
```

```
if (val == 10) {
```

```
    // Do a thing.
```

```
}else {
```

```
    // Do a different thing.
```

```
}
```

# Formatting

- ◆ Horizontal Whitespace(라인 내에 하나의 공백(space)삽입)
  - ◆ +, -, \*, / 등의 이진연산자 좌, 우로 하나의 스페이스 띄움

# Declarations(선언)

- ◆ 한 라인에 하나의 변수만을 선언하라.

```
int exampleVar1, exampleVar2; // bad style
```

```
int exampleVar1; //good style
```

```
int exampleVar2;
```

- ◆ 변수의 초기화는 최대한 빨리 하라.

상관없는 코드가 가운데 위치하면 좋지않다.

```
int myVar = 0; //good style
```

or

```
int myVar;
```

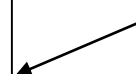
```
myVar = 0;
```

```
int myVar;
```

```
//some unrelated code
```

```
myVar = 0;
```

Bad style





# Statements(문장)

## ◆ Simple statement(단문)

- ◆ 한 라인에는 하나의 명령문만 사용하자

`argv++; // Correct`

`argc- -; // Correct`

`argv++; argc- -; // Avoid! 안좋다`

## ◆ Compound Statements:괄 호안에 단문들 존재

- ◆ 한 라인에 하나의 명령문만을 사용

```
... {  
    statement1;  
    statement2;  
    ...  
    statementn;  
}
```

# Statements(문장)

## ◆ return statement

- ◆ Return 문에는 일반적으로 ()를 사용하지 말것.  
특별히 값을 명확하게 할 경우만 () 사용

```
return;
```

```
return myDisk.size();
```

```
return (val > 17 && val < 19); //특별한 경우
```

# Statements(문장)

## ◆ If , if-else, if else-if else statement

```
if (condition) {  
    statements;  
}  
  
if (condition) {  
    statements;  
} else {  
    statements;  
}  
  
if (condition) {  
    statements;  
} else if (condition) {  
    statements;  
} else {  
    statements;  
}
```

# Statements(문장)

## ◆ For statement

```
for (initialization; condition; update) {  
    statements;  
}
```

## ◆ For each statements

```
for (Type identifier : iterableCollection) {  
    statements using identifier;  
}
```

## ◆ While statements

```
while (condition) {  
    statements;  
}
```

# Statements(문장)

## ◆ Do while statement

```
do {  
    statements;  
} while (condition);
```

## ◆ Switch statements

```
switch (condition) {  
    case ABC:  
        statements;  
        /* falls through */  
    case DEF:  
        statements;  
        break;  
    case XYZ:  
        statements;  
        break;  
    default:  
        statements;  
        break;  
}
```

# Statements(문장)

## ◆ Try-catch statement

```
try {  
    statements;  
} catch (ExceptionClass e) {  
    statements;  
}
```

```
try {  
    statements;  
} catch (ExceptionClass e) {  
    statements;  
} finally {  
    statements;  
}
```

# Comments(주석)

## ◆ Block comments

```
statements;
```

한라인 띄움

```
/**
```

```
* Here is a block comment preceded by a blank line,  
* and followed immediately by code.
```

```
*/ // 한라인 띄우지 않고 명령문 사용
```

```
int myVar = 0;
```

# Comments(주석)

## ◆ Single-Line comments

```
if (condition) {  
  
    /* This is what is done to handle the condition. */  
  
    . . .  
}
```



# Comments(주석)

## ◆ End-Of-Line comments

- ◆ 주석이 한 라인으로 끝날 수 있으면 // 도 좋다

```
if (foo > 1) {  
    // Do a double-flip.  
    ...  
} else {  
    return false; // Explain why here.  
}
```

# Documentation Comments(주석)

- ◆ 자바 클래스, 인터페이스, 생성자, 메소드, 변수등에 대한 주석등은 다음 처럼 하자
- ◆ `/** */`

```
/**  
* The Example class provides ...  
*/  
public class Example {  
    ...  
}
```

# Documentation Comments(주석)

```
/**
 * A class that handles writing to a specific file
 *
 * Use the write() method to write to a file
 */
public class WriterClass {

    private File myFile;

    /**
     * Constructs a WriterClass object.
     * @param fileName - the file name to write to
     */
    public WriterClass(String fileName) {
        this.File = new File(fileName);
    }

    /**
     * Writes a string to a file specified by a file name
     *
     * @param str - the string to write
     * @param file - the file name to write to
     *
     * @throws IOException
     * @return false if the file is created by this method, true otherwise
     */
    protected boolean write(String str) throws IOException {
        boolean fileExists = myFile.exists();

        // FileWriter throws an IOException when the target file cannot be opened
        BufferedWriter writer = new BufferedWriter(new FileWriter(myFile));
        writer.write(str);

        return fileExists;
    }
}
```

# Documentation Comments(주석)

@의 의미

@다음은 이미 지정된 예약어로의 기능

@author 개발자

@exception 메소드에서의 예외 확인

@param 메소드의 매개변수

@return 메소드의 반환값

@see 다른 주제에 관한 링크 지정

@serial 직렬화 필드

@since 릴리즈 기록

@throws 메소드에서의 예외

@version 클래스의 버전

# File Structure

## ◆ Headers

- ◆ : 자바 코드의 맨 처음 라인. 거의 package 선언이 첫 라인. 80자를 넘어도 되며, 모두 소문자

```
package exampleofapackagenamehatishverylongbuttheeightycharacterlimitdoesnotapply;  
;
```

- ◆ import 선언이 보통 두번째. Wild card(\*)를 사용하지 말라.

```
// Bad style: don't import wildcards!  
import java.util.*;  
  
// Good style: only import exactly what you need  
import java.util.List;  
import java.util.Map;
```

# File Structure

## ◆ Headers

- ◆ Package 와 import문 사이에 한 라인을 사용하라.

```
// Good style: eminently readable
package myexcellentpackage;

import java.util.List;
import java.util.Map;
```

# File Structure

## ◆ Class Structure

- ◆ Main 함수는 가장 나중에, 생성자(constructor)는 되도록 가장 먼저

```
/**
 * A class for awesome things.
 *
 * Use the makeAwesome method to make things the most awesome they can be
 * .
 */
public class MyAwesomeClass {

    private int count;
    private boolean isAwesome;
    private final static String AWESOME_STRING = "Super-totally-radical-
        groovy";

    /**
```

```

    * A class for creating an AwesomeException.
    */
    private class AwesomeException extends Exception {
        /**
         * Constructor for a MyAwesomeClass.
         */
        public AwesomeException() {
            super("This is not so awesome, unfortunately!");
        }
    }

    /**
     * Constructor for a MyAwesomeClass.
     * @param count - number of awesome strings
     * @param isAwesome - true if it is awesome, false otherwise
     */
    public MyAwesomeClass(int count, boolean isAwesome) {
        this.count = count;
        this.isAwesome = isAwesome;
    }

    /**
     * Proclaims how awesome things really are.
     *
     * @throws AwesomeException when something not so awesome happens
     */
    public void howAwesome() throws AwesomeException {
        if (!(this.isAwesome)) {
            throw new AwesomeException();
        }

        for (int i = 0 ; i < this.count; i++) {
            System.out.println(AWESOME_STRING);
        }

        this.isAwesome = true;
    }

    public static void main(String[] args) {
        MyAwesomeClass awesome = new MyAwesomeClass(18, true);

        try {
            awesome.howAwesome();
        } catch (AwesomeException e) {
            System.out.println(e.getMessage());
        }
    }
}

```



Java

특이한 사항

# 향상된 for 문

## ■ 배열을 순회하는 일반적인 방법

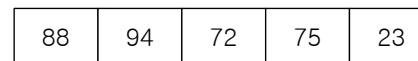
- for (int i = 0; i < score.length; i++) {
- ....
- }

## ■ 첨자로 순회하고 score[i] 식으로 요소를 액세스한다. 요소를 순서대로 읽을 때는 다음 형식을 사용한다.

- for (타입 변수 : 배열) {
- 명령
- }

## ■ 배열 요소가 순서대로 제어 변수에 대입된다. 제어 변수는 요소값의 사본이다

```
int[] score = { 88, 94, 72, 75, 23 };  
int sum = 0;  
for (int s : score) {  
    sum += s;  
}
```



배열 요소값이 순서대로  
제어 변수에 대입된다.

```
for (int s : score) {  
    sum += s;  
}
```

# 메모리의 종류

- 자바는 메모리를 세 부분으로 나누어 관리한다.
- 정적 영역 : 클래스가 로드된다. 실행 코드가 있는 영역이며 변경되지 않는다.

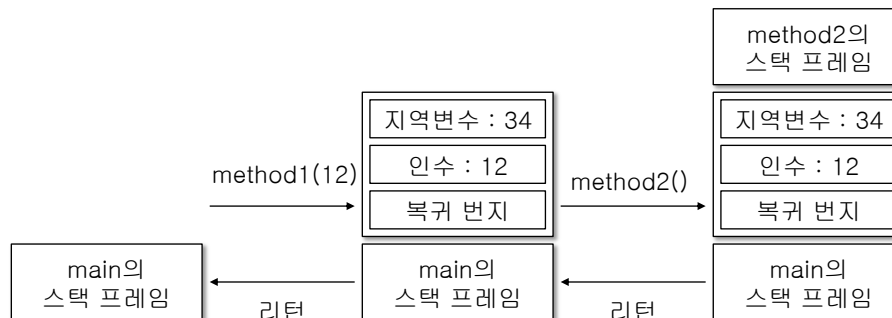
```
class1
static int count = 0;
void method1() {
    if (...) {
    }

    for (...) {
    }
}

class2
void method2() {
    a = b + c;
    println(...);
}

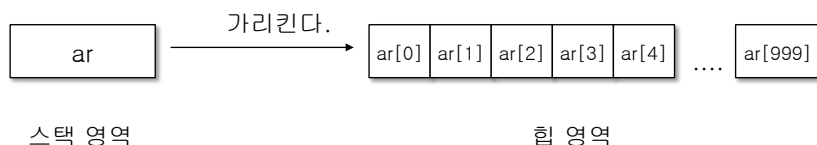
void method3(int a) {
    switch(a) {
    case 1:
        ....
        break;
    }
}
```

- 스택 : 실행에 필요한 정보를 저장하는 영역이다. 지역 변수가 이 영역에 생성되며 다 사용한 후 제거된다.



# 메모리의 종류

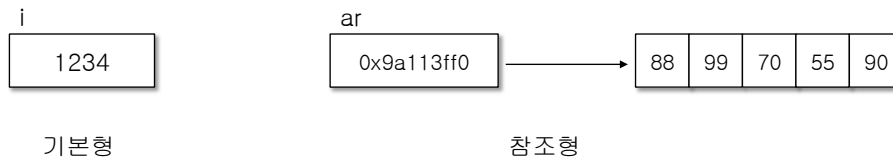
- 힙 : 큰 객체를 생성한다. 스택은 1M의 용량 한계가 있어 큰 객체를 저장할 수 없다.
- 배열은 힙에 저장한다.



- 힙은 물리적인 메모리의 영역을 모두 사용할 수 있어 크기 제한이 거의 없다.
- 사용이 끝난 객체는 가비지 컬렉터가 주기적으로 회수한다.
- 다 쓴 객체는 해제할 필요 없고 쓴 후에 내버려 두면 된다.

# 기본형과 참조형

- 기본형은 크기가 작아 스택에 생성된다.
- 참조형은 힙에 저장하고 시작 번지만 변수에 저장한다. 위치를 찾을 수 있는 참조값만 가진다.



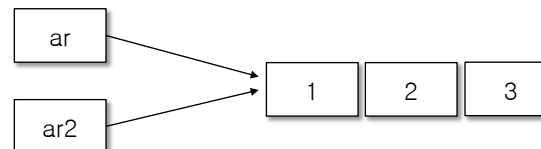
- 기본형은 사본 생성시 완전히 독립적이다.

```
int i = 1;  
int i2 = i;  
  
i2 = 1000;
```



- 참조형은 사본을 변경하면 원본도 바뀐다.

```
int[] ar = { 1, 2, 3 };  
int[] ar2 = ar;  
  
ar2[0] = 1000;
```



# 객체와 인스턴스

- 객체와 인스턴스는 거의 같은 의미이며 문맥에 따라 적당히 구분하여 사용한다.
- 클래스는 빵틀, 객체는 빵
- 인스턴스는 클래스가 메모리에 할당된 실체, 객체(Object)는 프로그램을 구성하는 독립적인 부품
- 클래스는 타입(자료형)이고 객체는 변수이다.



- Car 클래스로부터 무수한 자동차를 찍어낼 수 있다.
- 클래스 선언 순서는 중요치 않다.
- 클래스마다 별도의 소스를 작성하는 것이 원칙이다.

# 실습 관련

- ◆ 한 클래스는 한 파일이 일반적이지만, 자료구조 실습의 경우 **한 파일에 모든 클래스를 적도록 약속하자.**
- ◆ 파일을 업로드할 경우 **\*.java** 파일만을 업로드하라. 다른 파일들은 필요없다.

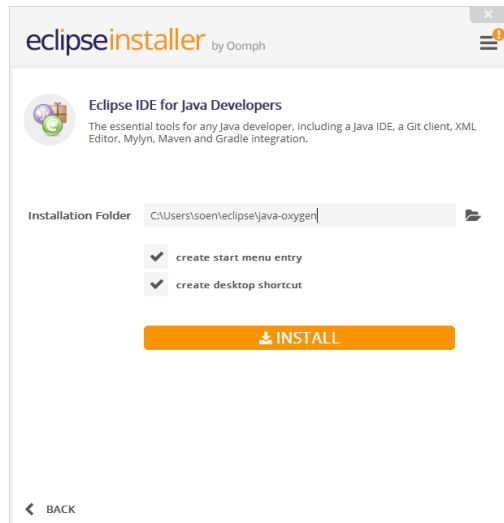
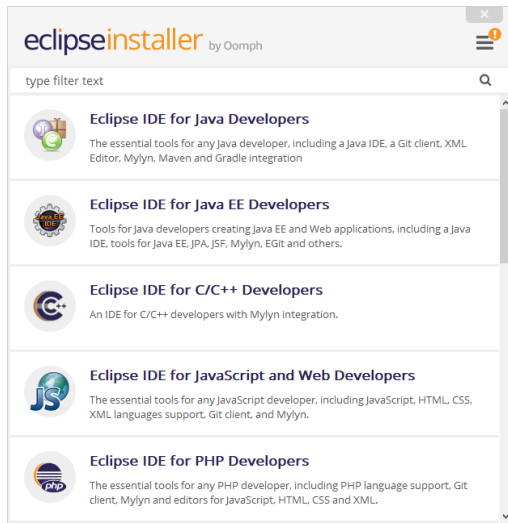
# Eclipse Editor



# Eclipse 사용 요령 및 단축 키

# 이클립스

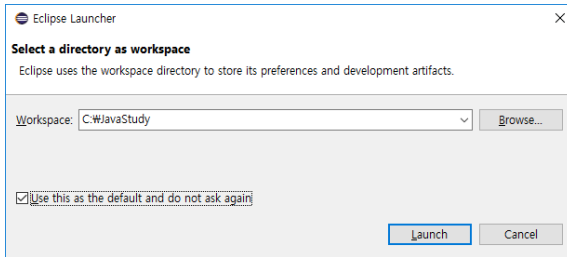
- 전통적인 자바 통합 개발 환경이다.
  - <http://www.eclipse.org>
- 최신 버전은 **SimRel(4.9.0)**이되 18년 12월 버전이 발표되었다.
- 설치 프로그램을 다운로드받아 실행하고 **Java Developers** 버전을 설치한다.



- 다운로드 설치가 번거로운 환경에서는 패키지를 미리 받아 압축만 풀어 주면 된다.

# 이클립스

- 최초 실행시 워크스페이스를 물어 오는데 실습 폴더로 지정한다.



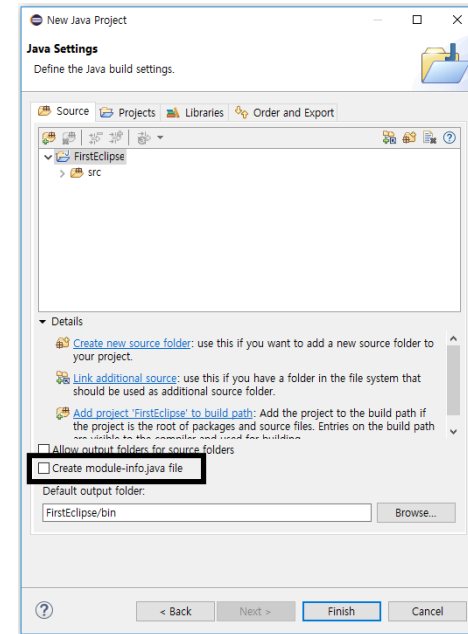
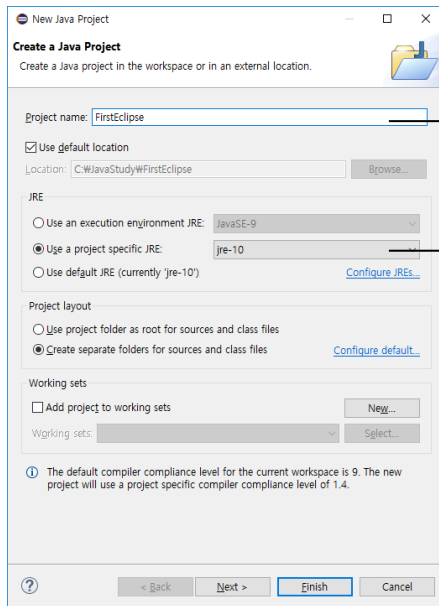
- 한글 사용을 위해 **Windows/Preferences/General/Workspace**에서 문자셋 옵션을 **UTF-8**로 변경한다. Default는 MS949로 되어있다.
- **Java/Compiler** 페이지의 레벨은 10 이상으로 맞춘다. 8 이상이면 무관하다.
- 글꼴이나 단축키는 취향에 따라 조정한다.
- **General/Appearance/Colors and Fonts** 페이지에서 Text font 항목의 폰트를 변경한다.
- 이클립스도 따로 배워야 할 정도로 부피가 크다.

# 배경화면 변경

- 이클립스 상단의 **Help-Eclipse Marketplace**를 선택한다.
- 검색창에 **color theme**을 검색한다. 원하는 **theme**를 선택하여 설치한다.
- 이클립스를 재시작한다.
- 재시작이 완료되면 **Window-Preferences**를 선택한다.
- **General - Appearance - Color Theme** 를 선택한 후 원하는 **theme**를 선택하여 적용한다.

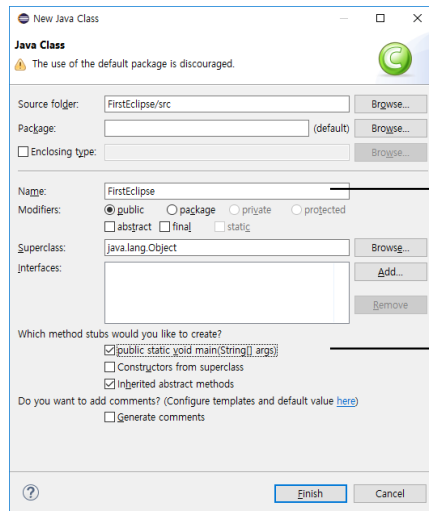
# FirstEclipse

- **File/New/Java Project** 항목을 선택하고 프로젝트명을 **FirstEclipse**로 선택한다.
- **Next** 버튼을 눌러 **Create module-info.java file** 옵션을 해제한다. 빈 프로젝트가 생성된다.



# FirstEclipse

- **File/New/Class** 를 선택하고 소스 파일명은 **FirstEclipse**로 입력한다.
- **public static void main(String[] args)** 옵션을 선택한다.



① 소스 파일 이름 입력

② main 옵션 선택

# FirstEclipse

- 새로 만든 소스 파일에 코드를 입력한다.
  - `class FirstEclipse {`
  - `public static void main(String[] args) {`
  - `System.out.println("Java Example Program");`
  - `}`
  - `}`
- **Ctrl+S**를 눌러 저장하면 컴파일된다.
- **Ctrl+F11**을 눌러 실행하면 아래쪽 **Console**창에 실행 결과가 나타난다.
- 에러 발생시 **Problems**창에 발생 위치와 원인이 표시된다. 수정 후 다시 컴파일한다.
- 실행전 항상 저장 옵션을 선택해 두면 수정 후 바로 컴파일해 볼 수 있다.

# Eclipse 화면

## ■ **Perspective(퍼스펙티브)-화면구성을 의미**

- 퍼스펙티브는 특정한 자원을 가지고 특정한 작업을 수행하기 위한 기능의 집합
- Java perspective는 자바 소스 파일을 편집할 때 필요한 뷰들을 제공
- 디버그 퍼스펙티브(Debug Perspective)는 자바 프로그램을 디버깅할 때 필요한 뷰들을 제공



# Eclipse Perspective

Menu bar

Tool bar

Perspective  
and  
Fast View  
bar

Resource  
Navigator  
view

Message  
area

Text  
editor

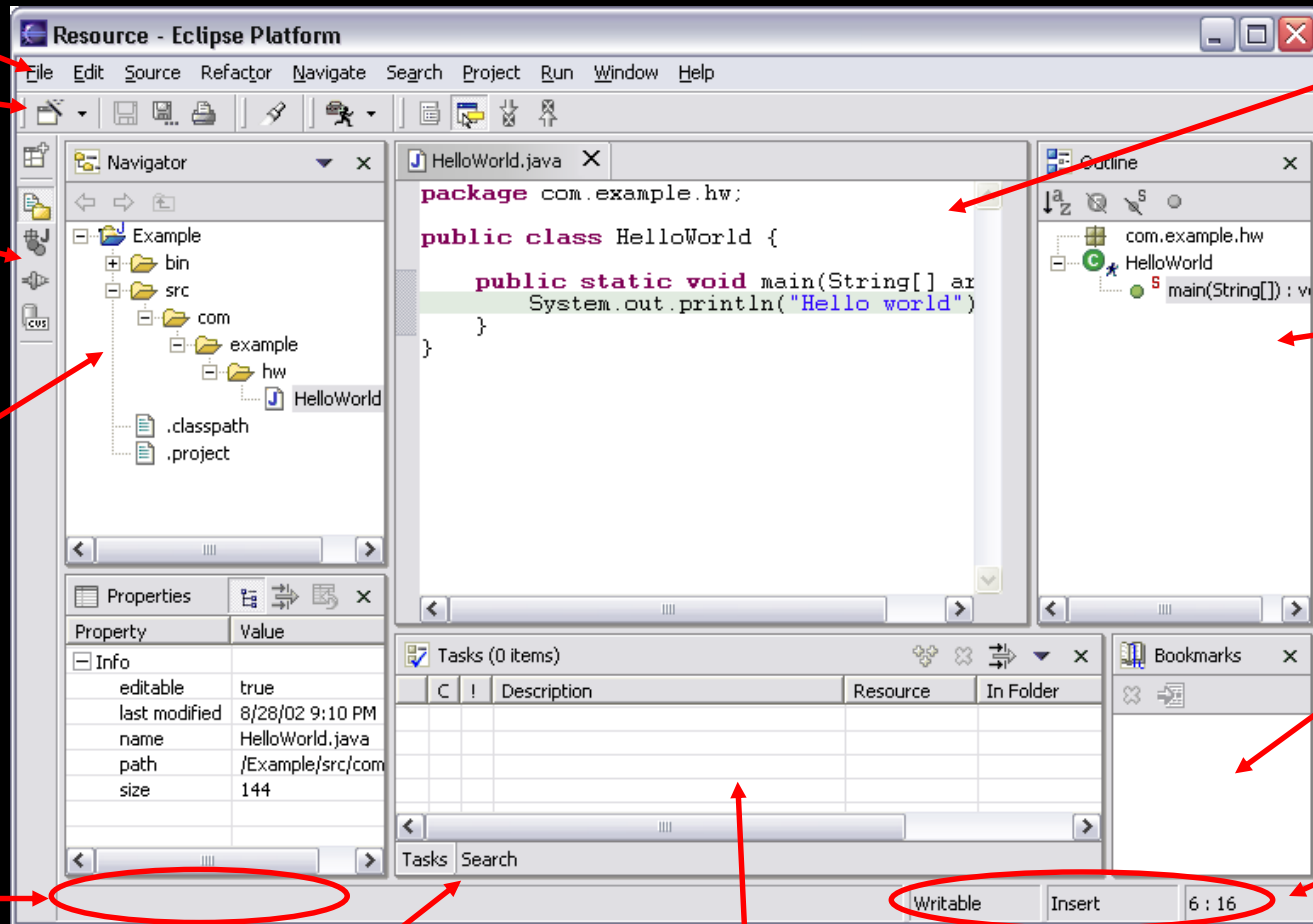
Outline  
view

Bookmarks  
view

Editor  
Status  
area

Stacked  
views

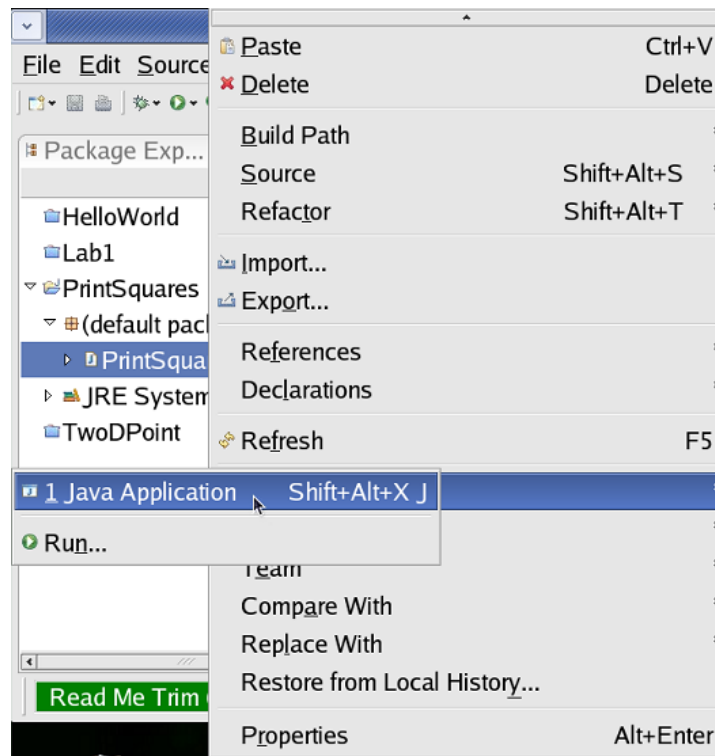
Tasks  
view



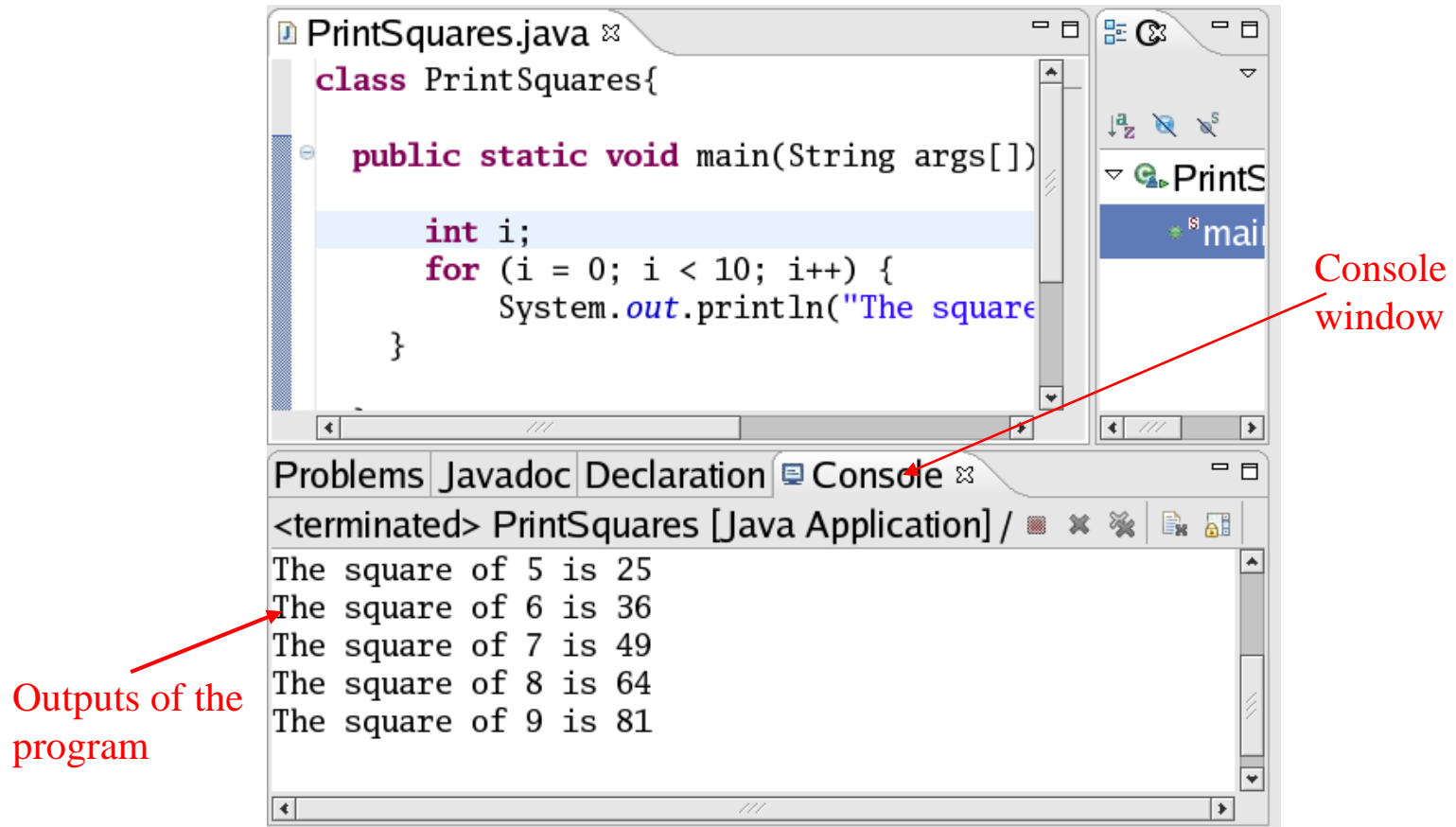
# Run the program(프로그램실행)

오른쪽 단추->run as -> java application선택

1. Right click *PrintSquare.java* in *Workspace* window and select *Run As> Java Application*.

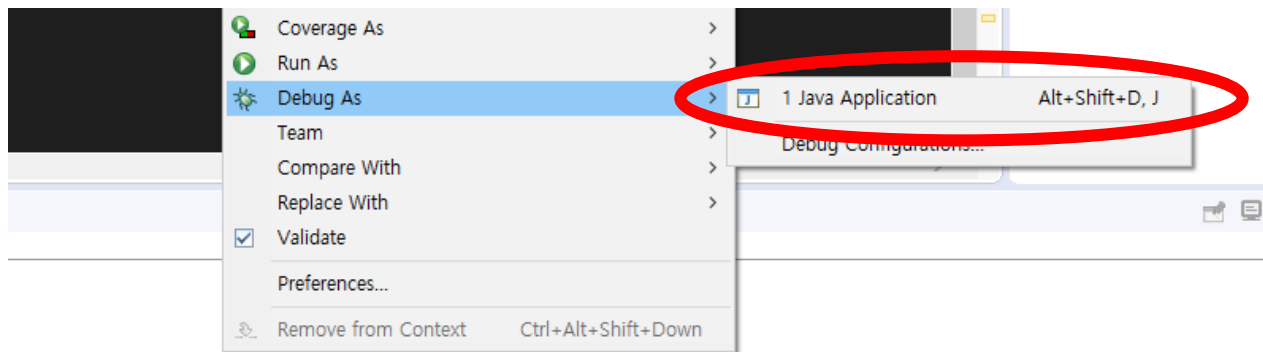


2. *Console* window shows the outputs of the program.



# 실행

- ◆ Ctrl + F11 : 바로 전에 실행했던 클래스 실행
- ◆ 혹은 오른쪽 단추-> Run as선택->Java application선택
- ◆ 가장 많은 자바 에러는 클래스이름과 파일이름이 일치하지 않을 때 발생. 이때 java application이 옵션으로 나타나지 않는다.

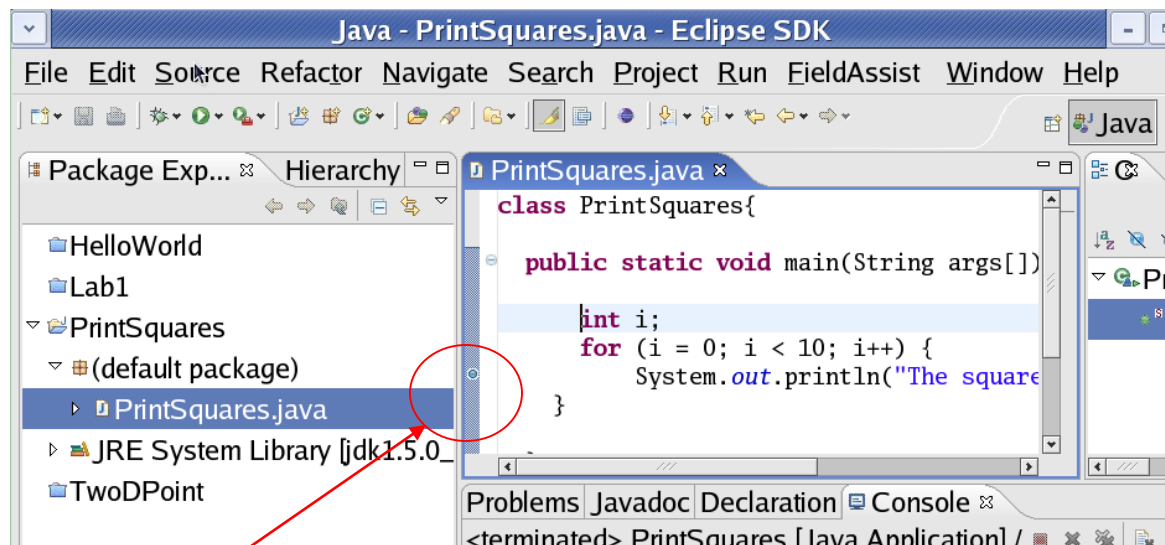


# 한글이 읽혀지지 않을때

- ◆ 한글은 UTF-8형식으로 읽을 수 있다.
- ◆ UTF-8으로 저장되지 않은 파일은 메모장에서 읽어들이고 후 UTF-8형식으로 다시 저장한다.
- ◆ 터미널에서 문제 : MinGW를 사용하여 C, C++컴파일후 실행할 때 한글이 정상출력안되면 폰트(글꼴)문제. 작업창에서
- ◆ >chcp 65001을 입력한다 혹은 C:\Windows\system32\cmd.exe의 속성을 찾아 글꼴을 레스터 글꼴로 바꾸면 된다

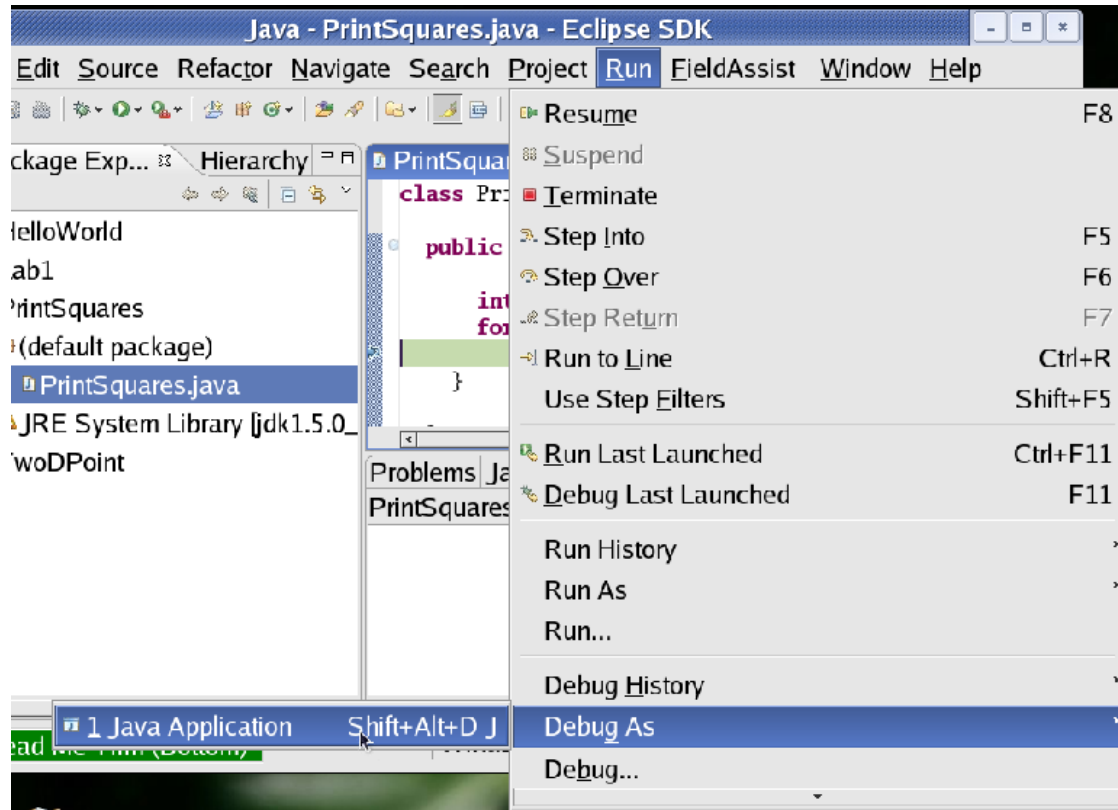
# Debug a Program

1. Add breakpoints: double-click the gray bar on the left of *Edit* window. A blue dot indicates a breakpoint. To remove a breakpoint, double click the breakpoint.

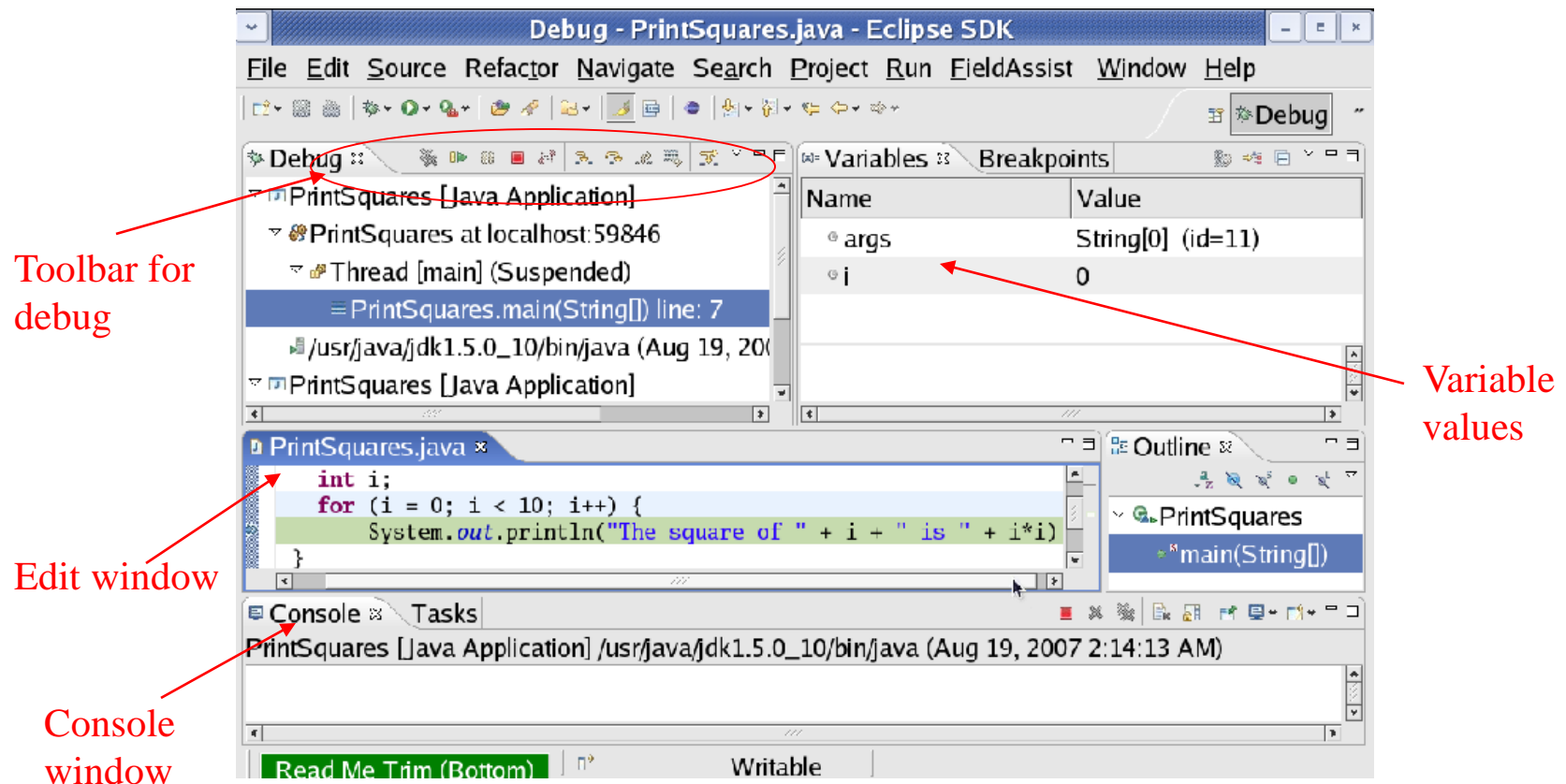


A break point

2. Select **Run->Debug as...->Java Application** to start the debugger.



3. Click **Yes** button in *Confirm Perspective Switch* window to switch Eclipse from *Java Perspective* to *Debug Perspective*.





#### 4. Play with the debug commands and watch the change of variable values in *Variable* window and the outputs in *Console* window.

**Resume** resume the execution of a paused program.

**Suspend** temporarily pause the execution of a program.

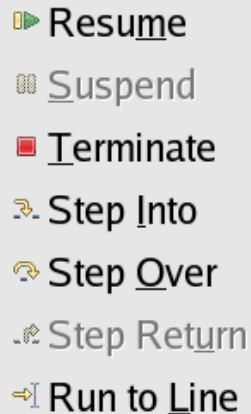
**Terminate** end the current debug session.

**Step Into** execute a single statement or step into a method.

**Step Over** execute a single statement. If the statement contains a call to a method, the entire method is executed without stepping into the method.

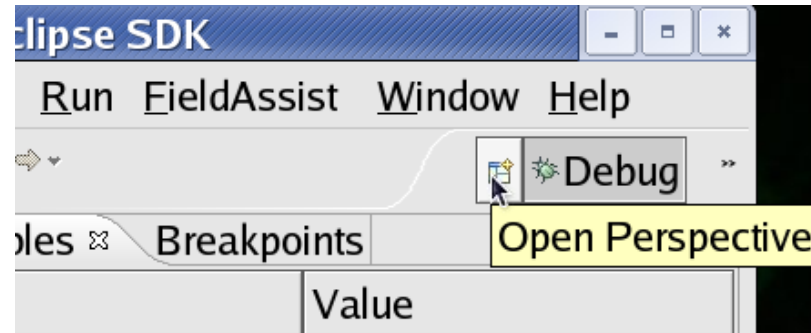
**Step Return** execute all the statements in the current method and returns to its caller.

**Run to Line** runs the program, starting from the current execution point, and pauses at a breakpoint.

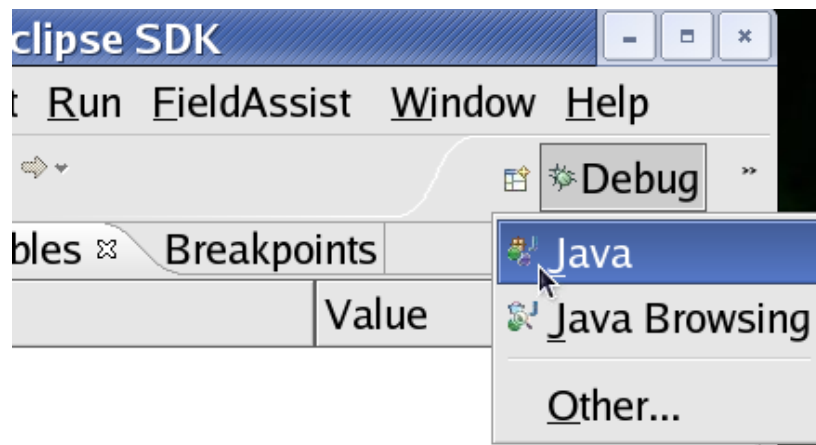


5. Switch Eclipse from *Debug Perspective* back to *Java Perspective*.

- Click *Open Perspective* button.



- Then click *Java*.

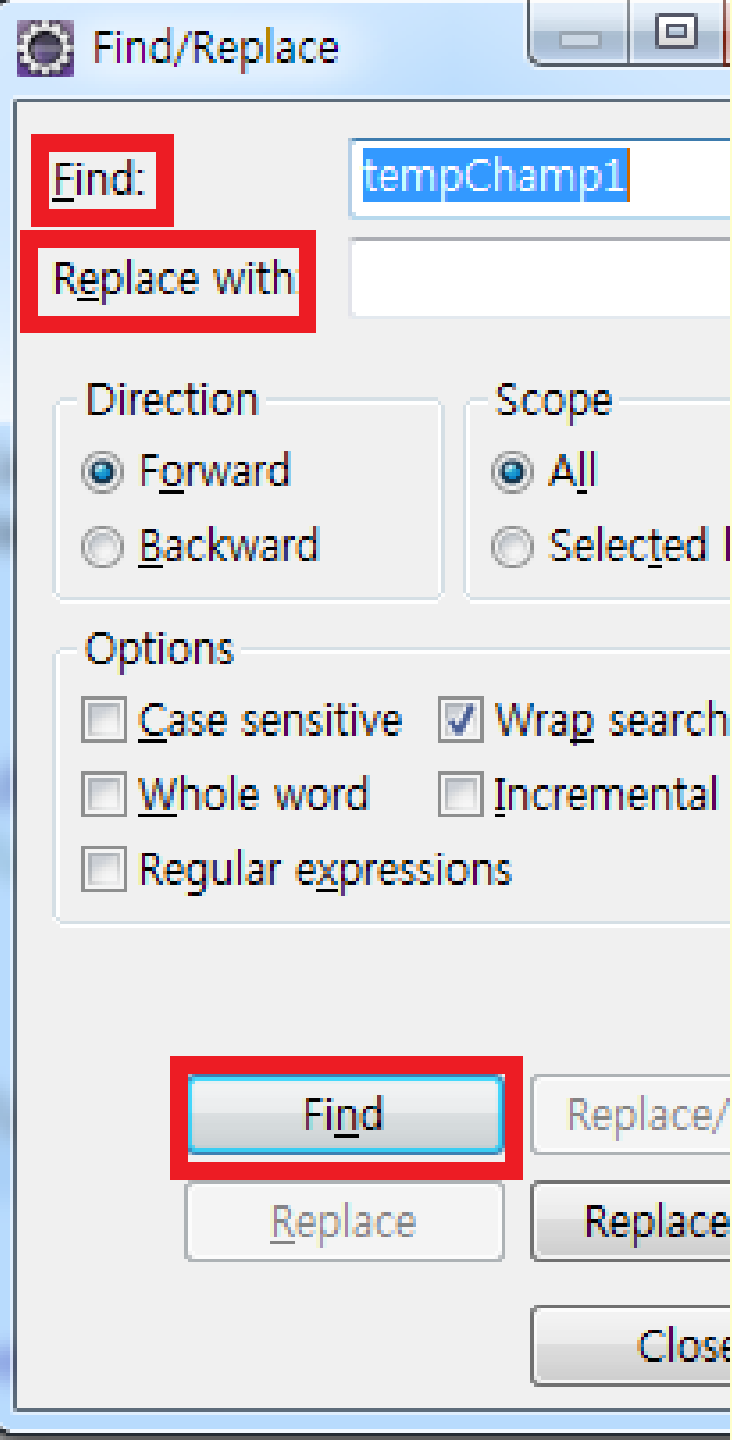


# 소스 네비게이션

- ◆ Ctrl + 마우스커서(혹은 F3) : 클래스나 메소드 혹은 멤버를 상세하게 검색하고자 할 때
- ◆ Alt + ->, Alt + <- : 이후, 이전
- ◆ Ctrl + o : 해당 소스의 메소드 리스트를 확인하려 할 때
- ◆ F4 : 클래스명을 선택하고 누르면 해당 클래스의 Hierarchy 를 볼 수 있다.

# 문자열 찾기

1. Ctrl + k : 찾고자 하는 문자열을 블록으로 설정한 후 키를 계속 누른다.
2. Ctrl + Shift + k : 역으로 찾고자 하는 문자열을 찾아감.
3. Ctrl + j : 입력하면서 찾을 수 있음.
4. Ctrl + Shift + j : 입력하면서 거꾸로 찾아갈 수 있음.
5. Ctrl + f : 기본적으로 찾기
6. Ctrl + shift + g : 커서를 메소드 위에 놓고 누르면 해당 메소드를 사용하는 곳을 찾아준다. 마우스 오른쪽-Reference - 에도 있음.



## 문자열 바꾸기

- 1. Ctrl + k : 찾고자 하는 문자열을 블록으로 설정한 후 키를 누른다.
- 2. 오른쪽 마우스를 클릭하고 Refact를 선택
- 혹은
- Ct기+F 사용하여 문자열을 검색하고  
찾아바꾸기(replace)하면 한번에 문자열 치환

# 원본 편집1

1. Ctrl + Space : 입력 보조장치(Content Assistance) 강제 호출 => 입력하는 도중엔 언제든지 강제 호출 가능하다.
2. F2 : 컴파일 에러의 빨간줄에 커서를 가져다가 이 키를 누르면 에러의 원인에 대한 힌트를 제공한다.
3. Ctrl + l : 원하는 소스 라인으로 이동  
로컬 히스토리 기능을 이용하면 이전에 편집했던 내용으로 변환이 가능하다.
4. Ctrl + Shift + Space : 메소드의 가로안에 커서를 놓고 이 키를 누르면 파라미터 타입 힌트를 볼 수 있다.
5. 한줄 삭제 CTRL + D
6. 파일 닫기 : CTRL+W
7. 들여쓰기 자동 수정. (3.0 NEW) : CTRL+I
8. 블록 주석(/\*..\*/ ) 추가.(3.0 NEW): CTRL+SHIFT+/  
8.1 Ctrl + / 해주면 여러줄이 한꺼번에 주석처리됨. 주석 해제하려면 반대로 하면 됨.
9. 위(아래)줄과 바꾸기 : ALT+UP(DOWN)
10. 블록 선택하기. : ALT+SHIFT+방향키

# 원본 편집2

- 11. 메소드의 파라미터 목록 보기. : CTRL+SHIFT+SPACE
- 12. 자동으로 import 하기 : CTRL+SHIFT+O
- 13. 열린 파일 모두 닫기 : CTRL + SHIFT + F4
- 14. 블록 주석 제거 : CTRL+SHIFT+
- 15. 전체화면 토글 : CTRL+M
- 16. 한줄(블럭) 복사 : Ctrl + Alt + 위(아래)
- 17. 다음 annotation(에러, 워닝, 북마크 가능)으로 점프 : Ctrl + , or .
- 18. 쿼 픽스 : Ctrl + 1
- 19. 메소드 정의부로 이동 : F3
- 20. 하이어라키 팝업 창 띄우기(인터페이스 구현 클래스간 이동시 편리) : Ctrl + T
- 21. 메소드나 필드 이동하기 CTRL + O
- 22. ULTRAEDIT나 EDITPLUS 의 CTRL+TAB 과 같은 기능. : CTRL+F6

# 원본 편집3

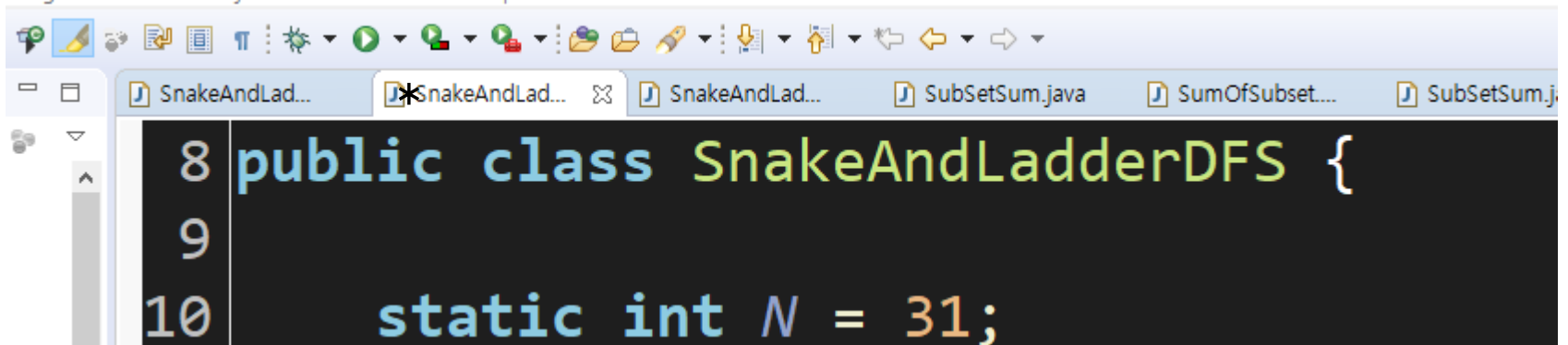
Source -> format 을 선택하면 형식에 맞추어 준다.

클래스 이름 변경 : 클래스 이름위에 오른쪽 마우스 선택후 -> Refact 선택

폰트 변경 : Window-> Preferences->Appearance-> Colors and Fonts->Java->Java Editor Text Font

혹은 Ct rl + (+ / -) 키

파일 탭의 \*파일이름 : 파일이 저장되어 있지 않은 상태.





# 템플릿 사용

1. `sysout` 입력한 후 `Ctrl + Space` 하면 `System.out.println();` 으로 바뀐다.
2. `try` 입력한 후 `Ctrl + Space` 하면 `try-catch` 문이 완성된다.
3. `for` 입력한 후 `Ctrl + Space` 하면 여러가지 `for` 문을 완성할 수 있다.
4. 템플릿을 수정하거나 추가하려면 환경설정/자바/편집기/템플리트 에서 할 수 있다.

# 주석 처리

Ctrl + / 해주면 여러줄이 한꺼번에 주석처리됨. 주석 해제하려면 반대로 하면 됨.

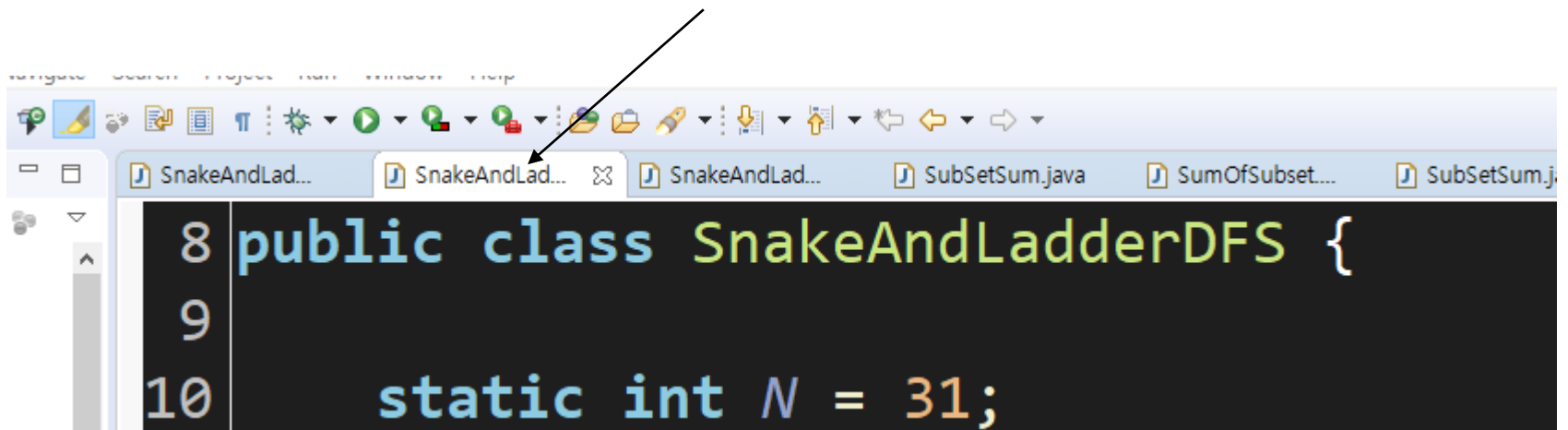
# 이클립스 단축키

Eclipse의 모든 단축키는 **Window >> Preferences >> Workbench >> Keys** 메뉴에서 확인 및 변경이 가능하다. 그 동안 다른 툴에서 자신의 손에 익숙한 단축키들이 있다면 이 메뉴에서 단축키들을 변경하는 것이 가능하다.

# 화면분할

- ◆ 파일이름이 표시된 탭을 드래그하면, 여러 파일을 한 화면에서 볼 수 있다. 즉 화면분할.

탭에 마우스 왼쪽 단추를 누른 상태에서 이리저리 drag해보라



# 이클립스 레이아웃 초기화

화면의 구성이 복잡해져 초기화하려면

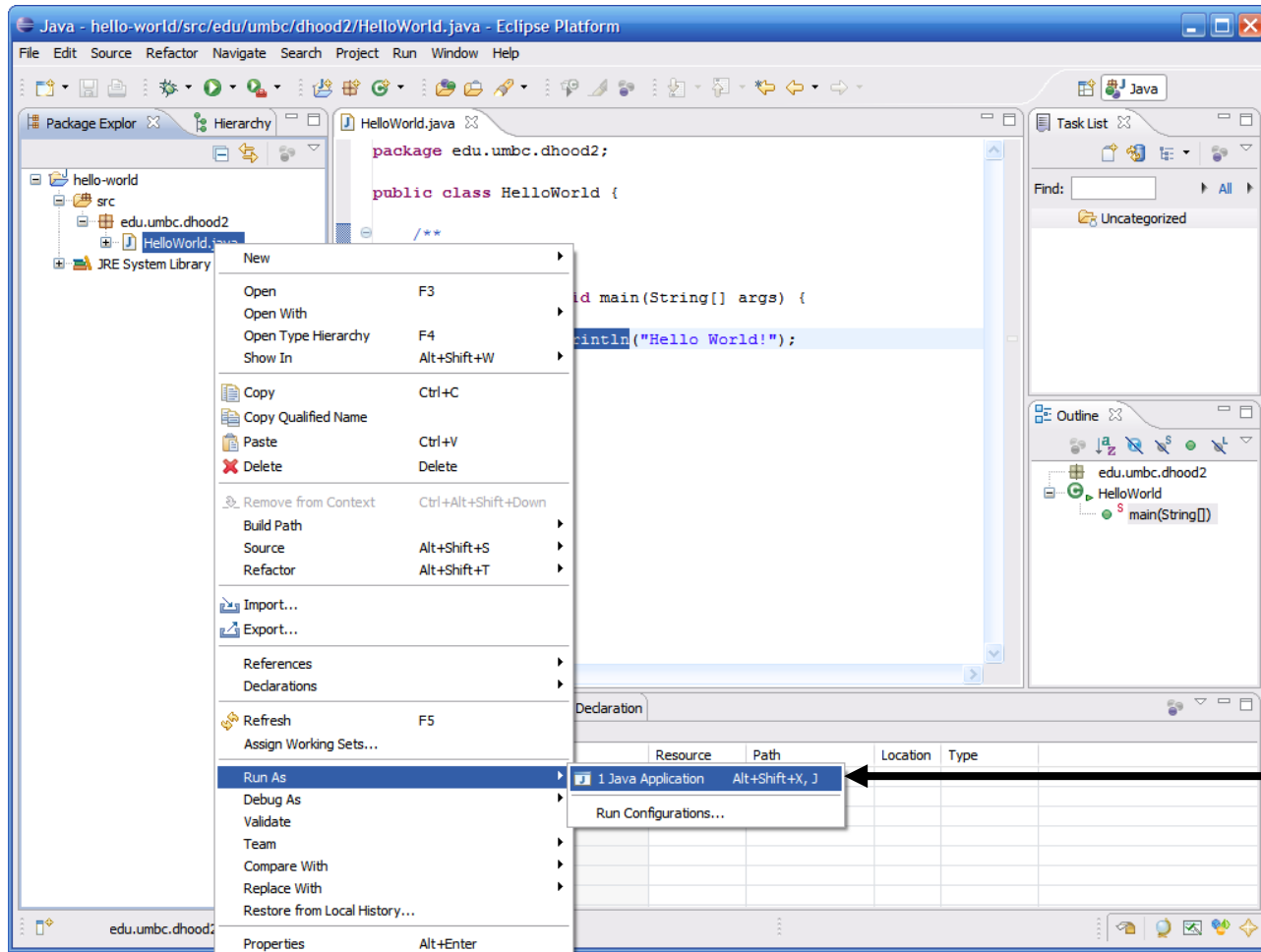
Eclipse의 화면을 초기화하려면 **Window >> Perspective >> Reset Perspective**

# 이클립스 테마변경 방법

Eclipse의 화면을 초기화하려면 **Window >> Preferences >> Appearance >> Theme**  
원하는 테마를 선택하고, Apply를 선택, OK 클릭

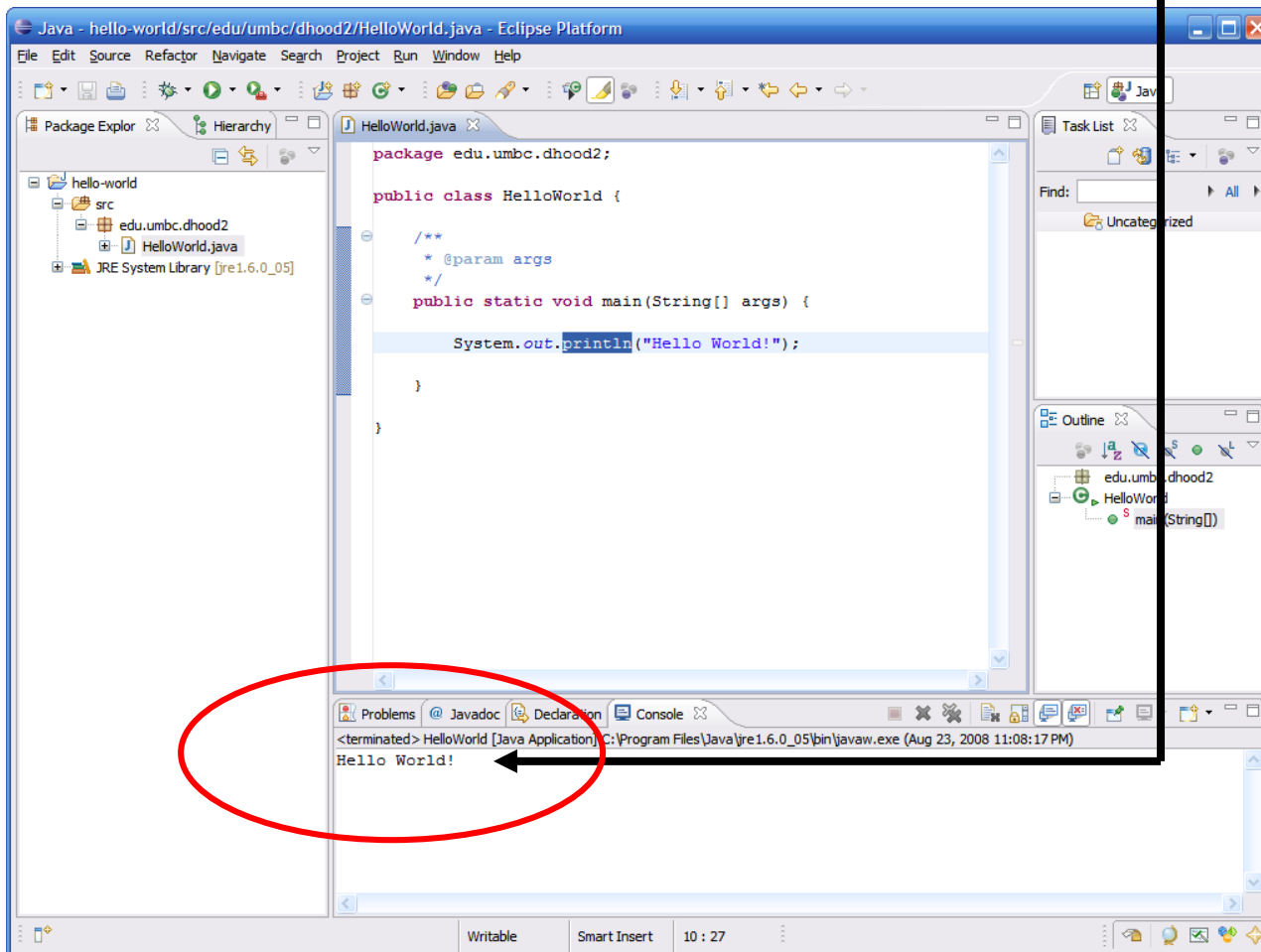
# Running Code(실행)

- ◆ An easy way to run code is to **right click** on the class and select **Run As → Java Application**



# Running Code (continued)

- ◆ The output of running the code can be seen in the Console tab in the bottom pane

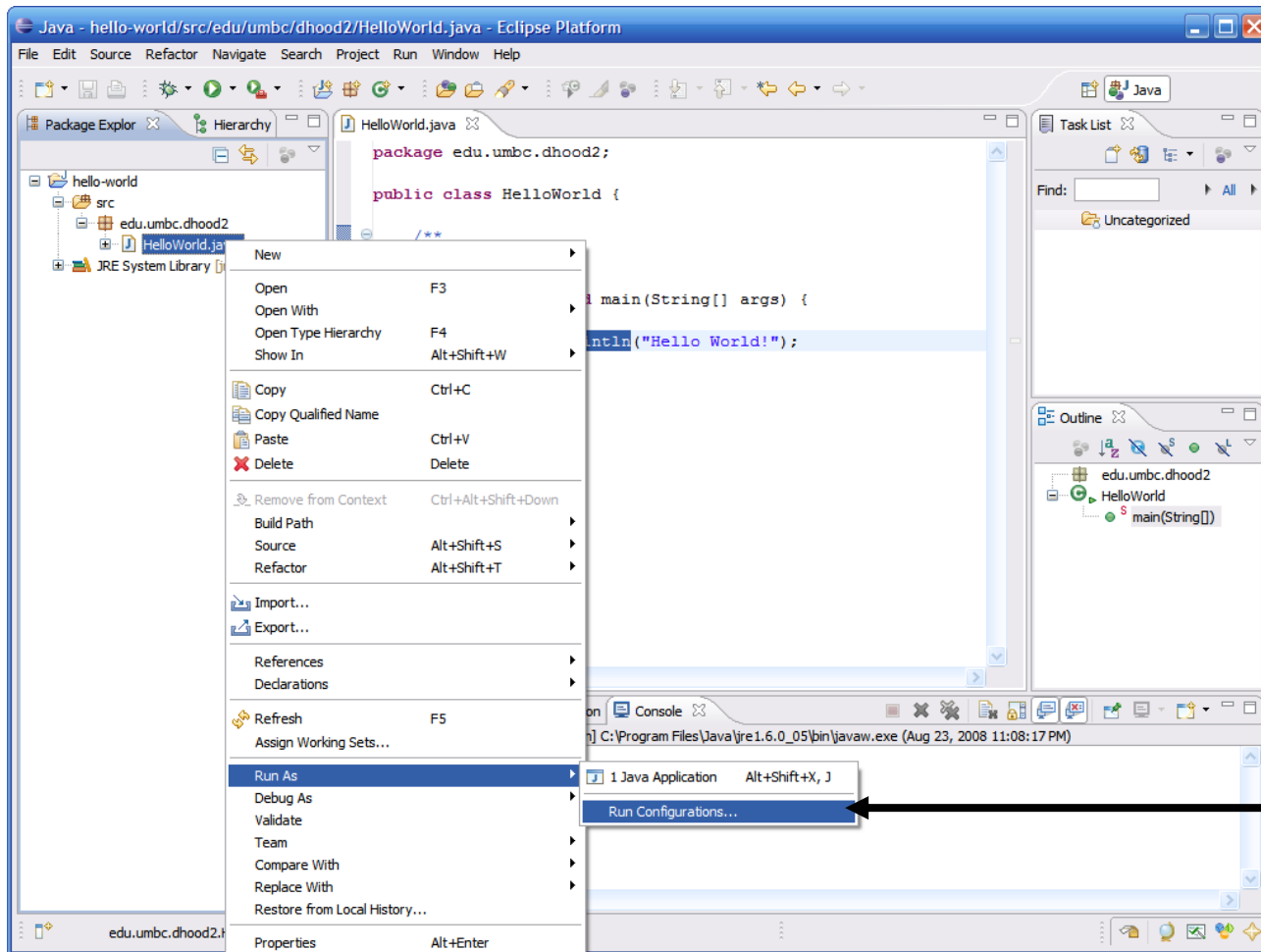




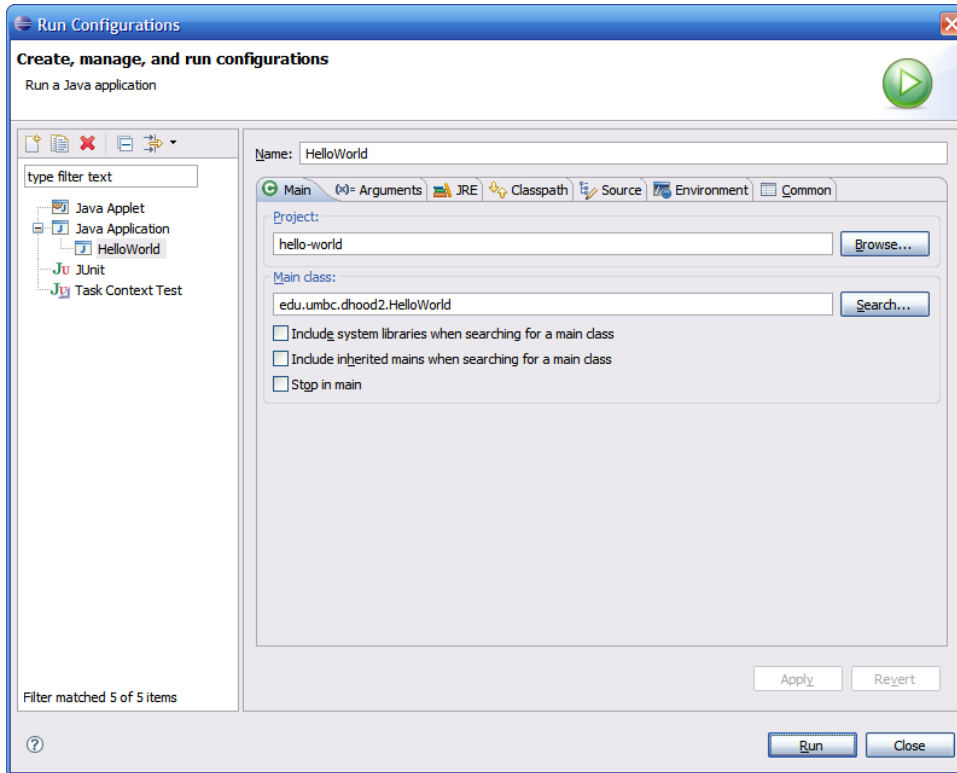
# Run Configuration

## 여러 옵션 조정 가능

- ◆ Advanced options for executing a program can be found by right clicking the class then clicking Run As → Run...



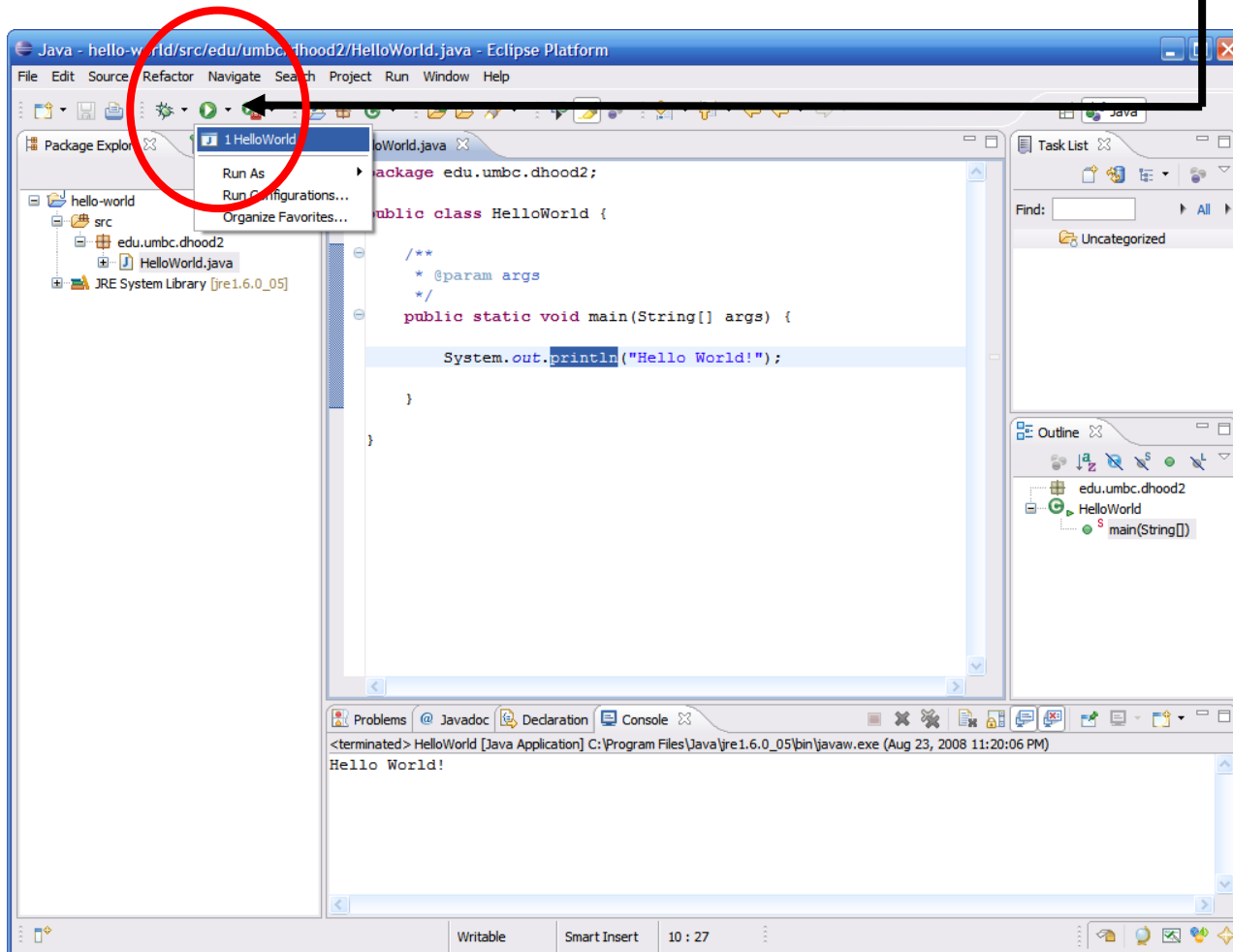
# Run Configuration (continued)



- ◆ Here you can change/add any of the following:
  - ◆ JVM arguments
  - ◆ Command line arguments
  - ◆ Classpath settings
  - ◆ Environment variables
  - ◆ Which JVM to use

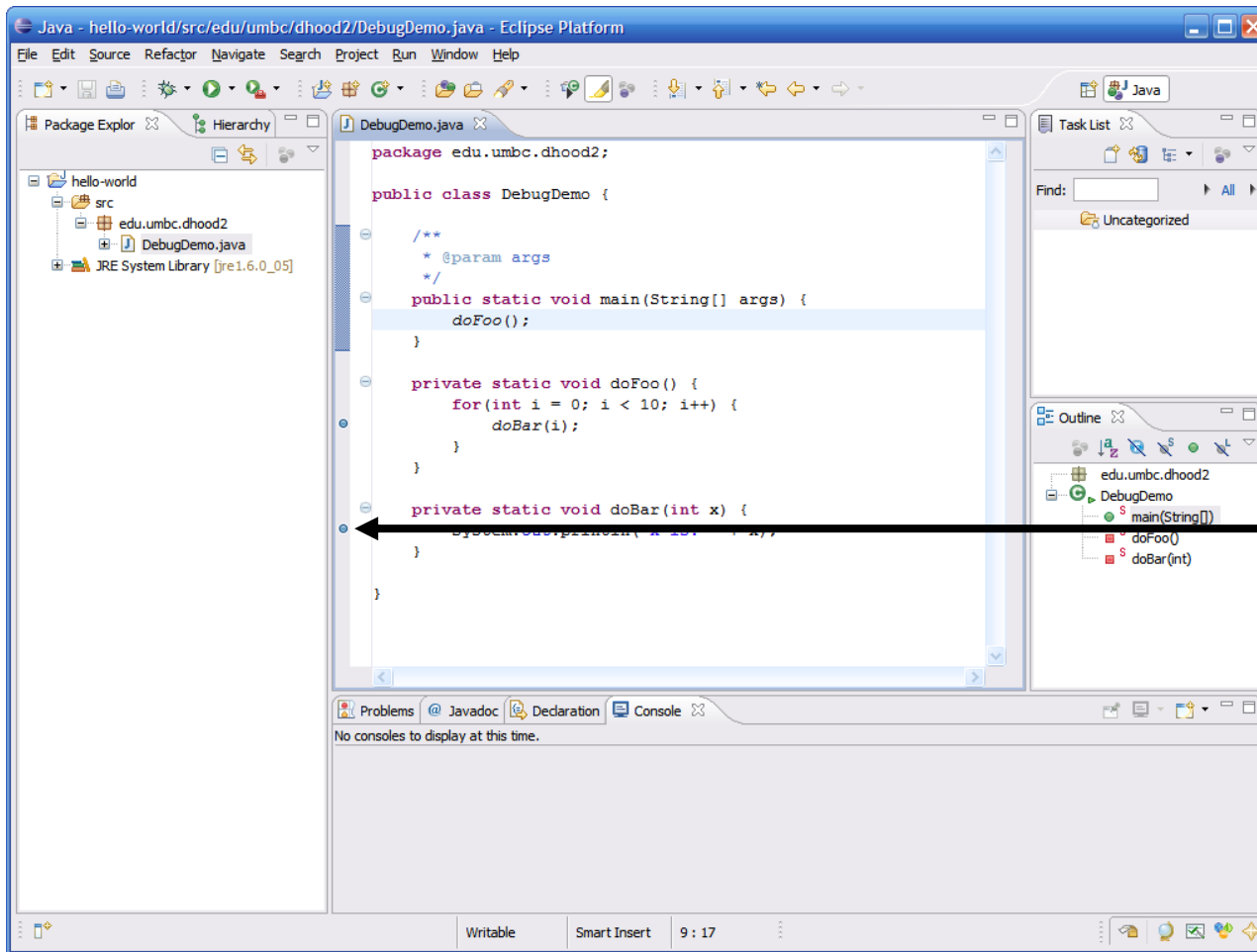
# Re-Running Code(다시실행)

- ◆ After you run the code a first time, you can re-run it just by selecting it from the run drop down menu



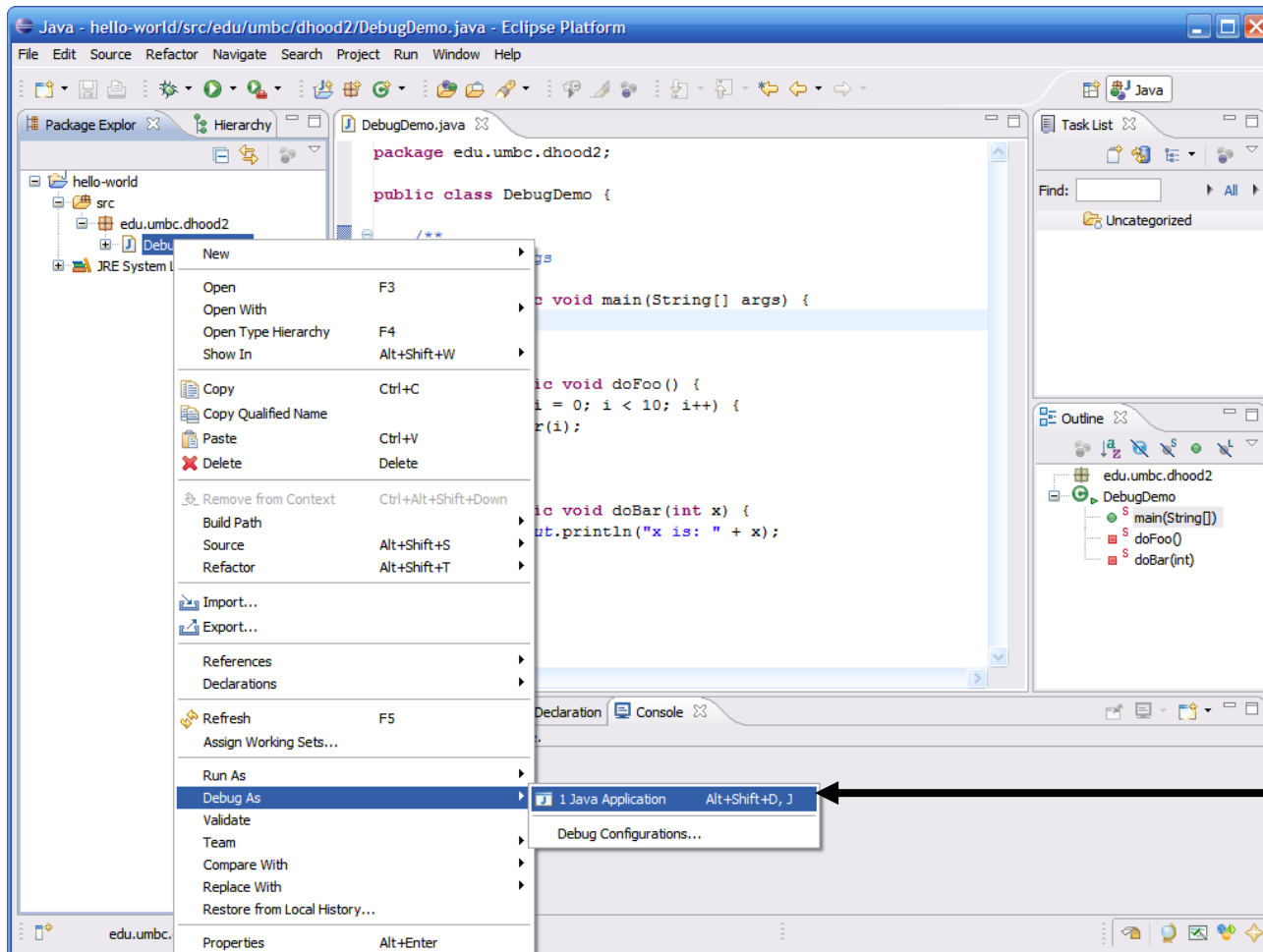
# Debugging Code(디버그)

- ◆ Eclipse comes with a pretty good built-in debugger
- ◆ You can set break points in your code by double clicking in the left hand margin – break points are represented by these blue bubbles



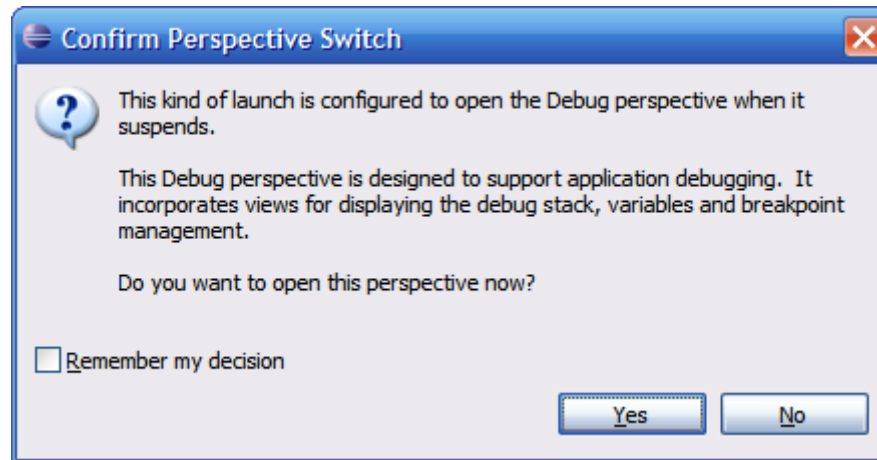
# Debugging Code (continued)

- ◆ An easy way to enter debug mode is to right click on the class and select **Debug As → Java Application**



# Debugging Code (Continued)

- ◆ The first time you try to debug code you will be presented with the following dialog



- ◆ Eclipse is asking if you want to switch to a perspective that is more suited for debugging, click Yes
- ◆ Eclipse has many perspectives based on what you are doing (by default we get the Java perspective)

# Debug Perspective

Note new Debug perspective  
– click Java to return to  
normal

These buttons allow you to  
step through the code

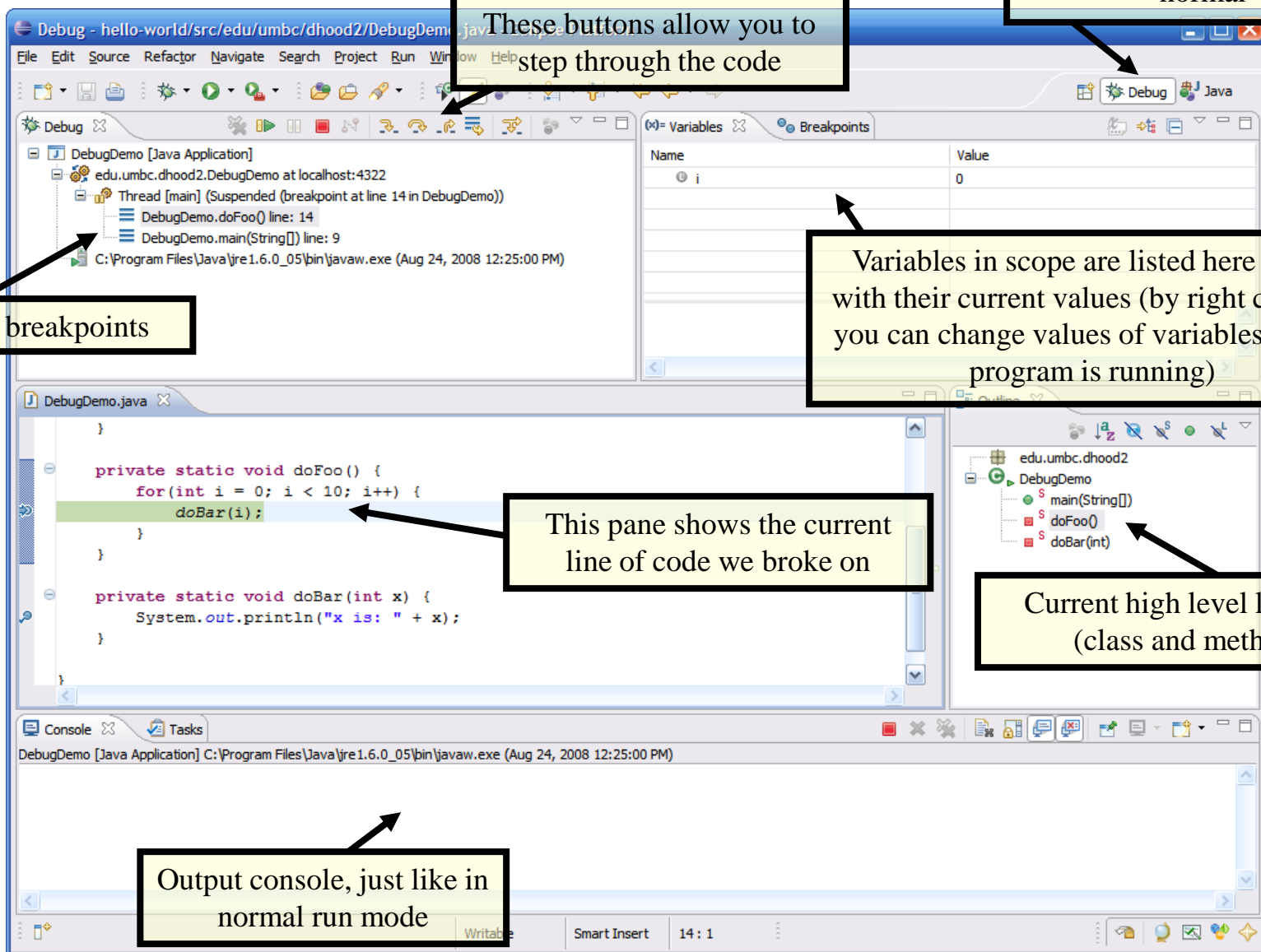
List of breakpoints

Variables in scope are listed here along  
with their current values (by right clicking  
you can change values of variables as you  
program is running)

This pane shows the current  
line of code we broke on

Current high level location  
(class and method)

Output console, just like in  
normal run mode



# Puppy(귀여운 강아지)





# 능름한 허스키



Innocent face



brave husky





감사합니다.

