



Introduction to Eclipse and Java Programming

Mehran Najafi

TA for SE3KO4 / SE3MO4

Software Development for Computer and
Electrical Engineering

najafm@mcmaster.ca



ECLIPSE



What is Eclipse?

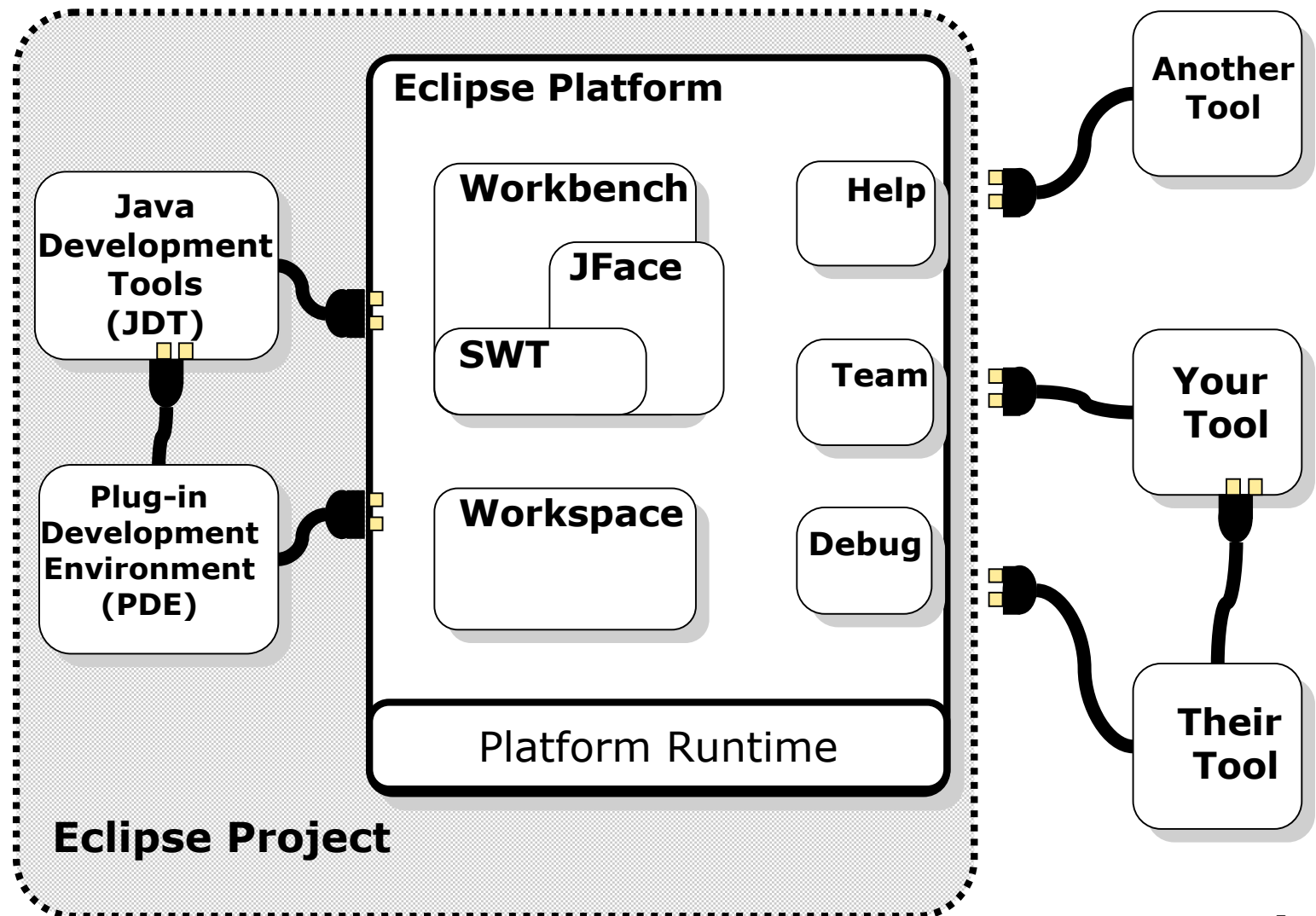
- **Eclipse** is an open-source software framework written primarily in Java.
- In its default form it is a Java IDE, consisting of
 - the *Java Development Tools* (JDT)
 - compiler (ECJ).

Users can extend its capabilities by installing plug-ins written for the Eclipse software framework, such as development toolkits for other programming languages, and can write and contribute their own plug-in modules.
- Eclipse can to be extended using other programming languages such as C and Python.

Eclipse Structure

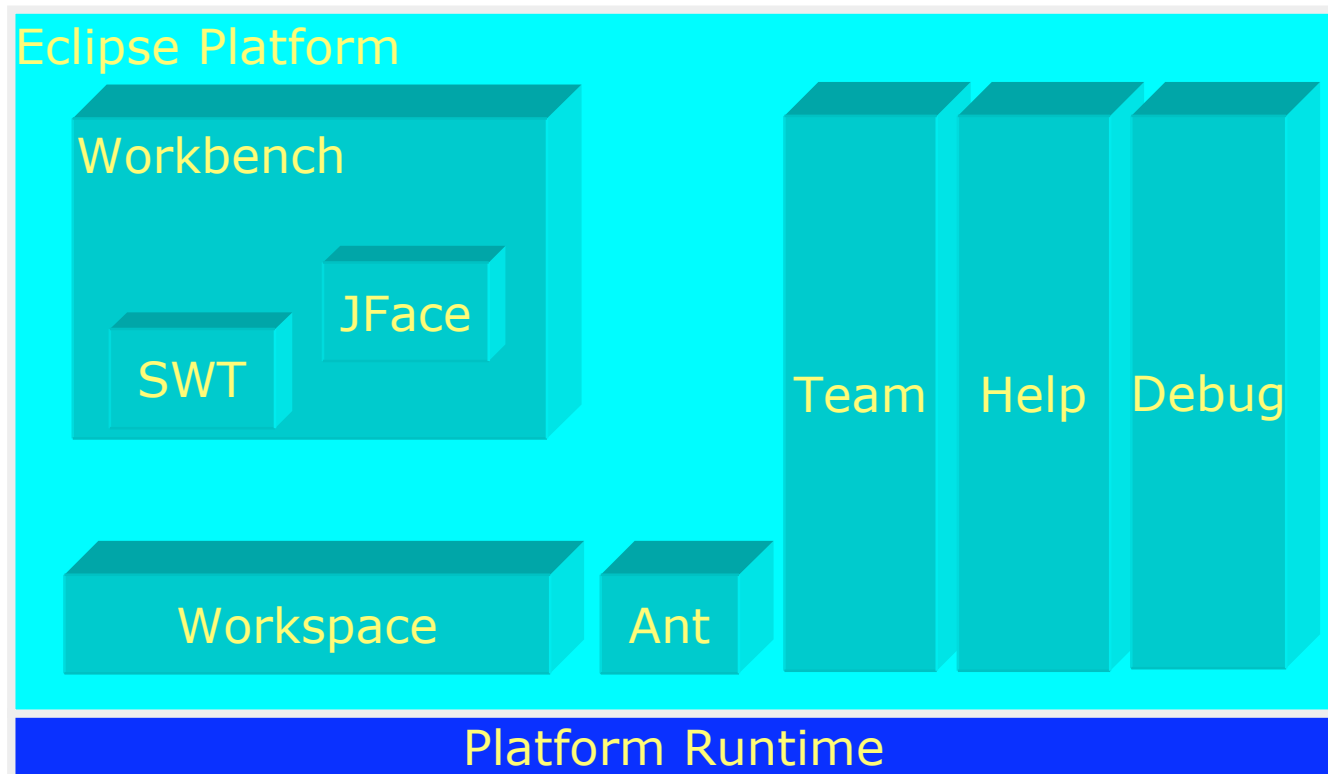
- The following components constitute the rich client platform:
 - Core platform - boot Eclipse, run plug-ins
 - OSGi - a standard bundling framework
 - the Standard Widget Toolkit (SWT) - a portable widget toolkit
 - JFace - file buffers, text handling, text editors
 - The Eclipse Workbench - views, editors, perspectives, wizards

Eclipse Overview



Eclipse Platform

- Eclipse Platform is the common base
- Consists of several key components





Java Development Tools

- It offers an IDE with a built-in incremental Java compiler and a full model of the Java source files. This allows for advanced refactoring techniques and code analysis.



Eclipse Vs Netbeans

- The advantages of eclipse are:
ease of use
speed
flexibility : import project, librairies,
plugins
SWT
Refactoring

Refactoring

- Eclipse provides a powerful set of automated refactorings that, among other things, let you rename Java™ elements, move classes and packages, create interfaces from concrete classes, turn nested classes into top-level classes, and extract a new method from sections of code in an old method



Plug-in

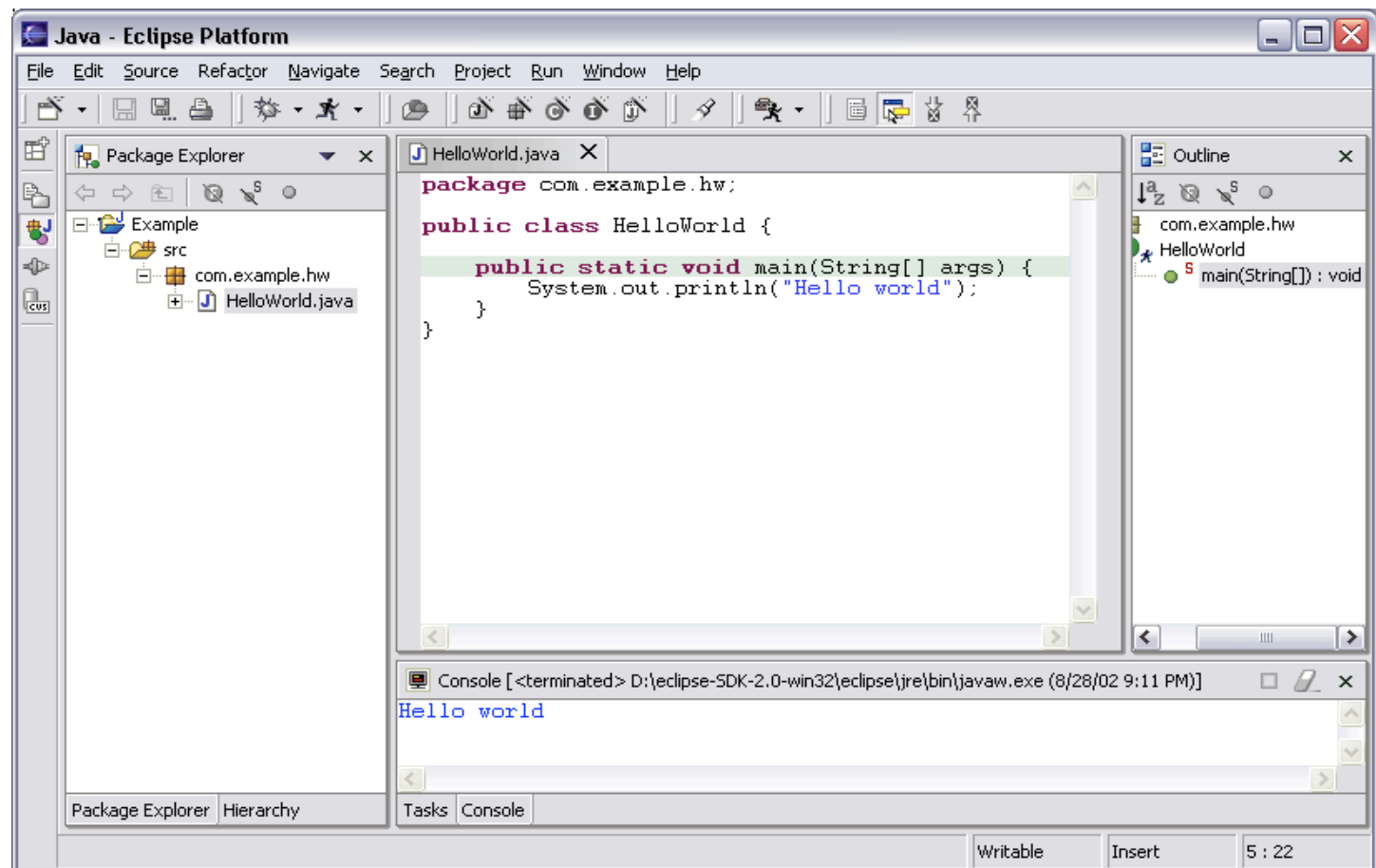
- A plugin is a computer program that interacts with a host application (a web browser or an email client, for example) to provide a certain, usually very specific, function "on demand".
- Applications support plugins for many reasons.
 - a) enabling developers to create capabilities to extend an application
 - b) reducing the size of an application
 - c) separating source code from the an application

SWT

- **SWT = Standard Widget Toolkit**
- The Standard Widget Toolkit (SWT) is a graphical widget toolkit for the Java platform originally developed by IBM and maintained now by the Eclipse Foundation in tandem with the Eclipse IDE.
- SWT is written in Java. To display GUI elements, the SWT implementation accesses the native GUI libraries of the operating system
- Simple
Small
Fast
OS-independent API
Uses native widgets where available
Emulates widgets where unavailable

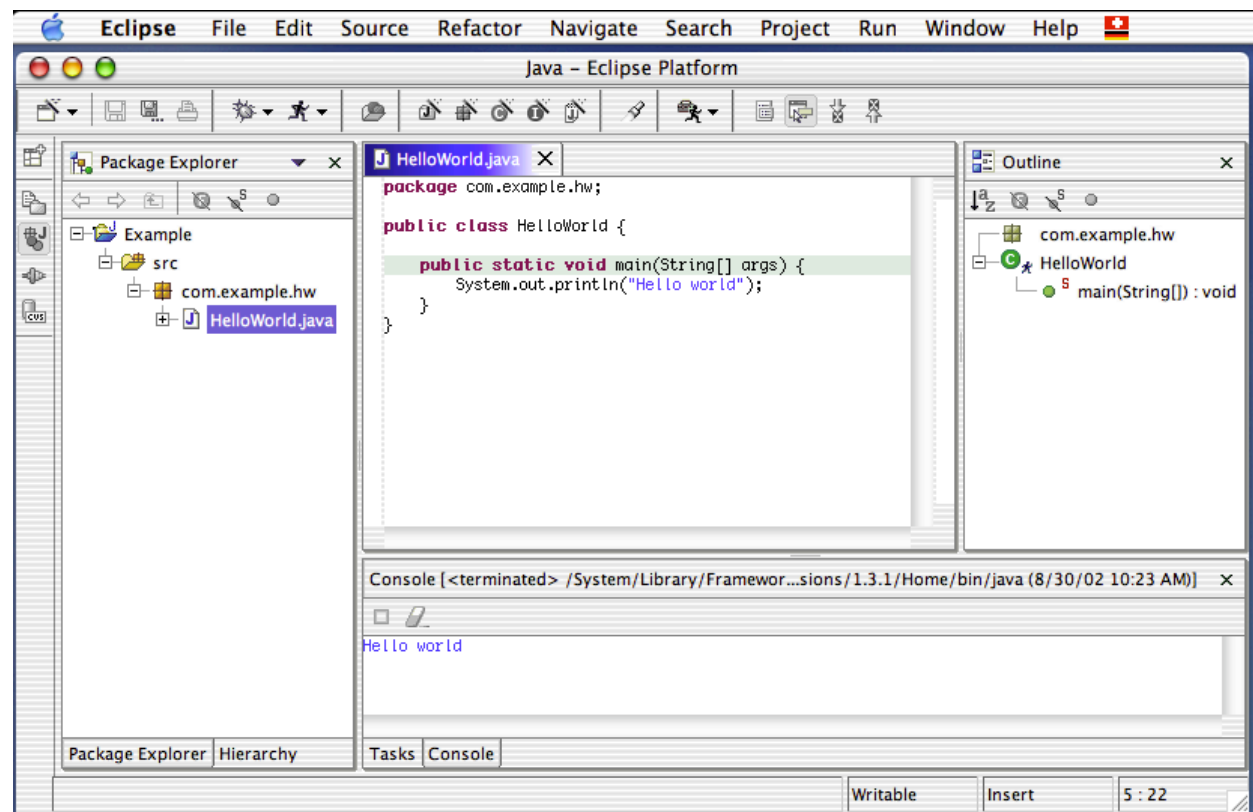
Why SWT?

Eclipse Platform on Windows XP



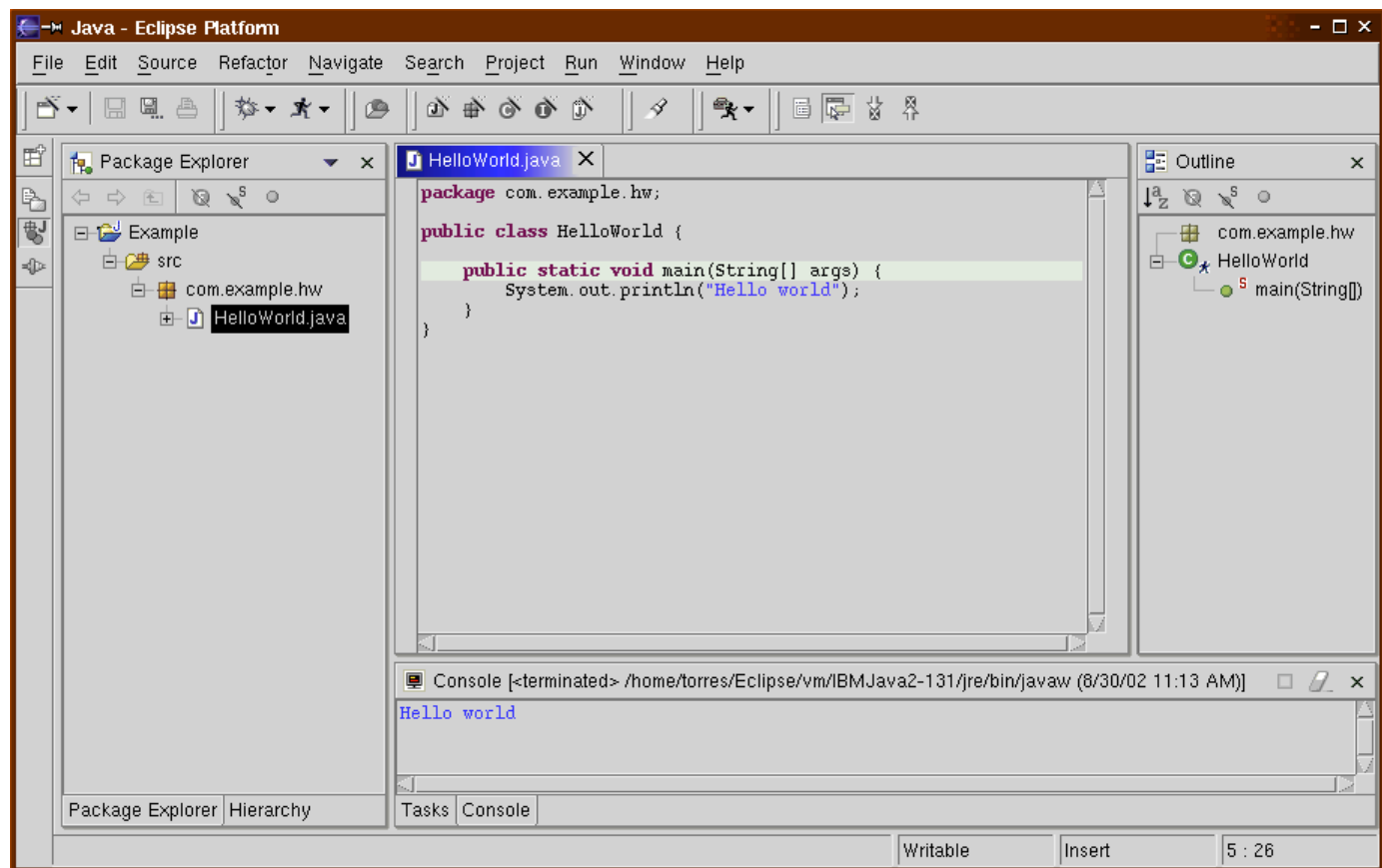
Why SWT?

Eclipse Platform on Mac OS X - Carbon



Why SWT?

Eclipse Platform on Linux - Motif



SWT Vs AWT, Swing

AWT (the Abstract Windowing Toolkit) was the first Java GUI toolkit.

- Use operating system-supplied objects to create GUI elements, such as menus, windows and buttons.
- AWT was a very thin wrapper around native widgets
- AWT was OS dependent

Swing was the next generation toolkit introduced by Sun

- Swing GUI elements are 100% Java, with no native code — instead of wrapping around native APIs, Swing calls low level operating system routines to draw the GUI elements by itself.

SWT is a wrapper around native code objects



Who's Shipping on Eclipse?

Commercial products

10 Technology – Visual PAD
Assisi – V4ALL Assisi GUI-Builder
Bocaloco – XMLBuddy
Borland – Together Edition for WebSphere Studio
Catalyst Systems – Openmake
Computer Associates – AllFusion Harvest Change Manager VCM
Ensemble Systems – Glider for Eclipse
Fujitsu – Interstage
Genuitec – EASIE Plug-ins
HP – OpenCall Media Platform OClet Development Environment
James Holmes – Struts Console
Instantiations – CodePro Studio



Installation

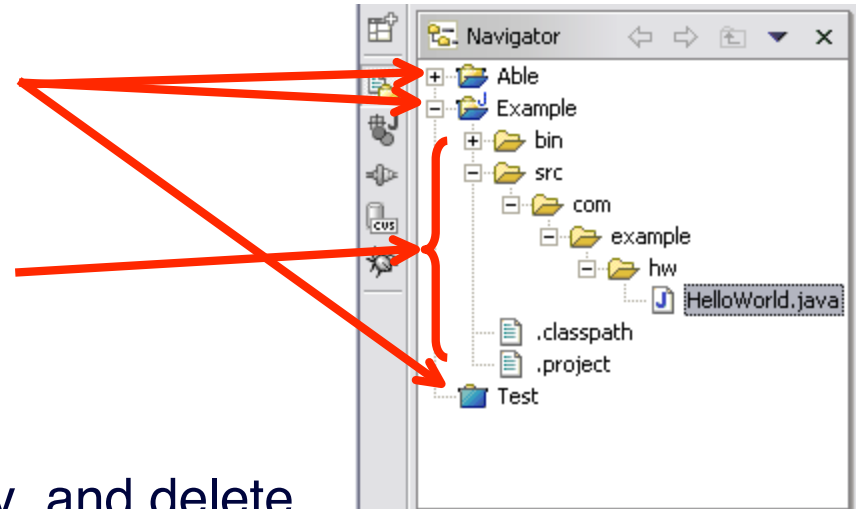
- 1) You will need a **Java Runtime Environment** (JRE) to run Eclipse IDE for Java Developers. A Java 5 JRE is recommended.
(http://java.sun.com/javase/downloads/index_jdk5.jsp)
- 2) Download the Eclipse IDE for Java Developers.
(<http://www.eclipse.org/downloads/>)
- 3) Use the utility for your platform to decompress the downloaded file onto your file system

The Java SE Development Kit (JDK) includes:

- the Java Runtime Environment (JRE)
- command-line development tools, such as compilers and debuggers, that are necessary or useful for developing applets and applications

Workspace Component

- Tools operate on files in user's **workspace**
- Workspace holds 1 or more top-level **projects**
- Projects map to directories in file system
- Tree of **folders** and **files**
- {Files, Folders, Projects} termed **resources**
- Tools read, create, modify, and delete resources in workspace
- Plug-ins access via workspace and resource APIs



Workbench Terminology

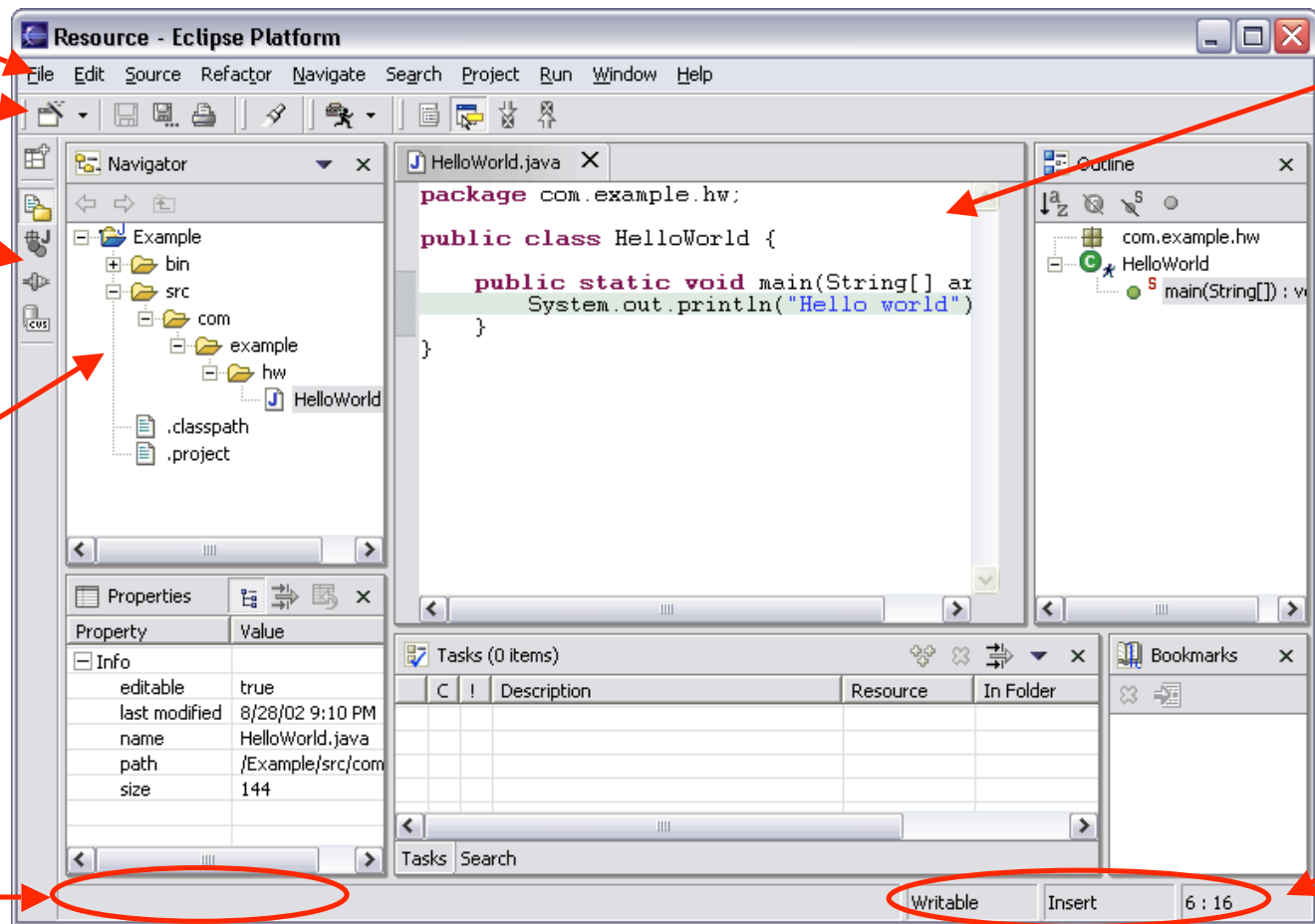
Menu bar

Tool bar

Perspective
and
Fast View
bar

Resource
Navigator
view

Message
area

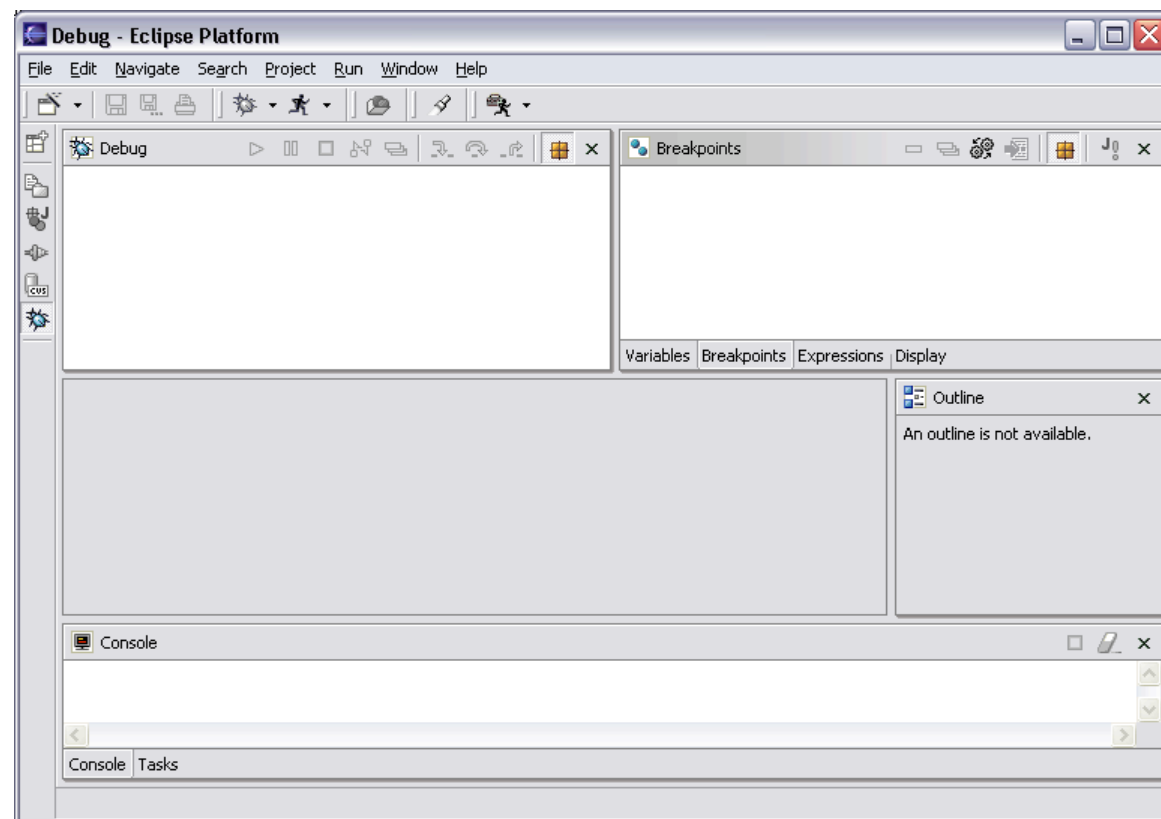


Text
editor

Editor
Status
area

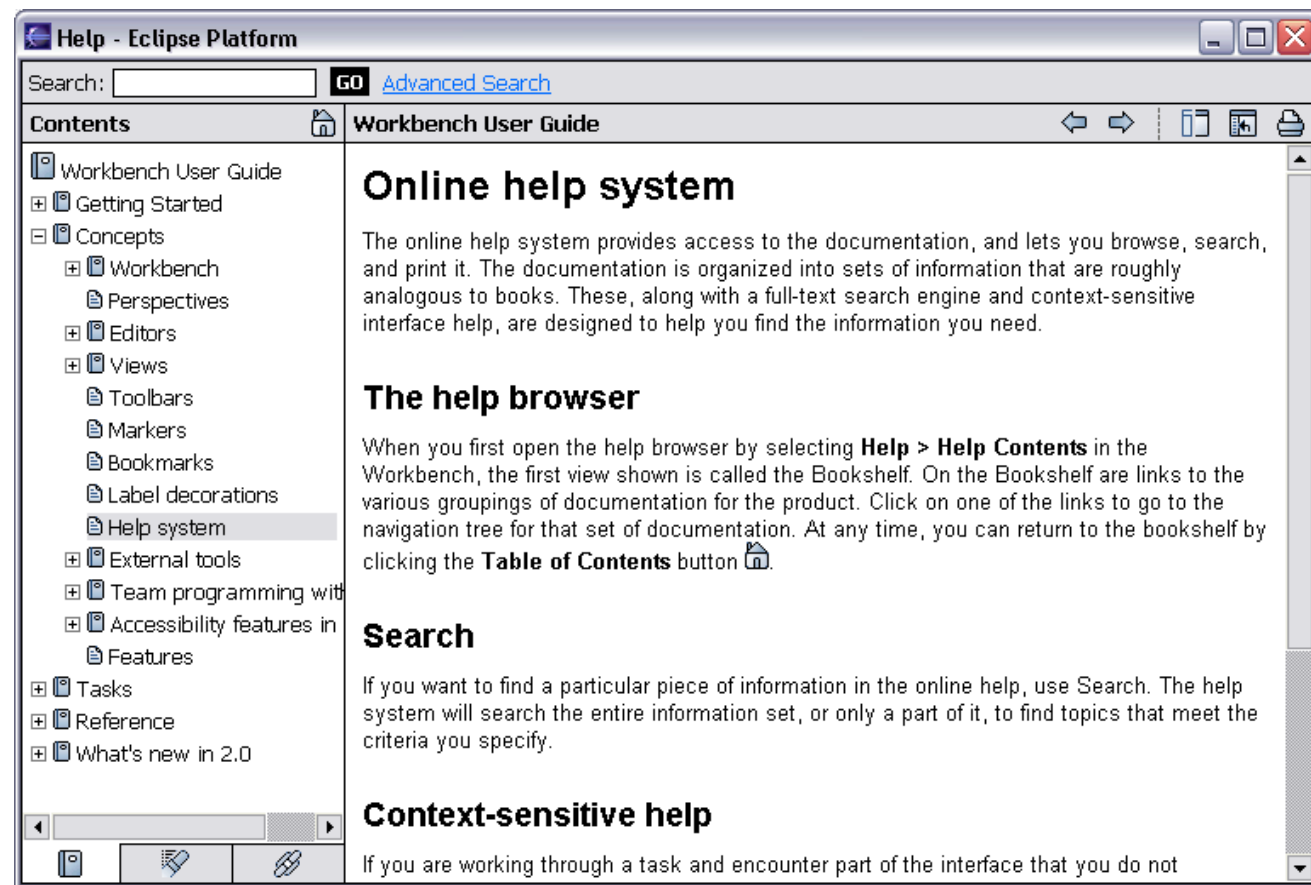
Debug Component

Common debug UI and underlying debug model



Help Component

Help is presented in a standard web browser



Eclipse Java Debugger

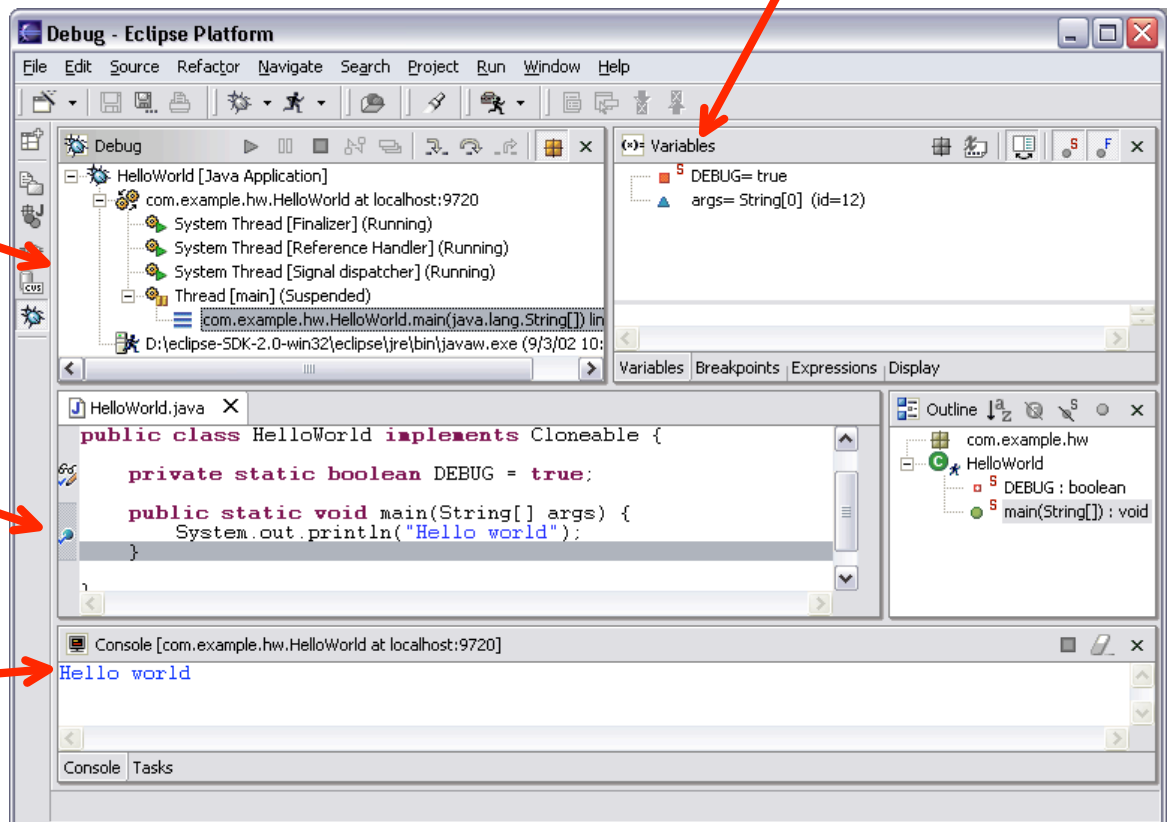
Run or debug Java programs

Local variables

Threads
and stack
frames

Editor with
breakpoint
marks

Console
I/O





Create a program

1) Create the workspace and project

- a) File -> New -> Project -> Java Project
- b) Specify the Project Name
- c) Press finish

2) Add a Class to the project

- a) Press right click on the project name
- b) New -> Class
- c) Specify the class name

3) Run the program

- a) Run -> Run as -> Java Application



Debug

- 1) Run-> Debug
- 2) Set Breakpoint(s): Right Click on the left side of the desired line(s).
- 3) Use Step Info and Step Over for controlling the running sequence
- 4) Use Terminate button for finishing the debug mode



JAVA

BASICS



Introduction to java

- Java
 - Powerful, object-oriented language
 - Fun to use for beginners, appropriate for experience programmers
 - Language of choice for Internet and network communications
- Java Systems
 - Consist of environment, language, Java Applications Programming Interface (API), Class libraries
- Java programs have five phases
 - **Edit**
 - Use an editor to type Java program
 - **vi** or **emacs**, notepad, Jbuilder, Visual J++
 - **.java** extension
 - **Compile**
 - Translates program into bytecodes, understood by Java interpreter
 - **javac** command: **javac myProgram.java**
 - Creates **.class** file, containing bytecodes (**myProgram.class**)



Introduction to java

- Java programs have five phases (continued)
 - **Loading**
 - Class loader transfers **.class** file into memory
 - Applications - run on user's machine
 - Applets - loaded into Web browser, temporary
 - Classes loaded and executed by interpreter with **java** command
 - java Welcome**
 - HTML documents can refer to Java Applets, which are loaded into web browsers. To load,
 - appletviewer Welcome.html**
 - **appletviewer** is a minimal browser, can only interpret applets



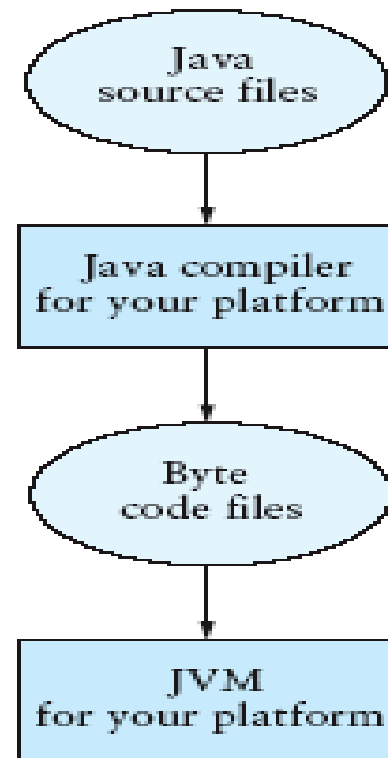
Introduction to java

- Java programs have five phases (continued)
 - **Verify**
 - Bytecode verifier makes sure bytecodes are valid and do not violate security
 - Java must be secure - Java programs transferred over networks, possible to damage files (viruses)
 - **Execute**
 - Computer (controlled by CPU) interprets program one bytecode at a time
 - Performs actions specified in program

Compiling and Executing a Java Program

FIGURE A.1

Compiling and Executing a Java Program



References and Primitive Data Types

- Java distinguishes two kinds of entities
 - Primitive types
 - Objects
- Primitive-type data is stored in primitive-type variables
- Reference variables store the *address of* an object

Primitive type	Range of values
byte	-128 .. 127 (8 bits)
short	-32,768 .. 32,767 (16 bits)
int	-2,147,483,648 .. 2,147,483,647 (32 bits)
long	-9,223,372,036,854,775,808 (64 bits)
float	$\pm 10^{-38}$ to $\pm 10^{+38}$ and 0, about 6 digits precision
double	$\pm 10^{-308}$ to $\pm 10^{+308}$ and 0, about 15 digits precision
char	Unicode characters (generally 16 bits per char)
boolean	True or false

Operators

- 1) subscript [], call (), member access .
- 2) pre/post-increment ++ --, boolean complement !, bitwise complement ~, unary + -, type cast (**type**), object creation **new**
- 3) * / %
- 4) binary + - (+ also concatenates strings)
- 5) signed shift << >>, unsigned shift >>>
- 6) comparison < <= > >=
- 7) equality comparison == !=
- 8) bitwise and &
- 9) bitwise or |
- 10) logical (sequential) and &&
- 11) logical (sequential) or ||
- 12) conditional **cond ? true-expr : false-expr**
- 13) assignment =
- 14) compound assignment += -= *= /= <<= >>= >>>= &= |=



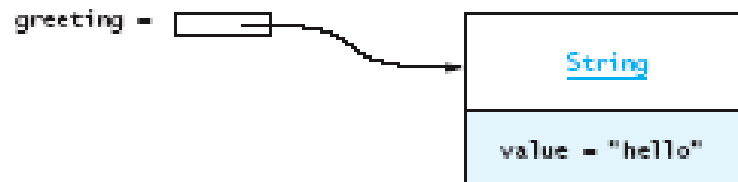
Declaring and Setting Variables

- `int square;`
`square = n * n;`
- `double cube = n * (double)square;`
 - Can generally declare local variables where they are initialized
 - All variables get a safe initial value anyway (zero/null)

Referencing and Creating Objects

- You can **declare reference variables**
 - They reference objects of **specified types**
- Two reference variables can reference **the same object**
- The **new** operator creates an instance of a class
Class_name variable_name=new class_name();
- A **constructor** executes when a new object is created

FIGURE A.2
Variable greeting
References a String
Object



Java Control Statements

- A group of statements executed in order is written
 - `{ stmt1; stmt2; ...; stmtN; }`
- The statements execute in the order 1, 2, ..., N
- Control statements alter this sequential flow of execution

TABLE A.4
Java Control Statements

Control Structure	Purpose	Syntax
if ... else	Used to write a decision with <i>conditions</i> that select the alternative to be executed. Executes the first (second) alternative if the <i>condition</i> is true (false).	<pre>if (<i>condition</i>) { ... } else { ... }</pre>
switch	Used to write a decision with scalar values (integers, characters) that select the alternative to be executed. Executes the <i>statements</i> following the <i>label</i> that is the <i>selector</i> value. Execution falls through to the next case if there is no return or break . Executes the statements following default if the <i>selector</i> value does not match any <i>label</i> .	<pre>switch (<i>selector</i>) { case <i>label</i> : <i>statements</i>; break; case <i>label</i> : <i>statements</i>; break; ... default : <i>statements</i>; }</pre>
while	Used to write a loop that specifies the repetition <i>condition</i> in the loop header. The <i>condition</i> is tested before each iteration of the loop and, if it is true, the loop body executes; otherwise, the loop is exited.	<pre>while (<i>condition</i>) { ... }</pre>
for	Used to write a loop that specifies the <i>initialization</i> , repetition <i>condition</i> , and <i>update</i> steps in the loop header. The <i>initialization</i> statements execute before loop repetition begins; the <i>condition</i> is tested before each iteration of the loop and, if it is true, the loop body executes; otherwise, the loop is exited. The <i>update</i> statements execute after each iteration.	<pre>for (<i>initialization</i>; <i>condition</i>; <i>update</i>) { ... }</pre>

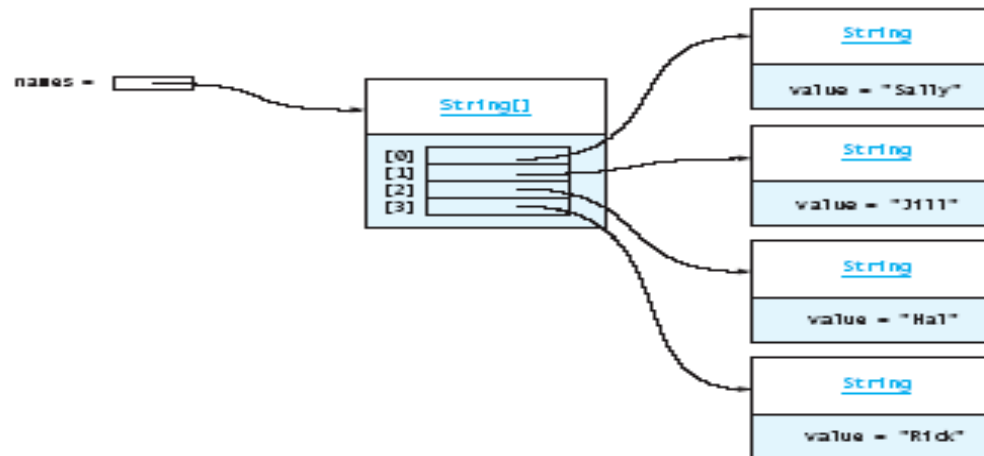
Java Control Statements

TABLE A.4 (continued)

Control Structure	Purpose	Syntax
<code>do ... while</code>	Used to write a loop that specifies the repetition <i>condition</i> after the loop body. The <i>condition</i> is tested after each iteration of the loop and, if it is true, the loop body is repeated; otherwise, the loop is exited. The loop body always executes at least one time.	<pre>do { ... while (<i>condition</i>) ;</pre>

Arrays

- In Java, an array is also an object
- The elements are indexes and are referenced using the form **arrayvar[subscript]**
- Declaring: `Int a[]=new int[12];`





JAVA Classes

- A *class* is the fundamental concept in JAVA (and other OOPLs)
- A class describes some data object(s), and the operations (or methods) that can be applied to those objects
- Every object and method in Java belongs to a class
- Classes have data (fields) and code (methods) and classes (member classes or inner classes)
- Static methods and fields belong to the class itself
- Others belong to instances



Java Classes-Example

```
public class Circle {  
    // A class field  
    public static final double PI= 3.14159;    // A useful constant  
  
    // A class method: just compute a value based on the arguments  
    public static double radiansToDegrees(double rads) {  
        return rads * 180 / PI;  
    }  
  
    // An instance field  
    public double r;                // The radius of the circle  
  
    // Two methods which operate on the instance fields of an object  
    public double area() {          // Compute the area of the circle  
        return PI * r * r;  
    }  
    public double circumference() { // Compute the circumference of the circle  
        return 2 * PI * r;  
    }  
}
```



Constructors

- Classes should define one or more methods to create or construct instances of the class
- Their name is the same as the class name
 - note deviation from convention that methods begin with lower case
- Constructors are differentiated by the number and types of their arguments
 - An example of overloading
- If you don't define a constructor, a default one will be created.

Extending a class

- Class hierarchies reflect subclass-superclass relations among classes.
- One arranges classes in hierarchies:
 - A class inherits instance variables and instance methods from all of its superclasses. Tree -> BinaryTree -> BST
 - You can specify only ONE superclass for any class.
- **Overloading** occurs when Java can distinguish two procedures with the same name by examining the number or types of their parameters. (example)
- **Shadowing or overriding** occurs when two procedures with the same signature (name, the same number of parameters, and the same parameter types) are defined in different classes, one of which is a superclass of the other. (example)



Extending a class-Example

```
public class PlaneCircle extends Circle {  
    // We automatically inherit the fields and methods of Circle,  
    // so we only have to put the new stuff here.  
    // New instance fields that store the center point of the circle  
    public double cx, cy;  
  
    // A new constructor method to initialize the new fields  
    // It uses a special syntax to invoke the Circle() constructor  
    public PlaneCircle(double r, double x, double y) {  
        super(r);          // Invoke the constructor of the superclass, Circle()  
        this.cx = x;        // Initialize the instance field cx  
        this.cy = y;        // Initialize the instance field cy  
    }  
  
    // The area() and circumference() methods are inherited from Circle  
    // A new instance method that checks whether a point is inside the circle  
    // Note that it uses the inherited instance field r  
    public boolean isInside(double x, double y) {  
        double dx = x - cx, dy = y - cy;           // Distance from center  
        double distance = Math.sqrt(dx*dx + dy*dy); // Pythagorean theorem  
        return (distance < r);                      // Returns true or false  
    }  
}
```



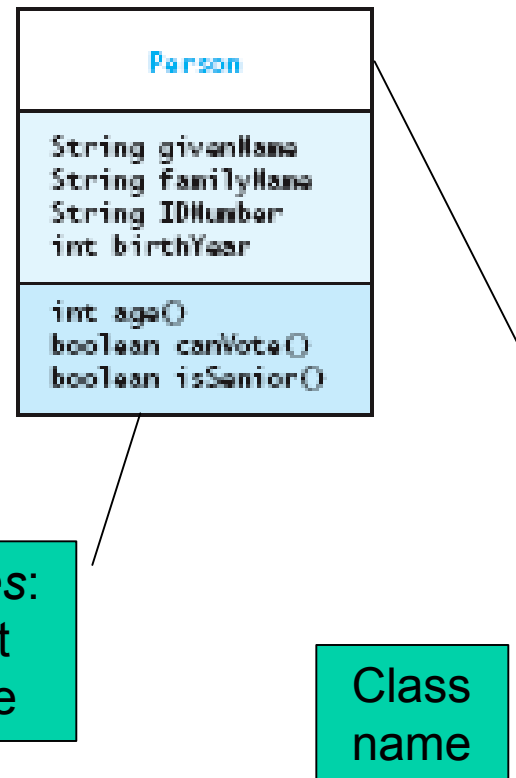
Data hiding and encapsulation

- Data-hiding or encapsulation is an important part of the OO paradigm.
- Classes should carefully control access to their data and methods in order to
 - Hide the irrelevant implementation-level details so they can be easily changed
 - Protect the class against accidental or malicious damage.
 - Keep the externally visible class simple and easy to document
- Java has a simple access control mechanism to help with encapsulation
 - Modifiers: public, protected, private, and package (default)

Defining Your Own Classes

- *Unified Modeling Language* (UML) is a standard diagram notation for describing a class

FIGURE A.6
Class Diagram for
Person



Method *signatures*:
name, argument
types, result type

Class
name

Input/Output using Class `JOptionPane`

- Java 1.2 and higher provide class `JOptionPane`, which facilitates display

TABLE A.13


Methods from Class `JOptionPane`

Method	Behavior
<code>static String showInputDialog(String prompt)</code>	Displays a dialog window that displays the argument as a prompt and returns the character sequence typed by the user.
<code>static void showMessageDialog(Object parent, String message)</code>	Displays a window containing a message string (the second argument) inside the specified container (the first argument).

FIGURE A.15

A Dialog Window (Left) and Message Window (Right)





Converting Numeric Strings to Numbers

- A dialog window always returns a reference to a **String**
- Therefore, a conversion is required, using **static** methods of class **String**:

TABLE A.14

Methods for Converting Strings to Numbers

Method	Behavior
<code>static int parseInt(String)</code>	Returns an <code>int</code> value corresponding to its argument string. A <code>NumberFormatException</code> occurs if its argument string contains characters other than digits.
<code>static double parseDouble(String)</code>	Returns a <code>double</code> value corresponding to its argument string. A <code>NumberFormatException</code> occurs if its argument string does not represent a real number.



Some Useful notes

- The **javadoc** program generates HTML API documentation from the “javadoc” style comments in your code.
- Integrated Development Environment (**IDE**)
 - Tools to support software development
 - Several Java IDE's are as powerful as C / C++ IDE's
- **Applet**
 - Program that runs in
 - **appletviewer** (test utility for applets)
 - Web browser (IE, Communicator)
 - Executes when HTML document containing applet is opened



Some Useful notes

- **API** = *Application Programming Interface*
- Java = small core + extensive collection of packages
- A **package** consists of some related Java classes:
 - Swing: a GUI (graphical user interface) package
 - AWT: Application Window Toolkit (more GUI)
 - util: utility data structures

Java Application: Adding Integers

```
1 // Fig. 24.6: Addition.java
2 // An addition program
3
4 import javax.swing.JOptionPane; // import class JOptionPane
5
6 public class Addition {
7     public static void main( String args[] )
8     {
9         String firstNumber,    // first string entered by user
10            secondNumber;      // second string entered by user
11         int number1,          // first number to add
12            number2,           // second number to add
13            sum;                // sum of number1 and number2
14
15         // read in first number from user as a string
16         firstNumber =
17             JOptionPane.showInputDialog( "Enter first integer" );
18
19         // read in second number from user as a string
20         secondNumber =
21             JOptionPane.showInputDialog( "Enter second integer" );
22
23         // convert numbers from type String to type int
24         number1 = Integer.parseInt( firstNumber );
25         number2 = Integer.parseInt( secondNumber );
26
27         // add the numbers
28         sum = number1 + number2;
29
30         // display the results
```


Java Application: Adding Integers

```
31     JOptionPane.showMessageDialog(  
32         null, "The sum is " + sum, "Results",  
33         JOptionPane.PLAIN_MESSAGE );  
34  
35     System.exit( 0 );    // terminate the program  
36 }  
37 }
```





The Person Class

```
public class Person {
```

```
    private String givenName;  
    private String familyName;  
    private String IDNumber;  
    private int birthYear;
```



The Person Class

```
// constructors: fill in new objects
public Person(String first, String family,
               String ID, int birth) {
    this.givenName  = first;
    this.familyName = family;
    this.IDNumber   = ID;
    this.birthYear  = birth;
}
public Person (String ID) {
    this.IDNumber = ID;
}
```



The Person Class

```
// modifier and accessor for givenName  
public void setGivenName (String given) {  
    this.givenName = given;  
}  
  
public String getGivenName () {  
    return this.givenName;  
}
```



The Person Class

```
// more interesting methods ...  
public int age (int inYear) {  
    return inYear - birthYear;  
}  
public boolean canVote (int inYear) {  
    int theAge = age(inYear);  
    return theAge >= VOTE_AGE;  
}
```

The Person Class

```
// "printing" a Person
public String toString () {
    return "Given name: " + givenName + "\n"
        + "Family name: " + familyName + "\n"
        + "ID number: " + IDNumber + "\n"
        + "Year of birth: " + birthYear +
        "\n";
}
```