

Introduction to Java Applications and Applets

Outline

- 1 Introduction
- 2 Basics of a Typical Java Environment
- 3 General Notes about Java and This Book
- 4 A Simple Program: Printing a Line of Text
- 5 Another Java Application: Adding Integers
- 6 Sample Applets from the Java 2 Software Development Kit
- 7 A Simple Java Applet: Drawing a String
- 8 Two More Simple Applets: Drawing Strings and Lines
- 9 Another Java Applet: Adding Integers

1 Introduction

- Java
 - Powerful, object-oriented language
 - Fun to use for beginners, appropriate for experience programmers
 - Language of choice for Internet and network communications
- In the Java ,we discuss
 - Graphics (and graphical user interfaces [GUI])
 - Multimedia
 - Event-driven programming

2 Basics of a Typical Java Environment

- 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**)

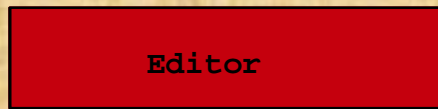
2 Basics of a Typical Java Environment (II)

- 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

2 Basics of a Typical Java Environment (II)

- 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
 - Program may not work on first try
 - Make changes in edit phase and repeat

Phase 1



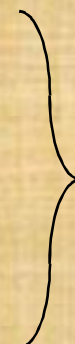
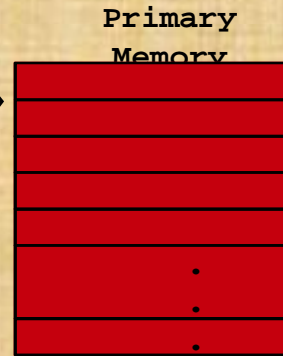
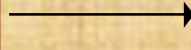
Program is created in the editor and stored on disk.

Phase 2



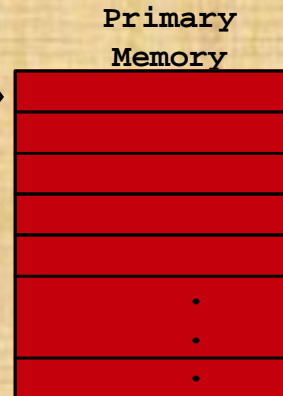
Compiler creates bytecodes and stores them on disk.

Phase 3



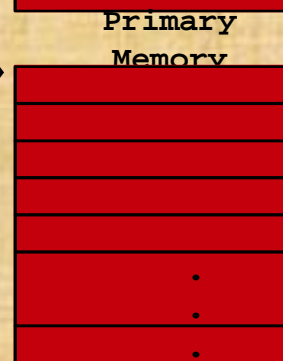
Class loader puts bytecodes in memory.

Phase 4



Bytecode verifier confirms that all bytecodes are valid and do not violate Java's security restrictions.

Phase 5



Interpreter reads bytecodes and translates them into a language that the computer can understand, possibly storing data values as the program executes.

3 General Notes about Java

- Java
 - Powerful language
 - Programming
 - Clarity - Keep it Simple
 - Portability - Java portable, but it is an elusive goal
 - Some details of Java not covered
 - <http://java.sun.com> for documentation
 - Performance
 - Interpreted programs run slower than compiled ones
 - Compiling has delayed execution, interpreting executes immediately
 - Can compile Java programs into machine code
 - Runs faster, comparable to C / C++

3 General Notes about Java

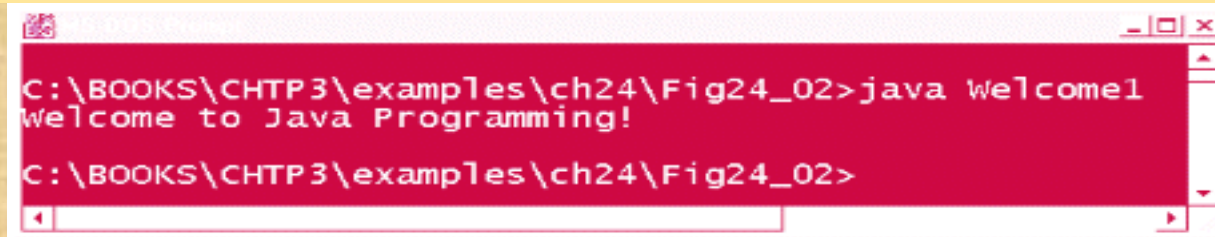
- Just-in-time compiler
 - Midway between compiling and interpreting
 - As interpreter runs, compiles code and executes it
 - Not as efficient as full compilers
 - Being developed for Java
 - Integrated Development Environment (IDE)
 - Tools to support software development
 - Several Java IDE's are as powerful as C / C++ IDE's

4 A Simple Program: Printing a Line of Text

- Application

- Program that runs using Java interpreter (discussed later)

```
1 // Fig.2: Welcome1.java
2 // A first program in Java
3
4 public class Welcome1 {
5     public static void main( String args[] )
6     {
7         System.out.println( "Welcome to Java Programming!" );
8     }
9 }
```



```
C:\BOOKS\CHTP3\examples\ch24\Fig24_02>java Welcome1
Welcome to Java Programming!
C:\BOOKS\CHTP3\examples\ch24\Fig24_02>
```

- Comments

- Java uses C-style `//` (preferred by Java programmers)
- Can also use `/* ... */`

4 A Simple Program: Printing a Line of Text (II)

- `public class Welcome1 {`
 - Begins class definition
 - Every Java program has a user-defined class
 - Use keyword (reserved word) **class** followed by **ClassName**
 - Name format - **MyClassName**
 - Identifier - letters, digits, underscores, dollar signs, does not begin with a digit, contains no spaces
 - Java case sensitive
 - **public** - For Chapters 1 and 25, every class will be **public**
 - Later, discuss classes that are not (Chapter 26)
 - Programmers initially learn by mimicking features. Explanations come later.
 - When saving a file, class name must be part of file name
 - Save file as **Welcome1.java**

4 A Simple Program: Printing a Line of Text (III)

- Braces
 - Body - delineated by left and right braces
 - Class definitions
- `public static void main(String args[])`
 - Part of every Java application
 - Program begins executing at **main**
 - Must be defined in every Java application
 - **main** is a method (a function)
 - **void** means method returns nothing
 - Many methods can return information
 - Braces used for method body
 - For now, mimic **main**'s first line

4 A Simple Program: Printing a Line of Text (IV)

- `System.out.println("Welcome to Java Programming!");`
 - Prints string
 - String - called character string, message string, string literal
 - Characters between quotes a generic string
 - **System.out** - standard output object
 - Displays information in command window
 - Method **System.out.println**
 - Prints a line of text in command window
 - When finished, positions cursor on next line
 - Method **System.out.print**
 - As above, except cursor stays on line
 - `\n` - newline
 - Statements must end with **;**

4 A Simple Program: Printing a Line of Text (V)

- Executing the program
 - `javac Welcome1`
 - Creates `Welcome1.class` (containing bytecodes)
 - `java Welcome1`
 - Interprets bytecodes in `Welcome1.class` (`.class` left out in `java` command)
 - Automatically calls `main`
- Output types
 - Command window
 - Dialog box / Windows

4 A Simple Program: Printing a Line of Text (VI)

- Packages
 - Predefined, related classes grouped by directories on disk
 - All in directory **java** or **javax**, or subdirectories
 - Referred to collectively as the Java class library or the Java applications programming interface (Java API)
 - **import** - locates classes needed to compile program
- Class **JOptionPane**
 - Defined in package called **javax.swing**
 - Contains classes used for a graphical user interface (GUI)
 - Facilitates data entry and data output
 - **import javax.swing.JOptionPane;**

4 A Simple Program: Printing a Line of Text (VII)

- Class **JOptionPane**
 - Contains methods that display a dialog box
 - **static** method **showMessageDialog**
 - First argument - **null** (more Chapter 29)
 - Second argument - string to display
- **static** methods
 - Called using dot operator (.) then method name
`JOptionPane.showMessageDialog(arguments);`
 - **exit** - method of class **System**
 - Terminates application, required in programs with GUIs
`System.exit(0);`
 - 0 - normal exit
 - non-zero** - signals that error occurred
 - Class **System** in package **java.lang**
 - Automatically imported in every Java program

```
1 // Fig. 4: Welcome2.java
2 // Printing multiple lines in a dialog box
3 import javax.swing.JOptionPane; // import class JOptionPane
4
5 public class Welcome2 {
6     public static void main( String args[] )
7     {
8         JOptionPane.showMessageDialog(
9             null, "Welcome\nto\nJava\nProgramming!" );
10
11         System.exit( 0 ); // terminate the program
12     }
13 }
```



1. import statement

2. Define class

3. main

4.
JOptionPane.showMessageDialog

5. System.exit

Program Output

5 Another Java Application: Adding Integers

- Variables
 - Locations in memory that hold data
 - Must be declared with name and data type before use
 - Primitive data types (keywords): **boolean, char, byte, short, int, long, float, double** (details in Chapter 25)
 - **String** (java.lang) - hold strings: **"Hi" "37"**
 - **int** - holds integers: **-1, 0, 15**
 - Name format - first letter lowercase, new word capitalized
 - **myVariable, myOtherVariable**
 - Declarations: specify name and type
 - Can have multiple variables per declaration
 - **int myInt, myInt2, myInt3;**
 - **String myString, myString2;**

5 Another Java Application: Adding Integers (II)

- Method **showInputDialog**
 - Of class **JOptionPane**
 - Displays prompt (gets user input)
 - Argument - Text to display in prompt
 - Java does not have a simple form of input
 - Nothing analogous to **System.out.print**
 - Returns what user input
 - Assign input to a variable using assignment operator =
myString = JOptionPane.showInputDialog("Enter an integer");
 - = has two operands (binary operator)
 - Expression on right evaluated, assigned to variable on left

5 Another Java Application: Adding Integers (III)

- **Integer.parseInt**
 - **static** method of class **Integer**
 - Input from **showInputDialog** a **String**
 - Want to convert it into an integer
 - **parseInt** takes a **String**, returns an integer
 - **myInt = Integer.parseInt(myString);**
 - Note assignment operator
- **The + operator**
 - String concatenation - "adding" strings
 - **"Hello" + " there "** same as **"Hello there"**
 - Print variables
 - **"myInt has a value of: " + myInt**
 - Used for addition, as in C / C ++:
 - **sum = int1 + int2;**

5 Another Java Application: Adding Integers (III)

- **showOptionDialog**
 - Another version
 - First argument: **null**
 - Second: message to display
 - Third: string to display in title bar
 - Fourth: type of message to display
 - **JOptionPane.PLAIN_MESSAGE**
 - Other types in Fig. 24.7


```

1 // Fig. 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

```

1. import

2. main

2.1 Declare variables

2.2 showInputDialog

2.3 Assign input to
firstNumber

2.4 Repeat for
secondNumber

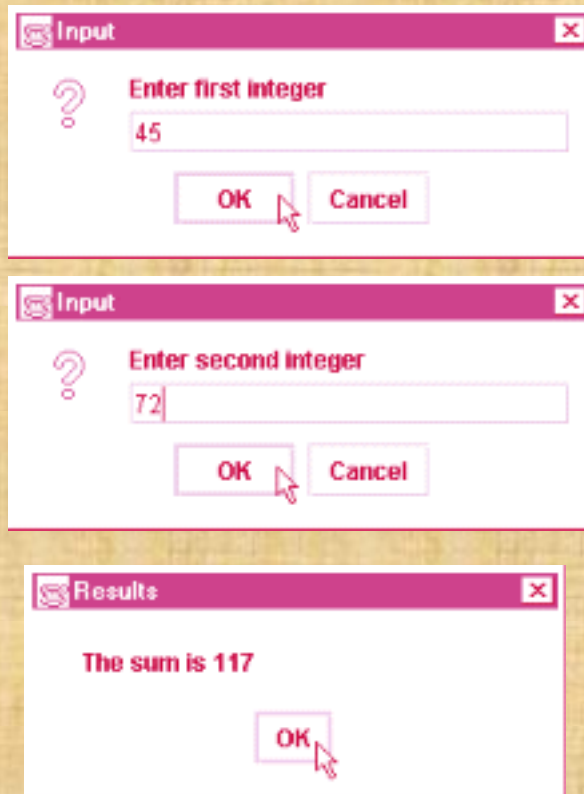
2.5 Convert strings
to ints

2.6 Sum the numbers

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

**2.7 Use
showMessageDialog
to display results**

Program Output



6 Sample Applets from the Java 2 Software Development Kit

- Applet
 - Program that runs in
 - **appletviewer** (test utility for applets)
 - Web browser (IE, Communicator)
 - Executes when HTML document containing applet is opened
- Sample Applets
 - Provided in Java 2 Software Development Kit (J2SDK)
 - Source code included (**.java** files)
 - Located in **demo** directory of J2SDK install

6 Sample Applets from the Java 2 Software Development Kit

- Running applets
 - In command prompt, change to subdirectory of applet
cd directoryName
 - There will be an HTML file used to execute applet
 - type **appletviewer example1.html**
 - Applet will run, **Reload** and **Quit** commands under **Applet** menu
- Example applets
 - Tic-Tac-Toe
 - Drawing programs
 - Animations
 - See Fig. 24.8

7 A Simple Java Applet: Drawing a String

- Create our own applet
 - Print "Welcome to Java Programming!"
 - `import javax.swing.JApplet`
 - Needed for all applets
 - `import java.awt.Graphics`
 - Allows program to draw graphics (lines, ovals, text) on an applet
 - Like applications, applets have at least one class definition
- Rarely create applets from scratch
 - Use pieces of class existing definitions

```
public class WelcomeApplet extends JApplet {
```

 - `extends ClassName` - class to inherit from
 - In this case, inherit from class `JApplet`

7 A Simple Java Applet: Drawing a String (II)

- Inheritance
 - **JApplet** is superclass (base class)
 - **WelcomeApplet** is subclass (derived class)
 - Derived class inherits data and methods of base class
 - Can add new features to derived class
 - Benefits
 - Someone else has already defined what an applet is
 - Applets require over 200 methods to be defined!
 - By using inheritance, all those methods are now ours
 - We do not need to know all the details of **JApplet**

7 A Simple Java Applet: Drawing a String (III)

- Classes
 - Templates/blueprints create or instantiate objects
 - Objects - locations in memory to store data
 - Implies that data and methods associated with object
- Methods
 - **paint**, **init**, and **start** called automatically for all applets
 - Get "free" version when you inherit from JApplet
 - By default, have empty bodies
 - Must override them and define yourself

7 A Simple Java Applet: Drawing a String (IV)

- Method **paint**

- Used to draw graphics, define:

```
public void paint( Graphics g )
```

- Takes a **Graphics** object **g** as a parameter
 - For now, all method definitions begin with **public**

- Call methods of object **g** to draw on applet

```
drawString("String to draw", x, y);
```

- Draws "**String to draw**" at location (**x,y**)
 - Coordinates specify bottom left corner of string
 - (**0, 0**) is upper left corner of screen
 - Measured in pixels (picture elements)

7 A Simple Java Applet: Drawing a String (IV)

- Create the HTML file (**.html** or **.htm**)
 - Many HTML codes (tags) come in pairs
`<myTag> ... </myTag>`
 - Create **<HTML>** tags with **<applet>** tags inside
 - **appletviewer** only understands **<applet>** tags
 - Minimal browser
 - Specify compiled **.class** file, width, and height of applet (in pixels)
`<applet code = "WelcomeApplet.class" width = 300
height = 30>`
 - Close tag with **</applet>**
- Running the applet
`appletviewer WelcomeApplet.html`

```
1 // Fig. 13: WelcomeApplet.java
2 // A first applet in Java
3 import javax.swing.JApplet; // import class JApplet
4 import java.awt.Graphics; // import class Graphics
5
6 public class WelcomeApplet extends JApplet {
7     public void paint( Graphics g )
8     {
9         g.drawString( "Welcome to Java Programming!", 25, 25 );
10    }
11 }
```

1. import

2. Define class
(extends JApplet)

2.1 Override paint

2.2 g.drawString

```
1 <html>
2 <applet code="WelcomeApplet.class" width=300 height=30>
3 </applet>
4 </html>
```

HTML file



Output

8 Two More Simple Applets: Drawing Strings and Lines

- Other methods of class **Graphics**
 - No concept of lines of text, as in **System.out.println** when drawing graphics
 - To print multiple lines, use multiple **drawString** calls
 - **drawLine(x1, y1, x2, y2) ;**
 - Draws a line from (**x1, y1**) to (**x2, y2**)

```
1 // Fig. 17: WelcomeLines.java
2 // Displaying text and lines
3 import javax.swing.JApplet; // import class JApplet
4 import java.awt.Graphics; // import class Graphics
5
6 public class WelcomeLines extends JApplet {
7     public void paint( Graphics g )
8     {
9         g.drawLine( 15, 10, 210, 10 );
10        g.drawLine( 15, 30, 210, 30 );
11        g.drawString( "Welcome to Java Programming!", 25, 25 );
12    }
13 }
```

1. import

2. Define class
(extends JApplet)

2.1 Override paint

Program Output



9 Another Java Applet: Adding Integers

- Next applet mimics program to add two integers
 - This time, use floating point numbers
 - Can have decimal point, **6.7602**
 - **float** - single precision floating point number (7 significant digits)
 - **double** - approximately double precision floating point number (15 significant digits)
 - Uses more memory
 - Use **showInputDialog** to get input, as before
 - Use **Double.parseDouble(String)**
 - Converts a **String** to a **double**

9 Another Java Applet: Adding Integers (II)

- **import** statements
 - Not necessary if specify full class name every time needed
 - public void paint(java.awt.Graphics g)**
 - * - indicates all classes in package should be available
 - **import java.swing.*;**
 - Recall that this contains **JApplet** and **JOptionPane**
 - Does not import subdirectories
- Instance variables
 - Variables declared in body of a class (not in a method)
 - Each object of class gets its own copy
 - Can be used inside any method of the class
 - Before, variables declared in **main**
 - Local variables, known only in body of method defined

9 Another Java Applet: Adding Integers (III)

- Instance variables
 - Have default values
 - Local variables do not, and require initialization before use
 - Good practice to initialize instance variables anyway
- Method **init**
 - Called automatically in all applets
 - Commonly used to initialize variables

```
public void init()
```
- References
 - Identifiers (such as **myString**) refer to objects
 - Contain locations in memory
 - References used to call methods, i.e. **g.drawString**

9 Another Java Applet: Adding Integers (III)

- Variables vs. Objects
 - Variables
 - Defined by a primitive data type
 - **char, byte, short, int, long, float, double, boolean**
 - Store one value at a time
 - Variable **myInt**
 - Objects defined in classes
 - Can contain primitive (built-in) data types
 - Can contain methods
 - **Graphics** object **g**
 - If data type a class name, then identifier is a reference
 - Otherwise, identifier is a variable

9 Another Java Applet: Adding Integers (IV)

- Other methods of class **Graphics**
 - `drawRect(x1, y1, x2, y2);`
 - Draws a rectangle with upper-left corner (**x1, y1**), and lower right corner (**x2, y2**)

```

1 // Fig. 19: AdditionApplet.java
2 // Adding two floating-point numbers
3 import java.awt.Graphics; // import class Graphics
4 import javax.swing.*; // import package javax.swing
5
6 public class AdditionApplet extends JApplet {
7     double sum; // sum of the values entered by the user
8
9     public void init()
10    {
11        String firstNumber, // first string entered by user
12            secondNumber; // second string entered by user
13        double number1, // first number to add
14            number2; // second number to add
15
16        // read in first number from user
17        firstNumber =
18            JOptionPane.showInputDialog(
19                "Enter first floating-point value" );
20
21        // read in second number from user
22        secondNumber =
23            JOptionPane.showInputDialog(
24                "Enter second floating-point value" );
25
26        // convert numbers from type String to type double
27        number1 = Double.parseDouble( firstNumber );
28        number2 = Double.parseDouble( secondNumber );
29
30        // add the numbers

```

1. import (note *)

2. Define class
(extends JApplet)

2.1 init

2.2 Declare variables

2.3 Input

2.4 Convert String to
double


```
31     sum = number1 + number2;
32 }
33
34 public void paint( Graphics g )
35 {
36     // draw the results with g.drawString
37     g.drawRect( 15, 10, 270, 20 );
38     g.drawString( "The sum is " + sum, 25, 25 );
39 }
40 }
```

2.5 sum numbers

2.6 paint

2.7 drawRect

2.8 drawString

```
1 <html>
2 <applet code="AdditionApplet.class" width=300 height=50>
3 </applet>
4 </html>
```

HTML file

Program Output

