

Java applet

```
1 // Fig. 3.6: 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 }
```

HTML file

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

Program Output



3.4 A Simple Java Applet: Drawing a String

```
1 // Fig. 3.6: WelcomeApplet.java
2 // A first applet in Java
```

- Lines that begin with `//` are comments
 - Gives name of source code and brief description of applet

```
3 import javax.swing.JApplet; // import class JApplet
4 import java.awt.Graphics; // import class Graphics
```

- As stated in Chapter 2, Java has predefined classes grouped into packages
 - **import** statements tell compiler where to locate classes used
 - When you create applets, **import** the **JApplet** class (package **javax.swing**)
 - **import** the **Graphics** class (package **java.awt**) to draw graphics
 - Can draw lines, rectangles ovals, strings of characters
 - **import** specifies directory structure

3.4 A Simple Java Applet: Drawing a String

- Applets have at least once class definition (like applications)
 - Rarely create classes from scratch
 - Use pieces of existing class definitions
 - Java uses inheritance to create new classes from old ones

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

- Begins **class** definition for class **WelcomeApplet**
 - Keyword **class** then class name
- **extends** followed by class name
 - Indicates the class to inherit from (**JApplet**)
 - **JApplet** : superclass (base class)
 - **WelcomeApplet** : subclass (derived class)
 - **WelcomeApplet** now has methods and data of **JApplet**

3.4 A Simple Java Applet: Drawing a String

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

- Someone else has defined "what it means to be an applet"
 - Class **JApplet** is defined for us
 - Applets require over 200 methods!
 - **extends JApplet** allows us to inherit methods
 - Do not have to define them all
 - Do not need to know every detail of class **JApplet**
- Class **WelcomeApplet** is a blueprint
 - Creates (instantiates) an object for use by program
 - **appletviewer** or browser creates an object of class **WelcomeApplet**
 - Keyword **public** required
 - File can only have one **public** class
 - **public** class name must be file name

3.4 A Simple Java Applet: Drawing a String

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

- Our class inherits method **paint** from **JApplet**
 - By default, **paint** has an empty body
 - We override (redefine) **paint** in our class
- Methods **paint**, **init**, and **start**
 - Guaranteed to be called automatically for us
 - Our applet gets a "free" version of these by inheriting from **JApplet**
 - Free versions have an empty body (do nothing)
 - Every applet does not need all three - override only the ones you need

3.4 A Simple Java Applet: Drawing a String

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

– Method **paint**

- Draws graphics on screen
- **void** means **paint** returns nothing when it finishes its task
- Parenthesis define parameter list - where methods receive data to perform tasks
 - Normally, data passed by programmer, as in **JOptionPane.showMessageDialog**
- **paint** gets parameters automatically
 - **Graphics** object used by **paint**
- Mimic **paint**'s first line

3.4 A Simple Java Applet: Drawing a String

```
8   {  
9       g.drawString( "Welcome to Java Programming!", 25, 25 );  
10  }
```

– Body of **paint**

- Method **drawString** (of class **Graphics**)
- Called using **Graphics** object **g** and dot operator (**.**)
- Method name followed by parenthesis containing argument list
 - First argument: **String** to draw
 - Second: x coordinate of location to draw at (in pixels)
 - Third: y coordinate of location to draw at (in pixels)

– Java coordinate system

- Measured in pixels (picture elements)
- Upper left is **(0,0)**



3.4 A Simple Java Applet: Drawing a String

- Line 11: Right brace to end class **WelcomeApplet**
- Running the applet
 - Compile
 - **javac WelcomeApplet.java**
 - If no errors, bytecodes stored in **WelcomeApplet.class**
 - We must create an HTML file
 - Loads the applet into **appletviewer** or a browser
 - Ends in **.htm** or **.html**
 - To execute an applet
 - Create an HTML file indicating which applet the browser (or **appletviewer**) should load and execute

3.4 A Simple Java Applet: Drawing a String

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

- Simple HTML file (**WelcomeApplet.html**)
 - Usually in same directory as **.class** file
 - Remember, **.class** file created after compilation
- HTML codes (tags)
 - Usually come in pairs
 - Begin with **<** and end with **>**
- Lines 1 and 4 - begin and end the HTML tags
- Line 2 - begins **<applet>** tag
 - Specifies code to use for applet
 - Specifies **width** and **height** of display area in pixels
- Line 3 - ends **<applet>** tag

3.4 A Simple Java Applet: Drawing a String

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

- **appletviewer** only understands **<applet>** tags
 - Ignores everything else
 - Minimal browser
- Executing the applet
 - **appletviewer WelcomeApplet.html**
 - Perform in directory containing **.class** file

```

1 // Fig. 3.6: 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 }

```

import allows us to use predefined classes (allowing us to use applets and graphics, in this case).

2. Class WelcomeApplet (extends JApplet)

extends allows us to inherit the capabilities of class **JApplet**.

Method **paint** is guaranteed to be called in all applets. Its first line must be defined as above.

```

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

```

HTML file



Program Output

3.5 Two More Simple Applets: Drawing Strings and Lines

- More applets
 - First example
 - Display two lines of text
 - Use **drawString** to simulate a new line
 - We will actually use two **drawString** statements
 - Second example
 - Method **drawLine(x1, y1, x2, y2)**
 - Draws a line from (x1, y1) to (x2, y2)
 - Remember that (0, 0) is upper left
 - Use **drawLine** to draw a line beneath and above a string

Outline

1. import

2. Class WelcomeApplet2 (extends JApplet)

3. paint

3.1 drawString

```
1 // Fig. 3.8: WelcomeApplet2.java
2 // Displaying multiple strings
3 import javax.swing.JApplet; // import class JApplet
4 import java.awt.Graphics; // import class Graphics
5
6 public class WelcomeApplet2 extends JApplet {
7     public void paint( Graphics g )
8     {
9         g.drawString( "Welcome to", 25, 25 );
10        g.drawString( "Java Programming!", 25, 40 );
11    }
12 }
```

The two **drawString** statements simulate a newline. In fact, the concept of lines of text does not exist when drawing strings.

```
1 <html>
2 <applet code="WelcomeApplet2.class" width=300 height=45>
3 </applet>
4 </html>
```

HTML file

Program Output



Outline

1. import

2. Class
WelcomeLines
(extends JApplet)

3. paint

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

Draw horizontal lines with
drawLine (endpoints have same
y coordinate).

```
1 <html>
2 <applet code="WelcomeLines.class" width=300 height=40>
3 </applet>
4 </html>
```

3.3 drawString

HTML file

Program Output



3.6 Another Java Applet: Adding Integers

- Next applet
 - Mimics application for adding two integers
 - This time, use floating point numbers (numbers with a decimal point)
 - Show program, then we will discuss it

```
1 // Fig. 3.12: 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
27        // convert numbers from type String to type double
28        number1 = Double.parseDouble( firstNumber );
29        number2 = Double.parseDouble( secondNumber );
30
```

Outline

Applet

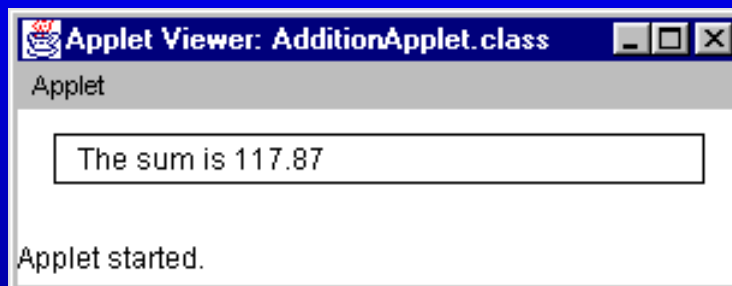
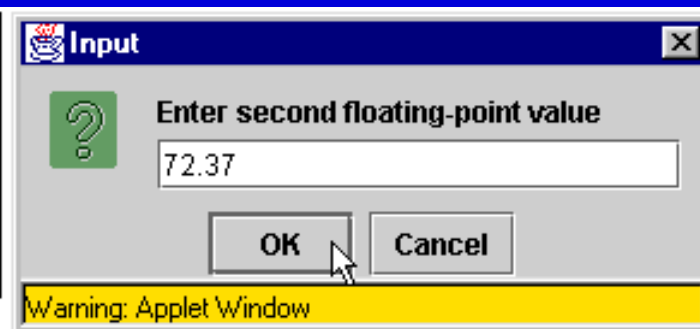
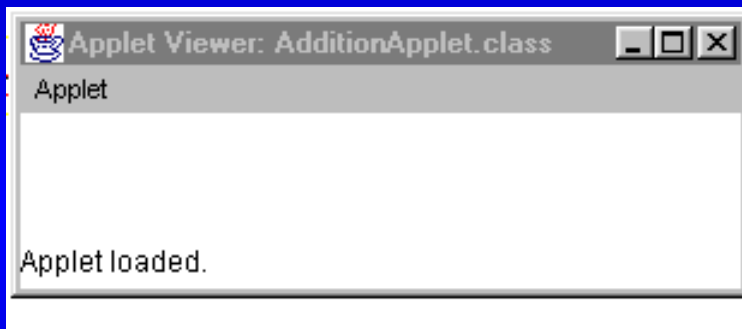
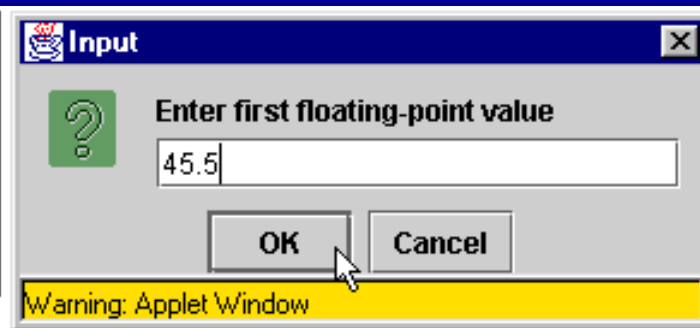
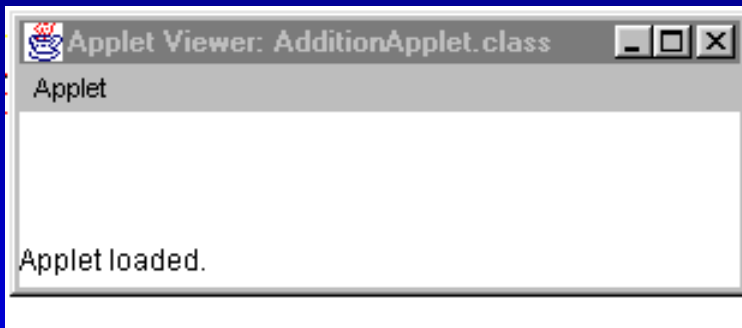
```
31     // add the numbers
32     sum = number1 + number2;
33 }
34
35 public void paint( Graphics g )
36 {
37     // draw the results with g.drawString
38     g.drawRect( 15, 10, 270, 20 );
39     g.drawString( "The sum is " + sum, 25, 25 );
40 }
41 }
```

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

HTML file

Outline

Program Output



3.6 Another Java Applet: Adding Integers

- Lines 1-2: Comments

```
3 import java.awt.Graphics;    // import class Graphics
4 import javax.swing.*;       // import package javax.swing
```

- Line 3: **imports** class **Graphics**
 - **import** not needed if use full package and class name
`public void paint (java.awt.Graphics g)`
- Line 4: specifies entire **javax.swing** package
 - ***** indicates all classes in **javax.swing** are *available*
 - Include **JApplet** and **JOptionPane**
 - Allows programmer to use shorthand name
 - Use **JOptionPane** instead of
`javax.swing.JOptionPane`
 - ***** does not load all classes in **javax.swing**
 - Compiler only loads the classes it uses

3.6 Another Java Applet: Adding Integers

```
6 public class AdditionApplet extends JApplet {
```

- Begin class definition
 - Inherit from **JApplet**, **imported** from package **javax.swing**

```
7     double sum; // sum of the values entered by the user
```

- Instance variable declaration
 - Every object of class gets a separate copy of the instance variable
 - Declared in body of class, but not inside methods
 - Variables declared in methods are *local variables*
 - Can only be used in body of method
 - Instance variables can be used anywhere in class
 - Have default value (**0 . 0** in this case)

3.6 Another Java Applet: Adding Integers

```
7 double sum; // sum of the values entered by the user
```

– primitive data type **double**

- Used to store floating point (decimal) numbers

```
9 public void init()
```

– Method **init**

- Normally initializes instance variables
- Guaranteed to be first method called
- First line must always appear as above
 - Returns nothing (**void**), takes no arguments

```
10 {
```

– Begins body of method **init**

3.6 Another Java Applet: Adding Integers

```
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
```

- Declare variables
- Two types of variables
 - Reference variables (called references)
 - Refer to objects (contain location in memory)
 - » Objects defined in a class definition
 - » Can contain multiple data and methods
 - **paint** receives a reference called **g** to a **Graphics** object
 - Reference used to call methods on the **Graphics** object
 - Primitive data types (called variables)
 - Can only contain one piece of data

3.6 Another Java Applet: Adding Integers

```
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
```

- Distinguishing references and variables
 - If data type is a class name, then reference
 - **String** is a class
 - **firstNumber, secondNumber**
 - If data type a primitive type, then variable
 - **double** is a primitive data type
 - **number1, number2**

3.6 Another Java Applet: Adding Integers

```
16      // read in first number from user
17      firstNumber =
18          JOptionPane.showInputDialog(
19              "Enter first floating-point value" );
```

- Method **JOptionPane.showInputDialog**
 - Prompts user with string
 - User enters value in text field, clicks **OK**
 - Can type anything, but if not of correct type, error occurs
 - In Chapter 14 we learn how to accomodate this
 - Returns string user inputs
- Variable **firstNumber** gets returned string
 - Assignment statement
- Lines 21-24: As above, assigns input to **secondNumber**

3.6 Another Java Applet: Adding Integers

```
28     number1 = Double.parseDouble( firstNumber );
29     number2 = Double.parseDouble( secondNumber );
```

- **static** method **Double.parseDouble**
 - Converts **String** argument to a **double**
 - Returns the **double** value
- **firstNumber** converted to **double** and assigned to **number1**
 - **secondNumber** similar

```
32     sum = number1 + number2;
```

- Assignment statement
 - Sums **number1** and **number2**, puts result in **sum**
 - **sum** an instance variable, can use anywhere in class
 - Not defined in **init** but still used

3.6 Another Java Applet: Adding Integers

```
33     }
```

- Ends method **init**
 - **appletviewer** (or browser) calls method **start**
 - **start** usually used with multithreading
 - Advanced concept, in Chapter 15
 - We do not define it, so empty definition in **JApplet** used
 - Next, method **paint** called

```
38     g.drawRect( 15, 10, 270, 20 );
```

- Method **drawRect(x1, y1, width, height)**
 - Draws a rectangle with an upper left corner (**x1**, **y1**), with specified **width** and **height**

3.6 Another Java Applet: Adding Integers

39

```
g.drawString( "The sum is " + sum, 25, 25 );
```

- Sends **drawString** message to **Graphics** object to which **g** refers (calls method)
 - "The sum is" + **sum** - string concatenation
 - **sum** converted to a string
 - **sum** can be used, even though not defined in **paint**
 - Instance variable, can be used anywhere in class

Outline

```
1 // Fig. 3.12: 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 entered by user
14            number2; // second number entered by user
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
27        // convert numbers from type String to type double
28        number1 = Double.parseDouble( firstNumber );
29        number2 = Double.parseDouble( secondNumber );
30
```

1. import

2. Class

* allows any class in the the package to be used.

3. Instance variable

Instance variable **sum** may be used anywhere in the class, even in other methods.

Data type **double** can store floating point numbers.

4.2 showInputDialog

4.3 parseDouble

Outline

4.4 sum inputs

5. paint

5.1 drawRect

```
31 // add the numbers
32 sum = number1 + number2;
33 }
34
35 public void paint( Graphics g )
36 {
37 // draw the results with g.drawString
38 g.drawRect( 15, 10, 270, 20 );
39 g.drawString( "The sum is " + sum, 25, 25 );
40 }
41 }
```

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

drawRect takes the upper left coordinate, width, and height of the rectangle to draw.

Outline

Program Output

