Chapter 3 - Introduction to Java Applets

- Applet
  - –Program that runs in
    - **`appletviewer`** (test utility for applets)
    - Web browser (IE, Communicator) ➔ IIS setting !!!
  - –Executes when HTML (Hypertext Markup Language) document containing applet is opened and downloaded
  - –Applications run in command windows

The TicTacToe Applet

- Running applets
  - In command prompt, change to demo subdirectory of applet

    ```
    cd c:\j2sdk1.4.2_03\demo\applets
    cd appletDirectoryName
    ```
  - There will be an HTML file used to execute applet
  - Type **`appletviewer example1.html`**
    - **`appletviewer`** loads the html file specified as its command-line argument
    - From the HTML file, determines which applet to load
    - Applet will run, **Reload** and **Quit** commands under **Applet** menu

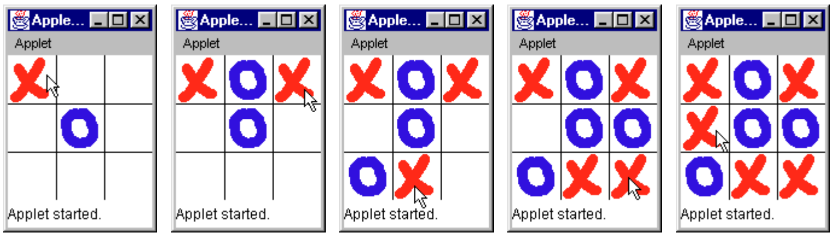
Java applet running environment

- Run from the click file and invoke the IE browser
- Use the MS-DOC command mode and type in the command ➔ appletviewer ***.htm
  - Remember to setup the JRE environment variable
    - set path=%path%;c:\j2sdk1.4.2_03\bin
    - set CLASSPATH=.;C:\j2sdk1.4.2_03\lib\tools.jar

The TicTacToe Applet

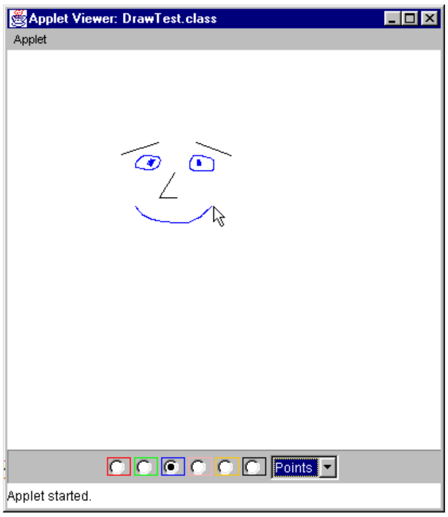
- You start as player "X"

Fig. 3.2 Sample execution of the `TicTacToe` applet.



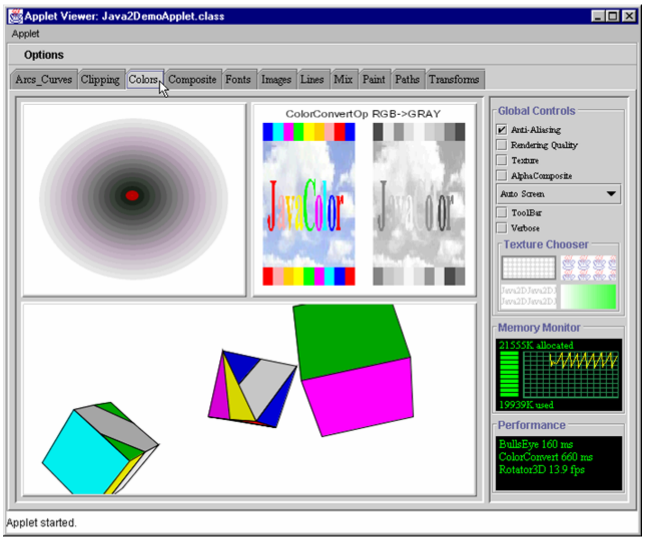
The DrawTest Applet

Fig. 3.4 Sample execution of applet **`DrawTest`**.



The Java2D Applet

- Demonstrates 2D drawing capabilities built into Java2



☑ Anti-Aliasing
□ Rendering Quality
□ Texture
□ AlphaComposite
Auto Screen
□ ToolBar
□ Verbose

21555K allocated
19939K used

BullsEye 160 ms
ColorConvert 660 ms
Rotator3D 13.9 fps

A Simple Java Applet: Drawing a String

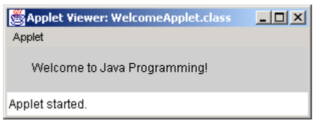
- Now, create applets of our own
  - Take a while before we can write applets like in the demos
  - Cover many of same techniques
- Upcoming program
  - Create an applet to display **`"Welcome to Java Programming!"`**
  - Show applet and HTML file, then discuss them line by line

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

// Java core packages
import java.awt.Graphics;     // import class Graphics

// Java extension packages
import javax.swing.JApplet;  // import class JApplet

public class WelcomeApplet extends JApplet {

   // draw text on applet’s background
   public void paint( Graphics g )
   {
      // call inherited version of method paint
      super.paint( g );

      // draw a String at x-coordinate
      g.drawString( "Welcome to Java P

   }  // end method paint

}  // end class WelcomeApplet
```

Java applet

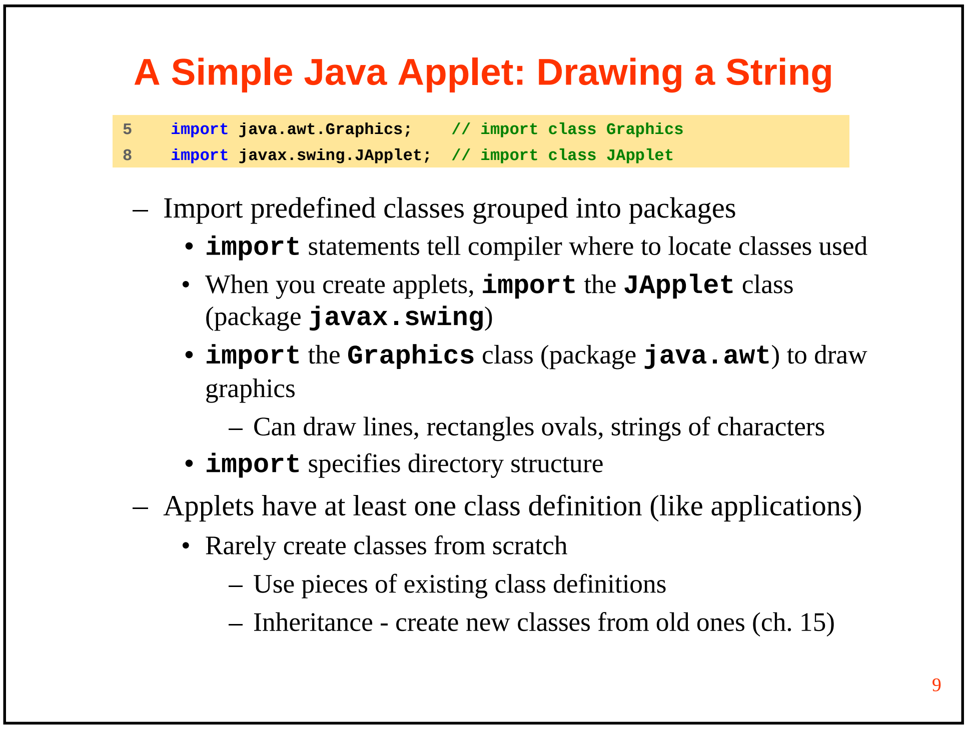
`extends` allows us to inherit the capabilities of class **`JApplet`**.

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



```
Welcome to Java Programming!

Applet started.
```

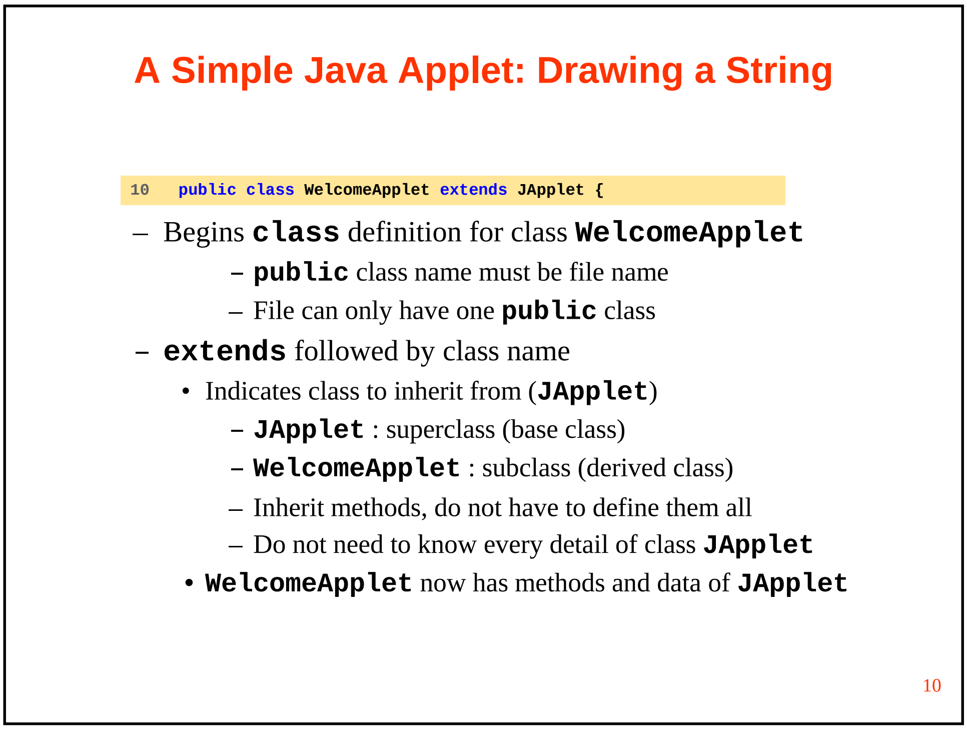
Program Output



A Simple Java Applet: Drawing a String

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

- Import 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
- Applets have at least one class definition (like applications)
  - Rarely create classes from scratch
    - Use pieces of existing class definitions
    - Inheritance - create new classes from old ones (ch. 15)



A Simple Java Applet: Drawing a String

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

- Begins **`class`** definition for class **`WelcomeApplet`**
    - **`public`** class name must be file name
    - File can only have one **`public`** class
- **`extends`** followed by class name
  - Indicates class to inherit from (**`JApplet`**)
    - **`JApplet`** : superclass (base class)
    - **`WelcomeApplet`** : subclass (derived class)
    - Inherit methods, do not have to define them all
    - Do not need to know every detail of class **`JApplet`**
  - **`WelcomeApplet`** now has methods and data of **`JApplet`**

A Simple Java Applet: Drawing a String

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

- Our class inherits method **`paint`** from **`JApplet`**
  - By default, **`paint`** has empty body
  - Override (redefine) **`paint`** in our class
- Methods **`paint`**, **`init`**, and **`start`**
  - Guaranteed to be called automatically
  - Our applet gets "free" version of these by inheriting from **`JApplet`**
    - Free versions have empty body (do nothing)
    - Every applet does not need all three methods
      - Override the ones you need
- Applet container “draws itself” by calling method **`paint`**

A Simple Java Applet: Drawing a String

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

- Method **`paint`**
  - Lines 13-21 are the definition of paint
  - Draws graphics on screen
  - **`void`** indicates **`paint`** returns nothing when finishes 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

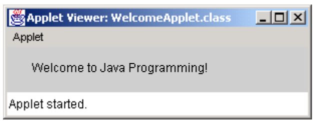
A Simple Java Applet: Drawing a String

```
        super.paint( g );
```

- Calls version of method paint from superclass **`JApplet`**
- Should be first statement in every applet’s paint method

```
        g.drawString( "Welcome to Java Programming!", 25, 25 );
```

- Body of **`paint`**
  - Method **`drawString`** (of class **`Graphics`**)
  - Called using **`Graphics`** object **`g`** and dot operator (**`.`**)
  - Method name, then parenthesis with arguments
    - First argument: **`String`** to draw
    - Second: x coordinate (in pixels) location
    - Third: y coordinate (in pixels) location
- Java coordinate system
  - Measured in pixels (picture elements)
  - Upper left is (**`0`**,**`0`**)

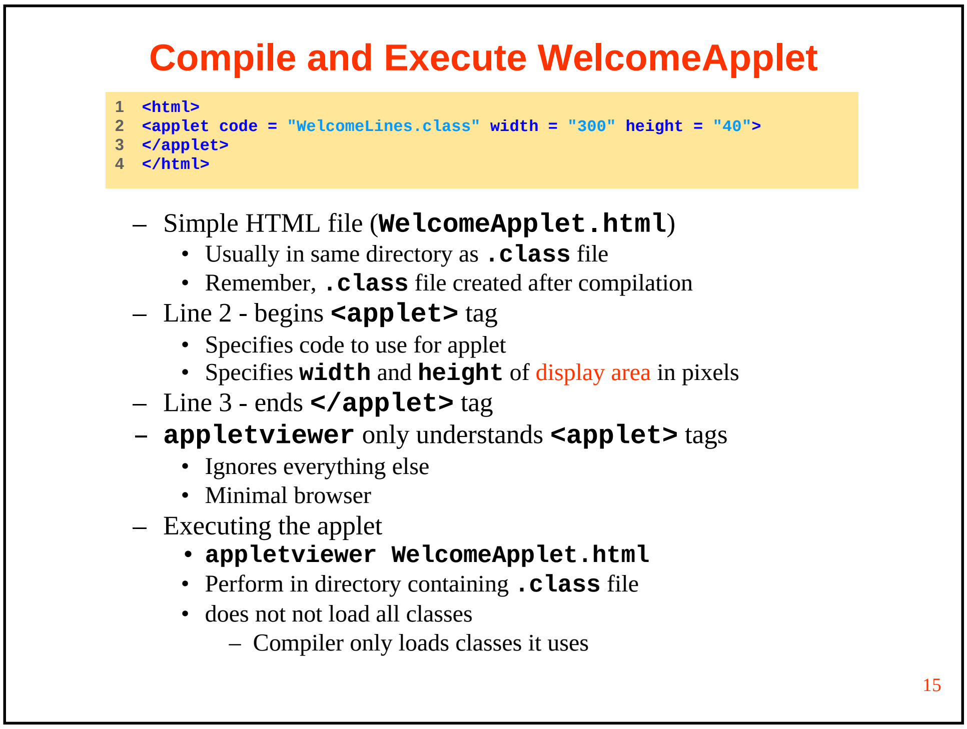


Welcome to Java Programming!

Applet started.

Compile and Execute WelcomeApplet

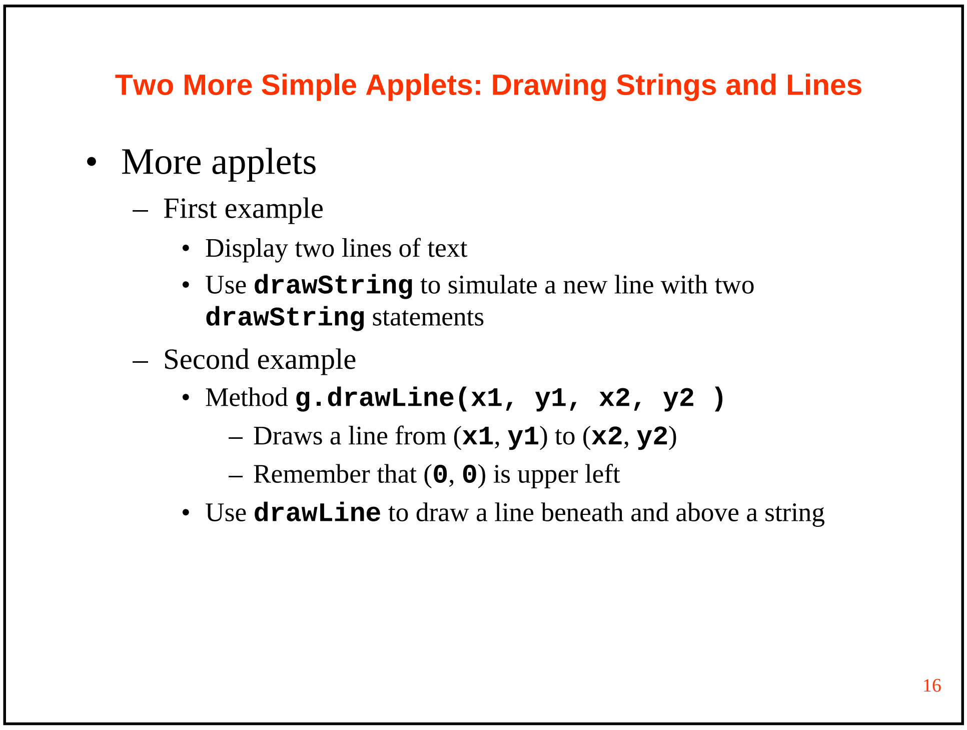
- Running the applet
  - Compile
    - **`javac WelcomeApplet.java`**
    - If no errors, bytecodes stored in **`WelcomeApplet.class`**
  - Create an HTML file
    - Loads the applet into **`appletviewer`** or a browser
    - Ends in **`.htm`** or **`.html`**
  - To execute an applet
    - Create an HTML file indicating which applet the browser (or **`appletviewer`**) should load and execute



Compile and Execute WelcomeApplet

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

- Simple HTML file (**`WelcomeApplet.html`**)
  - Usually in same directory as **`.class`** file
  - Remember, **`.class`** file created after compilation
- 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
- **`appletviewer`** only understands **`<applet>`** tags
  - Ignores everything else
  - Minimal browser
- Executing the applet
  - **`appletviewer WelcomeApplet.html`**
  - Perform in directory containing **`.class`** file
  - does not not load all classes
    - Compiler only loads classes it uses



Two More Simple Applets: Drawing Strings and Lines

- More applets
  - First example
    - Display two lines of text
    - Use **`drawString`** to simulate a new line with two **`drawString`** statements
  - Second example
    - Method **`g.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

```java
// Fig. 3.8: WelcomeApplet2.java
// Displaying multiple strings in an applet.

// Java core packages
import java.awt.Graphics;    // import class Graphics

// Java extension packages
import javax.swing.JApplet;  // import class JApplet

public class WelcomeApplet2 extends JApplet {

   // draw text on applet’s background
   public void paint( Graphics g )
   {
      // call inherited version of method paint
      super.paint( g );

      // draw two Strings at different locations
      g.drawString( "Welcome to", 25, 25 );
      g.drawString( "Java Programming!", 25, 40 );

   }  // end method paint

}  // end class WelcomeApplet2
```

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

1. **`import`**

2. **Class `WelcomeApplet2` (`extends JApplet`)**

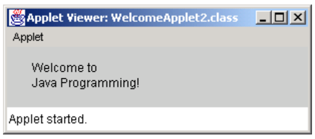
3. **`paint`**

3.1 **`drawString`**

3.2 **`drawString` on same x coordinate, but 15 pixels down**

```html
<html>
<applet code = "WelcomeApplet2.class" width = "300" height = "60">
</applet>
</html>
```

HTML file



```
Welcome to
Java Programming!
```

Applet started.

Program Output

```java
// Fig. 3.10: WelcomeLines.java
// Displaying text and lines

// Java core packages
import java.awt.Graphics;    // import class Graphics

// Java extension packages
import javax.swing.JApplet;  // import class JApplet

public class WelcomeLines extends JApplet {

   // draw lines and a string on applet’s background
   public void paint( Graphics g )
   {
      // call inherited version of method paint
      super.paint( g );

      // draw horizontal line from (15, 10) to (210, 10)
      g.drawLine( 15, 10, 210, 10 );

      // draw horizontal line from (15, 30) to (210, 30)
      g.drawLine( 15, 30, 210, 30 );

      // draw String between lines at location (25, 25)
      g.drawString( "Welcome to Java Programming!", 25, 25 );

   }  // end method paint

}  // end class WelcomeLines
```

WelcomeLines.java

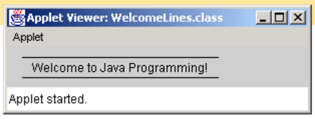
2. Class `WelcomeLines` (`extends JApplet`)

3. `paint`

3.1 `drawLine`

3.2 `drawLine`

3.3 `drawString`



Welcome to Java Programming!

Applet started.

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

Two More Simple Applets: Drawing Strings and Lines

- Method **`drawLine`** of class **`Graphics`**
  - Takes as arguments **`Graphics`** object and line’s end points
  - X and y coordinate of first endpoint
  - X and y coordinate of second endpoint

Another Java Applet: Add Floating-Point Numbers

- Next applet
  - Mimics application for adding two integers (Fig 2.9)
    - This time, use floating point numbers (numbers with a decimal point)
      - Using primitive data types
        - `Double` – double precision floating-point numbers
        - `Float` – single precision floating-point numbers
  - Show program, then discuss

```java
// Fig. 3.12: AdditionApplet.java
// Adding two floating-point numbers.

// Java core packages
import java.awt.Graphics;   // import class Graphics

// Java extension packages
import javax.swing.*;       // import package javax.swing

public class AdditionApplet extends JApplet {
   double sum;  // sum of values entered by user

   // initialize applet by obtaining values from user
   public void init()
   {
      String firstNumber;   // first string entered by user
      String secondNumber;  // second string entered by user
      double number1;       // first number to add
      double number2;       // second number to add

      // obtain first number from user
      firstNumber = JOptionPane.showInputDialog(
         "Enter first floating-point value" );

      // obtain second number from user
      secondNumber = JOptionPane.showInputDialog(
         "Enter second floating-point value" );

      // convert numbers from type String to type double
      number1 = Double.parseDouble( firstNumber );
      number2 = Double.parseDouble( secondNumber );

```

`AdditionApplet.java`

1. `import`

2. Class `AdditionApplet` (`extends JApplet`)

3. Instance variable

4. `init`

4.1 Declare variables

4.2 `showInputDialog`

4.3 `parseDouble`

```java
      // add numbers
      sum = number1 + number2;
   }

   // draw results in a rectangle on applet’s background
   public void paint( Graphics g )
   {
      // call inherited version of method paint
      super.paint( g );

      // draw rectangle starting from (15, 10) that is 270
      // pixels wide and 20 pixels tall
      g.drawRect( 15, 10, 270, 20 );

      // draw results as a String at (25, 25)
      g.drawString( "The sum is " + sum, 25, 25 );

   }  // end method paint

}  // end class AdditionApplet
```

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

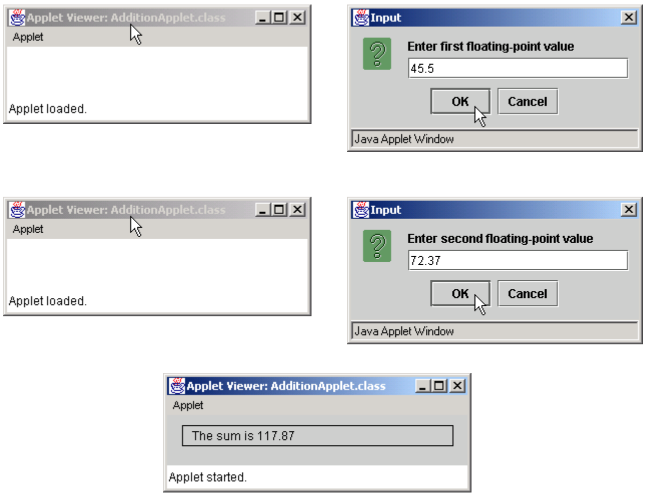
5. Draw applet contents

5.1 Draw a rectangle

5.2 Draw the results

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

HTML file



Applet Viewer: AdditionApplet.class

Applet

Applet loaded.

Input

Enter first floating-point value

45.5

OK Cancel

Java Applet Window

Applet Viewer: AdditionApplet.class

Applet

Applet loaded.

Input

Enter second floating-point value

72.37

OK Cancel

Java Applet Window

Applet Viewer: AdditionApplet.class

Applet

The sum is 117.87

Applet started.

Program Output

Another Java Applet: Adding Floating-Point Numbers

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

- Instance variable declaration
  - Each object of class gets own 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

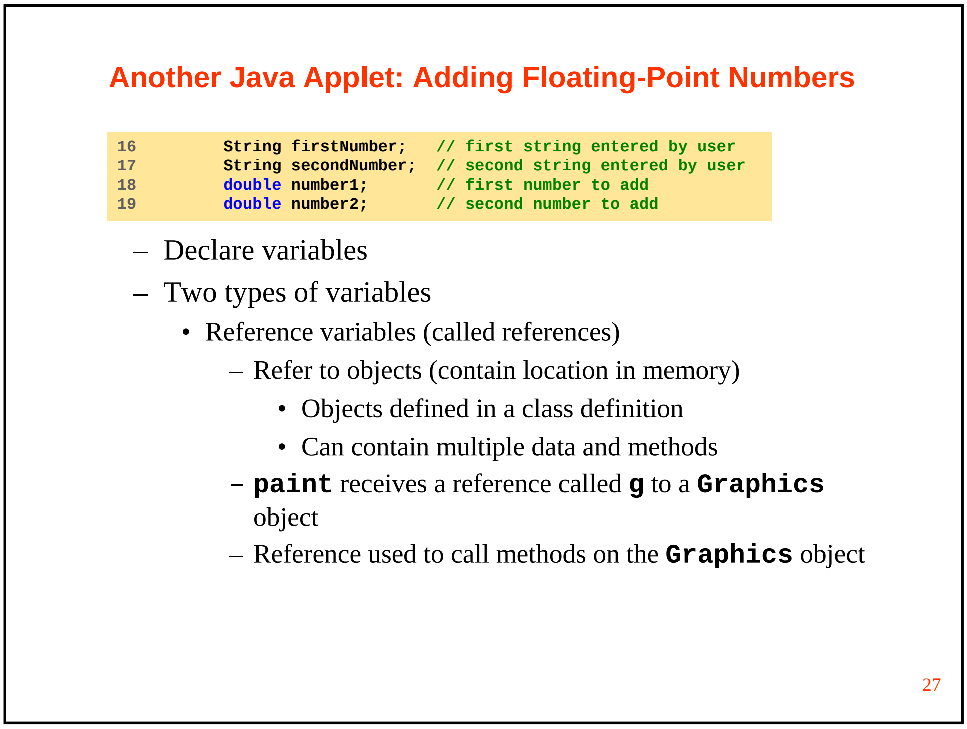
Another Java Applet: Adding Floating-Point Numbers

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

- Primitive data type **`double`**
  - Used to store floating point (decimal) numbers

```
   public void init()
```

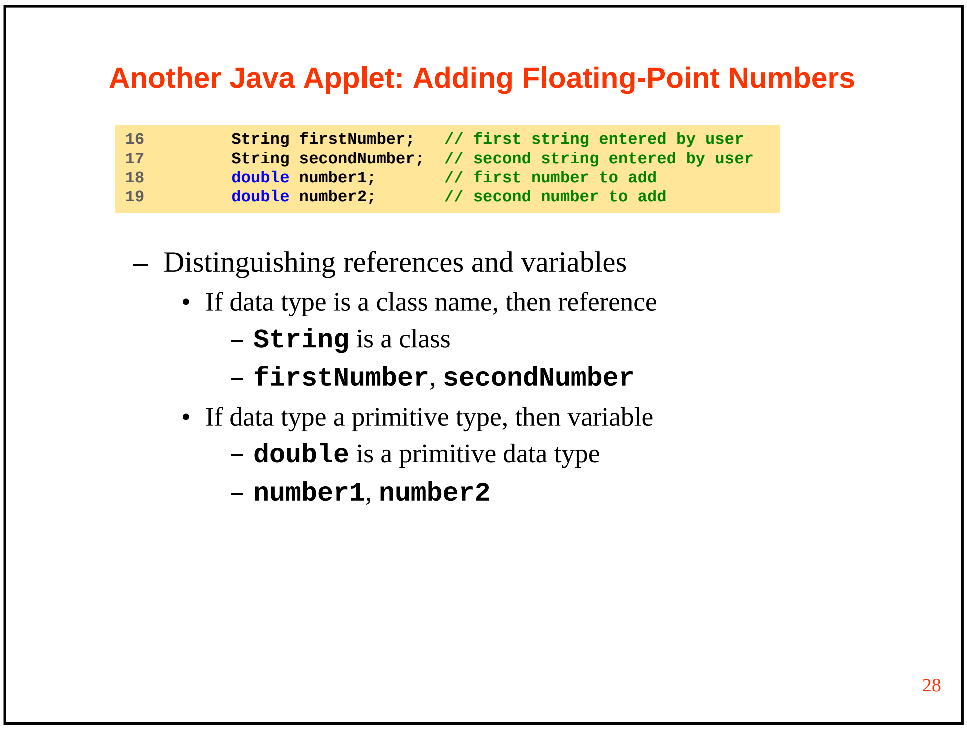
- Method **`init`**
  - Normally initializes instance variables and applet class
  - ***Guaranteed to be first method called in applet***
  - First line must always appear as above
    - Returns nothing (**`void`**), takes no arguments



Another Java Applet: Adding Floating-Point Numbers

```
      String firstNumber;   // first string entered by user
      String secondNumber;  // second string entered by user
      double number1;       // first number to add
      double 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



Another Java Applet: Adding Floating-Point Numbers

```
      String firstNumber;   // first string entered by user
      String secondNumber;  // second string entered by user
      double number1;       // first number to add
      double 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`**

Another Java Applet: Adding Floating-Point Numbers

```
22         firstNumber = JOptionPane.showInputDialog(
23            "Enter first floating-point value" );
```

- Method **`JOptionPane.showInputDialog`**
    - Prompts user for input with string
    - Enter value in text field, click **OK**
        - If not of correct type, error occurs
    - Returns string user inputs
    - Assignment statement to string
  - Lines 26-27: As above, assigns input to **`secondNumber`**

Another Java Applet: Adding Floating-Point Numbers

```
30         number1 = Double.parseDouble( firstNumber );
31         number2 = Double.parseDouble( secondNumber );
```

- **`static`** method **`Double.parseDouble`**
    - Converts **`String`** argument to a **`double`**
    - Returns the **`double`** value
    - Remember static method syntax
        - ClassName.methodName(arguments)

Another Java Applet: Adding Floating-Point Numbers

```
33    }
```

- Ends method **`init`**
  - **`appletviewer`** (or browser) calls inherited 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

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

- Method **`drawRect( x1, y1, width, height )`**
  - Draw rectangle, upper left corner (**`x1, y1`**), specified **`width`** and **`height`**
  - Line 45 draws rectangle starting at (15, 10) with a width of 270 pixels and a height of 20 pixels

31

Another Java Applet: Adding Floating-Point Numbers

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

- Sends **`drawString`** message (calls method) to **`Graphics`** object using reference **`g`**
  - **`"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
    - Non-local variable

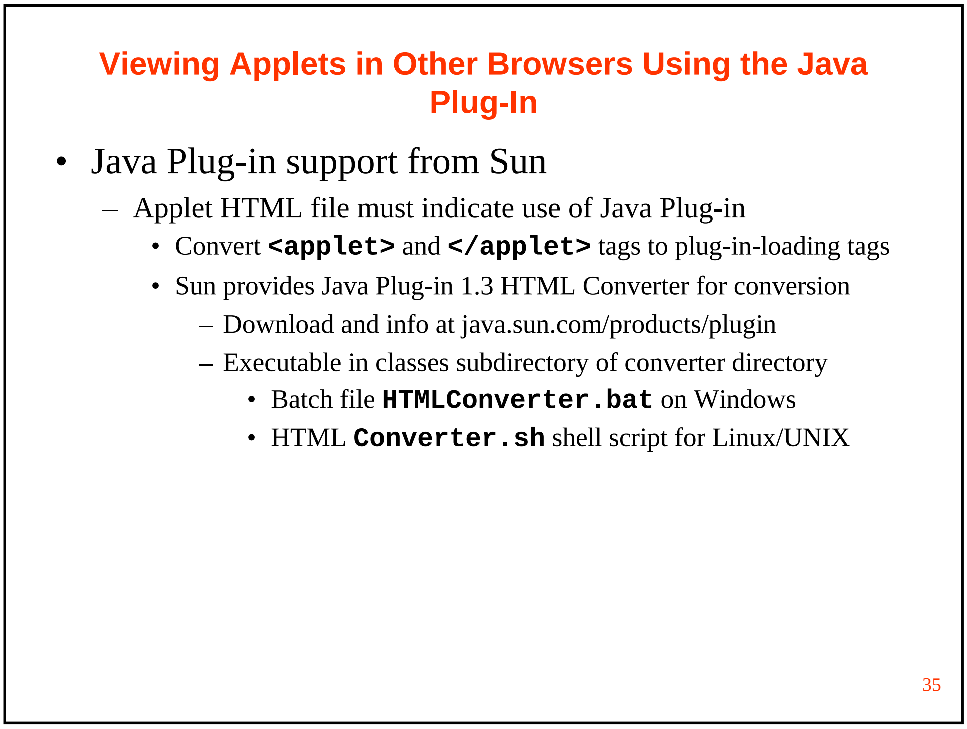
32

Viewing Applets in a Web Browser

- Applets can execute on Java-enabled browsers
  - Netscape Navigator 6 supports Java 2 (section 3.6.1)
  - Use Java Plug-in to execute Java 2 applets on other browsers (section 3.6.2)

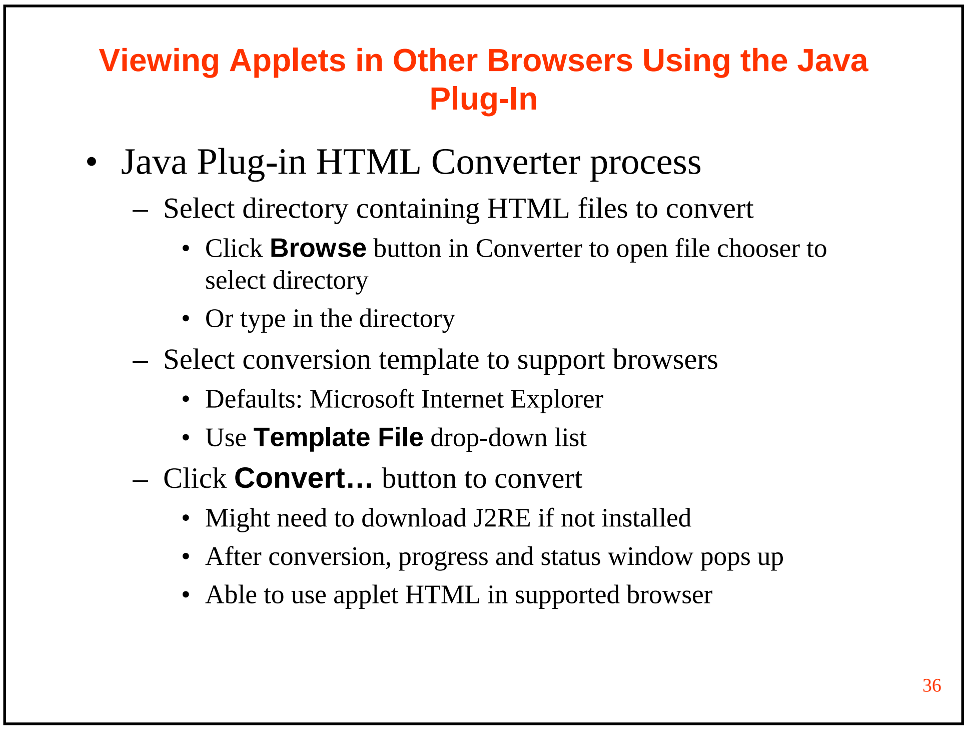
Viewing Applets in Netscape Navigator 6

- Netscape Navigator 6 supports Java 2
  - Default installation component
  - able to load applet HTML into browser and execute applet
  - Download browser at **www.netscape.com**
  - After installing, open applet HTML file using **Open File…** menu item in **File** menu



Viewing Applets in Other Browsers Using the Java Plug-In

- Java Plug-in support from Sun
  - Applet HTML file must indicate use of Java Plug-in
    - Convert **`<applet>`** and **`</applet>`** tags to plug-in-loading tags
    - Sun provides Java Plug-in 1.3 HTML Converter for conversion
      - Download and info at java.sun.com/products/plugin
      - Executable in classes subdirectory of converter directory
        - Batch file **`HTMLConverter.bat`** on Windows
        - HTML **`Converter.sh`** shell script for Linux/UNIX

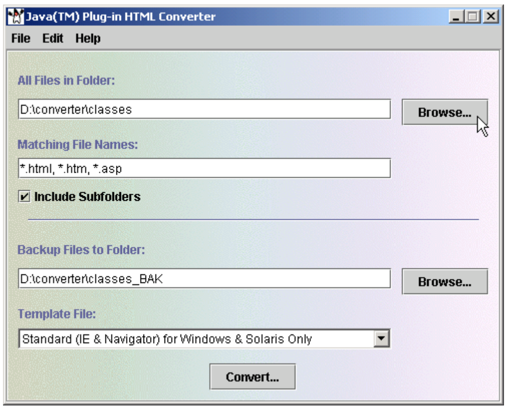


Viewing Applets in Other Browsers Using the Java Plug-In

- Java Plug-in HTML Converter process
  - Select directory containing HTML files to convert
    - Click **Browse** button in Converter to open file chooser to select directory
    - Or type in the directory
  - Select conversion template to support browsers
    - Defaults: Microsoft Internet Explorer
    - Use **Template File** drop-down list
  - Click **Convert…** button to convert
    - Might need to download J2RE if not installed
    - After conversion, progress and status window pops up
    - Able to use applet HTML in supported browser

Viewing Applets in Other Browsers Using the Java Plug-In

Fig. 3.15 Java Plug-in HTML Converter window.



All Files in Folder: D:\converter\classes

Matching File Names: *.html, *.htm, *.asp

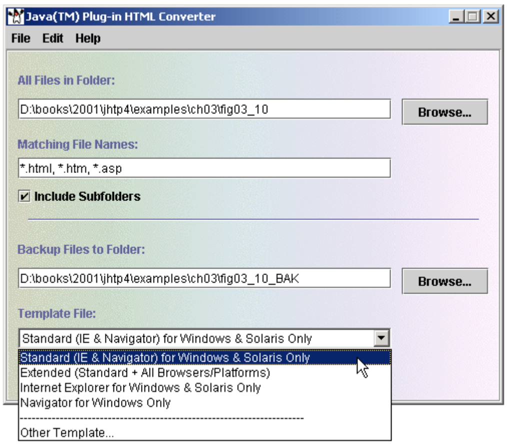
☑ Include Subfolders

Backup Files to Folder: D:\converter\classes_BAK

Template File: Standard (IE & Navigator) for Windows & Solaris Only

Viewing Applets in Other Browsers Using the Java Plug-In

Fig. 3.17 Selecting the template used to convert the HTML files.



All Files in Folder: D:\books\2001\jhtp4\examples\ch03\fig03_10

Matching File Names: *.html, *.htm, *.asp

☑ Include Subfolders

Backup Files to Folder: D:\books\2001\jhtp4\examples\ch03\fig03_10_BAK

Template File: Standard (IE & Navigator) for Windows & Solaris Only

- Standard (IE & Navigator) for Windows & Solaris Only
- Extended (Standard + All Browsers/Platforms)
- Internet Explorer for Windows & Solaris Only
- Navigator for Windows Only
- Other Template...

Java Applet Internet and World Wide Web Resources

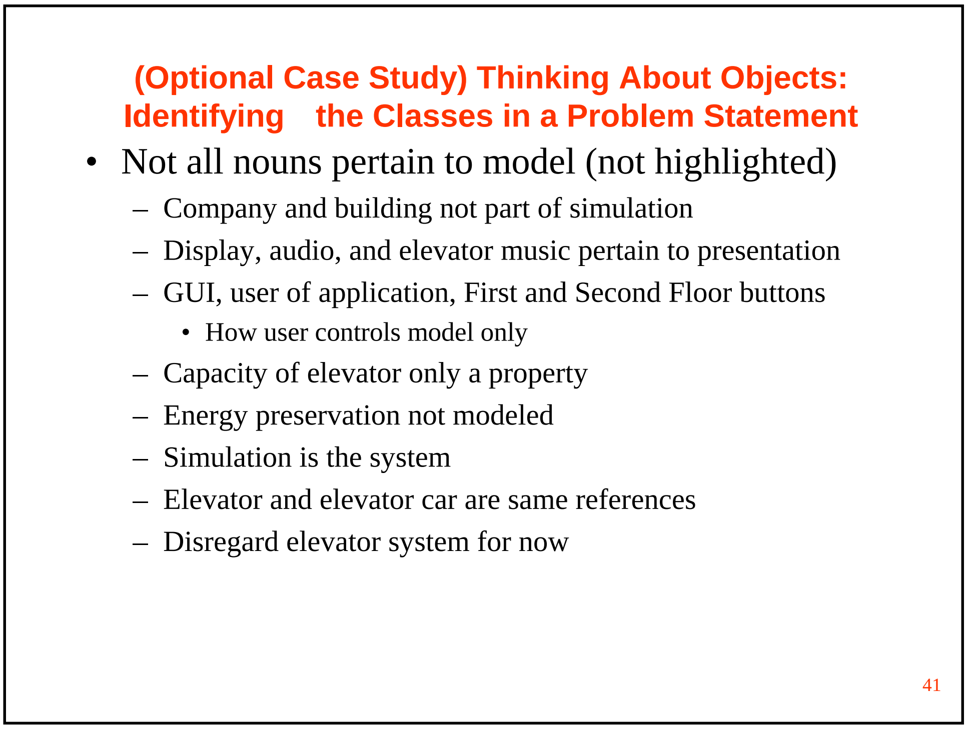
- Many Java applet resources available
  - http://java.sun.com/applets/
  - Many resources and free applets
    - Has demo applets from J2SDK
  - Sun site developer.java.sun.com/developer
    - Tech support, discussion forums, training, articles, links, etc.
    - Registration required
  - www.jars.com
    - Rates applets, top 1, 5 and 25 percent
    - View best applets on web

(Optional Case Study) Thinking About Objects: Identifying the Classes in a Problem Statement

- Identifying classes in a System
  - Nouns of system to implement elevator simulation

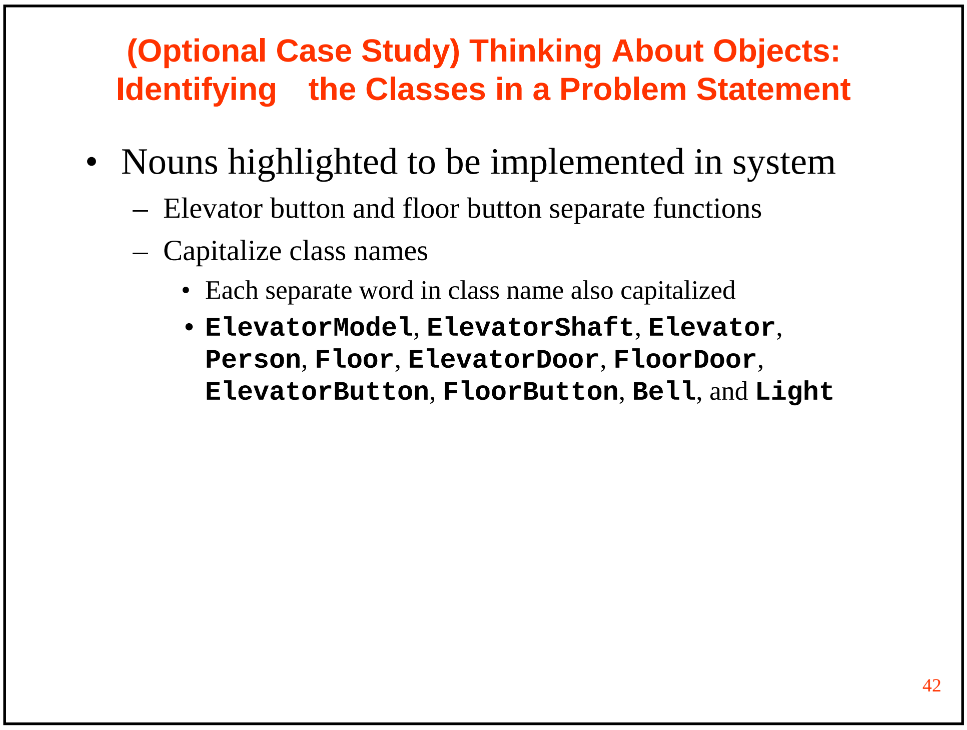
Nouns (and noun phrases) in the problem statement		
company	elevator system	graphical user interface (GUI)
office building	elevator shaft	elevator car
elevator	display	person
software-simulator application	model	floor (first floor; second floor)
passenger	bell inside the elevator	**First Floor GUI button**
floor door	light on that floor	**Second Floor** GUI button
user of our application	energy	audio
floor button	capacity	elevator music
elevator button		

Fig. 3.19 Nouns (and noun phrases) in problem statement.



(Optional Case Study) Thinking About Objects: Identifying the Classes in a Problem Statement

- Not all nouns pertain to model (not highlighted)
  - Company and building not part of simulation
  - Display, audio, and elevator music pertain to presentation
  - GUI, user of application, First and Second Floor buttons
    - How user controls model only
  - Capacity of elevator only a property
  - Energy preservation not modeled
  - Simulation is the system
  - Elevator and elevator car are same references
  - Disregard elevator system for now

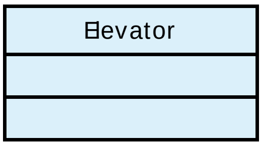


(Optional Case Study) Thinking About Objects: Identifying the Classes in a Problem Statement

- Nouns highlighted to be implemented in system
  - Elevator button and floor button separate functions
  - Capitalize class names
    - Each separate word in class name also capitalized
    - **`ElevatorModel`**, **`ElevatorShaft`**, **`Elevator`**, **`Person`**, **`Floor`**, **`ElevatorDoor`**, **`FloorDoor`**, **`ElevatorButton`**, **`FloorButton`**, **`Bell`**, and **`Light`**

(Optional Case Study) Thinking About Objects: Identifying the Classes in a Problem Statement

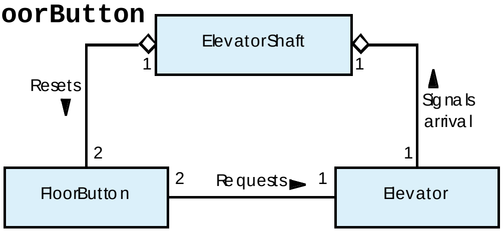
- Using UML to model elevator system
  - Class diagrams models classes and relationships
    - Model structure/building blocks of system
- Representing class **`Elevator`** using UML



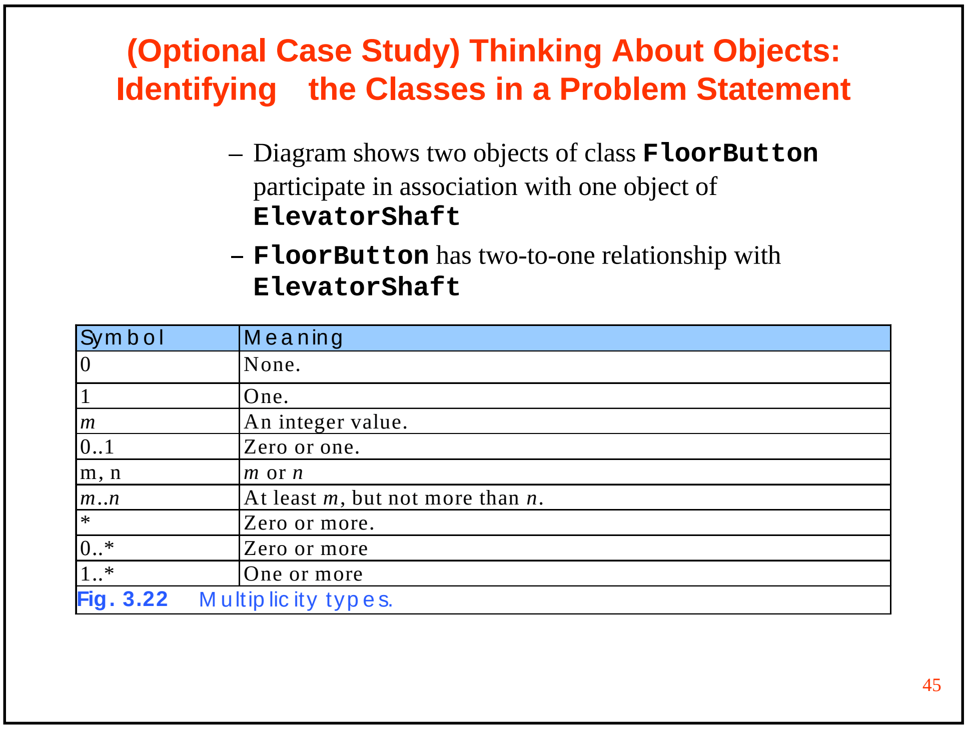
  - Top rectangle is class name
  - Middle contains class’ attributes
  - Bottom contains class’ operations

(Optional Case Study) Thinking About Objects: Identifying the Classes in a Problem Statement

- Class associations using UML
  - Elided diagram
    - Class attributes and operations ignored
    - Class relation among **`ElevatorShaft`**, **`Elevator`** and **`FloorButton`**



    - Solid line is an association, or relationship between classes
    - Numbers near lines express multiplicity values
      - Indicate how many objects of class participate association

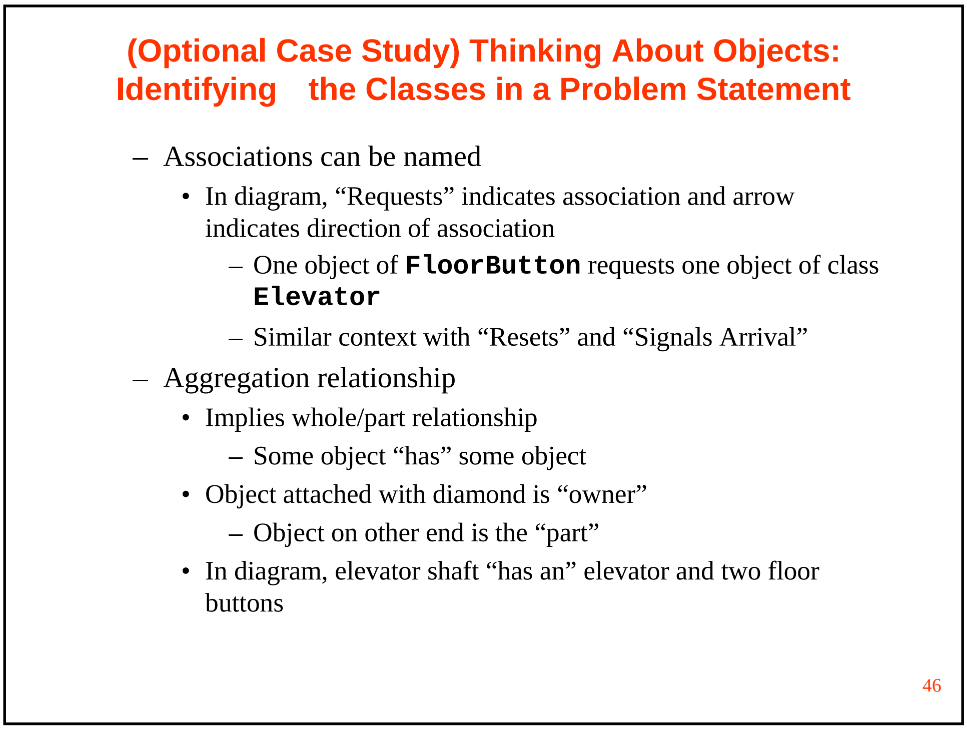


(Optional Case Study) Thinking About Objects: Identifying the Classes in a Problem Statement

- Diagram shows two objects of class **`FloorButton`** participate in association with one object of **`ElevatorShaft`**
- **`FloorButton`** has two-to-one relationship with **`ElevatorShaft`**

Symbol	Meaning
0	None.
1	One.
m	An integer value.
0..1	Zero or one.
m, n	*m* or *n*
m..n	At least *m*, but not more than *n*.
*	Zero or more.
0..*	Zero or more
1..*	One or more

Fig. 3.22 Multiplicity types.

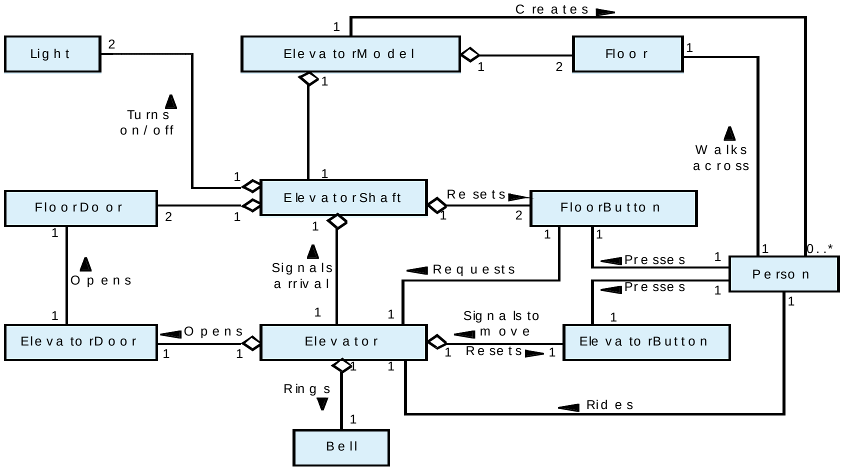


(Optional Case Study) Thinking About Objects: Identifying the Classes in a Problem Statement

- Associations can be named
  - In diagram, “Requests” indicates association and arrow indicates direction of association
    - One object of **`FloorButton`** requests one object of class **`Elevator`**
    - Similar context with “Resets” and “Signals Arrival”
- Aggregation relationship
  - Implies whole/part relationship
    - Some object “has” some object
  - Object attached with diamond is “owner”
    - Object on other end is the “part”
  - In diagram, elevator shaft “has an” elevator and two floor buttons

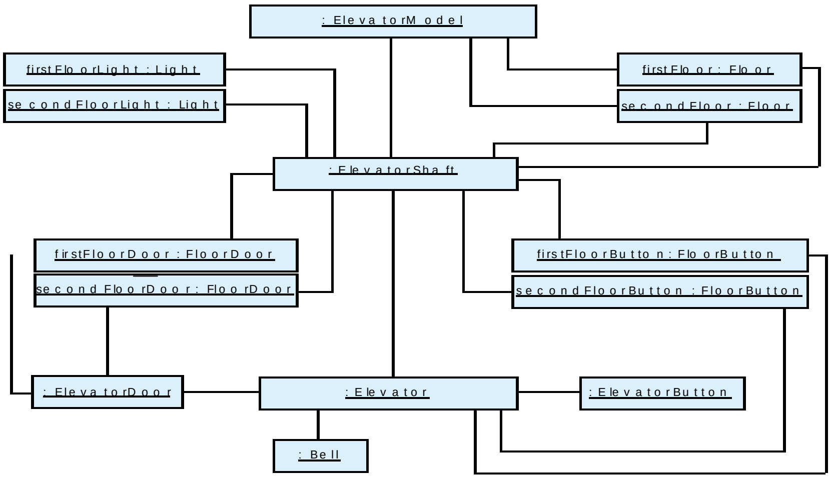
(Optional Case Study) Thinking About Objects: Identifying the Classes in a Problem Statement

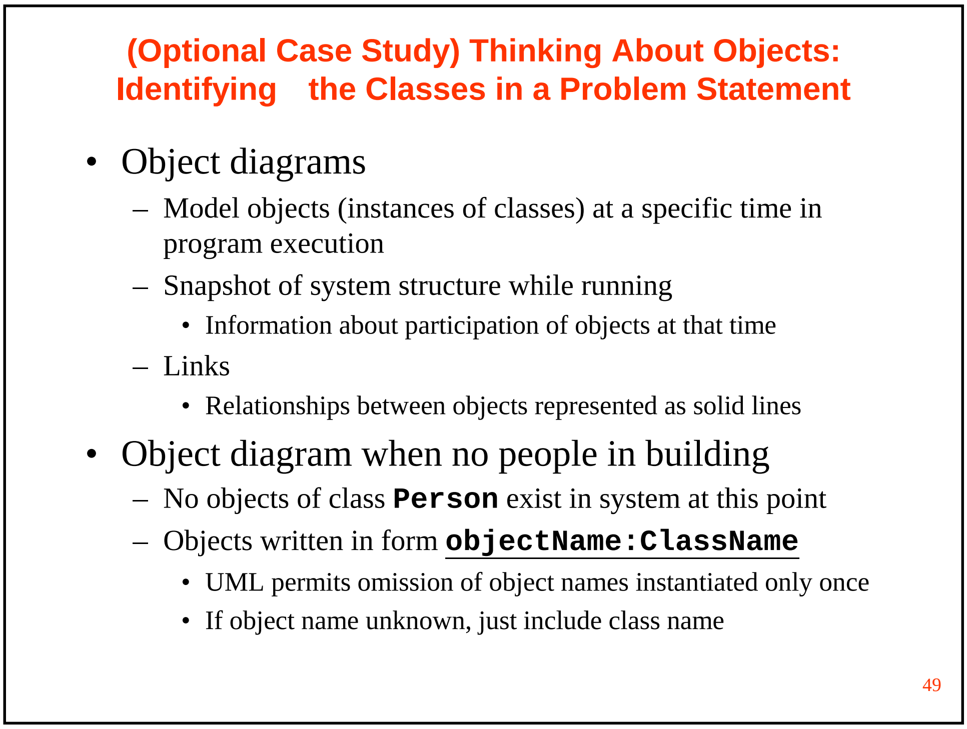
Fig. 3.23 Class diagram for the elevator model.



(Optional Case Study) Thinking About Objects: Identifying the Classes in a Problem Statement

Fig. 3.24 Object diagram of an empty building in our elevator model.





(Optional Case Study) Thinking About Objects: Identifying the Classes in a Problem Statement

- Object diagrams
  - Model objects (instances of classes) at a specific time in program execution
  - Snapshot of system structure while running
    - Information about participation of objects at that time
  - Links
    - Relationships between objects represented as solid lines
- Object diagram when no people in building
  - No objects of class **`Person`** exist in system at this point
  - Objects written in form <u>**`objectName:ClassName`**</u>
    - UML permits omission of object names instantiated only once
    - If object name unknown, just include class name