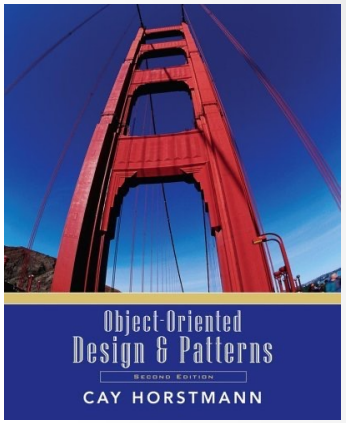
Object-Oriented Design & Patterns

2nd edition

Cay S. Horstmann



Chapter 4: Interface Types and Polymorphism

CPSC 2100

Software Design and Development

Chapter Topics

- Displaying an Image
- Polymorphism
- The Comparable Interface
- The Comparator Interface
- Anonymous Classes
- Frames and User Interface Components
- User Interface Actions
- Timers
- Drawing Shapes
- Designing an Interface

Chapter Objective

- Define a set of operations (the interface) and statements that specify how to carry out the operations and how to represent object state (the implementation).
- Separate the interface concept from that of a class can help in the development of reusable code.
- Focusing on interface types first, you will study polymorphism in its purist and simplest form.

Displaying an Image

- Use `JOptionPane` to display message:

```
JOptionPane.showMessageDialog(null, "Hello, World!");
```

- Note icon to the left

http://docs.oracle.com/javase/tutorial/uiswing/components/dialog.html

Displaying an Image

- Can specify arbitrary image file

```
JOptionPane.showMessageDialog(
null, // parent window
"Hello, World!", // message
"Message", // window title
JOptionPane.INFORMATION_MESSAGE, // message type
new ImageIcon("globe.gif"));
```

Displaying an Image

- What if we don't want to generate an image *file*?
- Fortunately, can use any class that *implements* Icon *interface type*.
- ImageIcon is one such class
- Easy to supply your own class.

```
public interface Icon
{
   int getIconWidth();
   int getIconHeight();
   voif paintIcon(Component c, Graphics g, int x, int y);
}
```

Interface Types

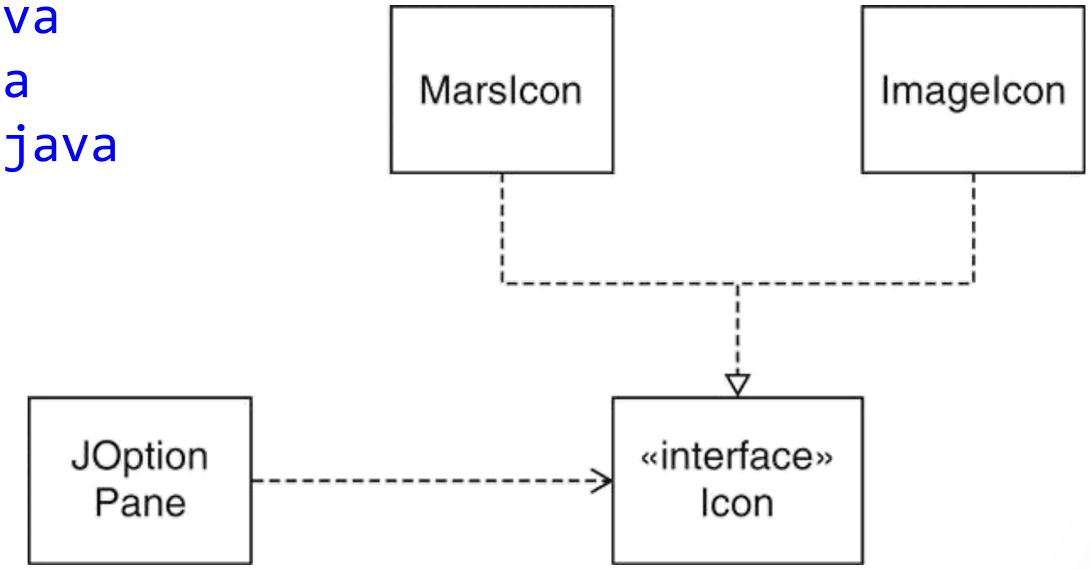
Technical remarks:

1. No implementation.
2. All methods of an interface are automatically public.
3. Implementing class must supply implementation of all methods.

- showMessageDialog expects Icon object.
- Ok to pass MarsIcon.

The Icon Interface Type and Implementing Classes

MarsIcon.java
CarIcon.java
IconTester.java



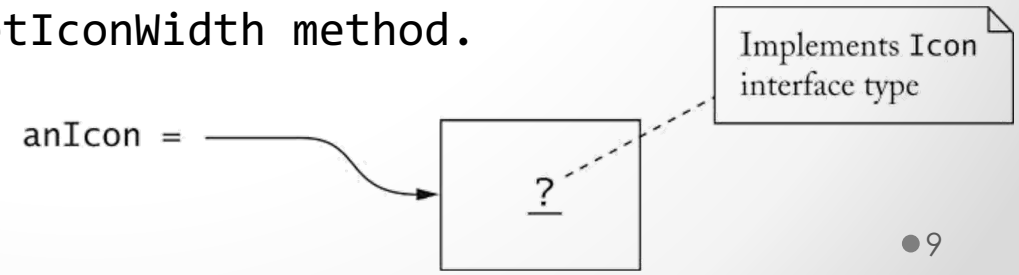
Polymorphism

- `showMessageDialog` doesn't know *which* icon is passed
  - ImageIcon?
  - MarsIcon?
  - . . .?

- Compute the size of the dialog box:

  width = iconWidth + message width + blank separation space

  - ✓ The actual type of **anIcon** is *not* **Icon**.
  - ✓ There are no objects of type Icon.
  - ✓ anIcon belongs to a *class* that implements Icon.
  - ✓ That class defines a getIconWidth method.



Polymorphism

- Which `getIconWidth` method is called?
- Could be
  - `MarsIcon.getIconWidth`
  - `ImageIcon.getIconWidth`
  - `. . .`

- Depends on object to which anIcon reference points, e.g.

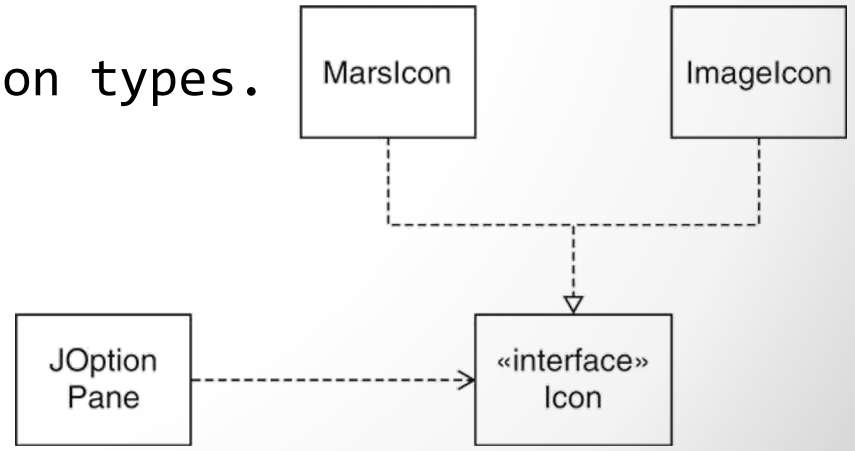
```
showMessageDialog(..., new MarsIcon(50))
```

Polymorphism: Select different methods according to actual object type.

Benefits of Polymorphism

The ability to select appropriate method for a particular object is called ***polymorphism***.

- **Loose coupling**
  - `showMessageDialog` decoupled from `ImageIcon`.
  - Doesn't need to know about image processing.
- **Extensibility**
  - Client can supply new icon types.



The **Comparable** Interface Type

- **Collections class** has static sort method:

```
ArrayList<E> a = . . .
Collections.sort(a);
```

- Objects in array list must implement the **Comparable interface type.**

```
public interface Comparable<T>
{
   int compareTo(T other);
}
```

The **Comparable** Interface Type

- `object1.compareTo(object2)` returns
  - Negative number if object1 less than object2.
  - 0 if objects identical.
  - Positive number if object1 greater than object2.

- Why implement Comparable interface type?
  - sort method compares and rearranges elements
    `if (object1.compareTo(object2) > 0) . . .`

- Country class: compare countries by area

Country.java

CountrySortTester.java

The **Comparator** interface type

- How can we sort countries by name?
- Can't implement Comparable twice!
- ***Comparator* interface** type gives added flexibility.

```
public interface Comparator<T>
{
    int compare(T obj1, T obj2);
}
```

- Pass comparator object to sort:
  `Collections.sort(list, comp);`

The **Comparator** interface type

Country.java
CountryComparatorByName.java
ComparatorTester.java

- Comparator object is a *function object*
- This particular comparator object has no state.

The **Comparator** interface type

- State can be useful, e.g. flag to sort in ascending or descending order.

```
public class CountryComparator implements Comparator <Country>
{
     public CountryComparator(boolean ascending)
     {
          if (ascending)
               direction = 1;
          else
               direction = -1;
     }
     public int compare(Country country1, Country country2)
     {
               return direction *
                    country1.getName().compareTo(country2.getName());
     }
     private int direction;

}
```

Anonymous Classes

- No need to name objects that are used only once

```
Collections.sort(countries,
        new CountryComparatorByName());
```

- No need to name classes that are used only once

```
Comparator<Country> comp = new Comparator<Country>()
{
   public int compare(Country country1, Country country2)
   {
      return country1.getName().compareTo(country2.getName());
   }
};
```

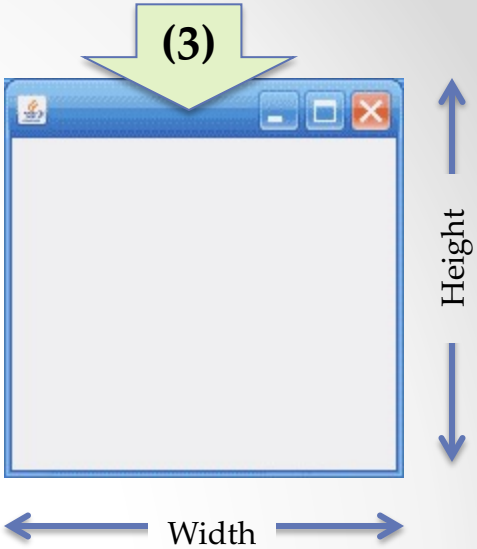
Anonymous Classes

- Commonly used in factory methods:

```
public static Comparator<Country> comparatorByName()
{
   return new Comparator<Country>()
   {
      public int compare(Country country1, Country country2
      { . . . }
   };
}
Collections.sort(a, Country.comparatorByName());
```

- Neat arrangement if multiple comparators make sense (by name, by area, ...)

Frames



1. Construct an object of the JFrame class.

```
JFrame aFrame = new JFrame();
```

2. Set the size of the frame.

```
frame.setSize(with, height);
```

3. Set the title of the frame.

```
frame.setTitle(". . . . . .");
```

4. Set the default close operation.

```
frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
```

5. Make the frame visible.

```
frame.setVisible(true);
```

Frames

- Construct components

```
JButton helloButton = new JButton("Say Hello");
```

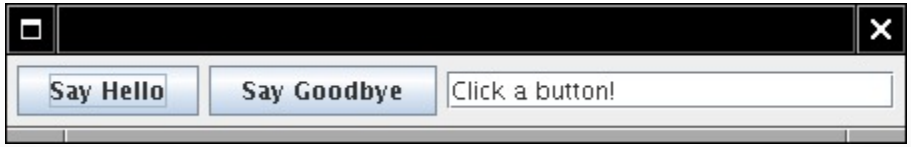
- Set frame layout

```
frame.setLayout(new FlowLayout());
```

- Add components to frame

```
frame.add(helloButton);
```

FrameTester.java



User Interface Actions

- Previous program's buttons don't have any effect.
- Add *listener object(s)* to button.
- Belong to class implementing **ActionListener interface type**.

```
public interface ActionListener
{
    void actionPerformed(ActionEvent event);
}
```

- Listeners are notified when button is clicked.

User Interface Actions

- Add action code into `actionPerformed` method
- Gloss over routine code

```
helloButton.addActionListener(new ActionListener()
{
    public void actionPerformed(ActionEvent event)
    {
          textField.setText("Hello, World");
    }
});
```

- When button is clicked, text field is set.

User Interface Actions

FrameTester.java

- Constructor attaches listener:
  `helloButton.addActionListener(listener);`

- Button remembers all listeners.
- When button clicked, button notifies listeners
  `listener.actionPerformed(event);`

- Listener sets text of text field
  `textField.setText("Hello, World!");`

Accessing Variables from Enclosing Scope

- **Remarkable:** Inner class can access variables from enclosing scope.
  e.g. textField

- Can access enclosing instance fields, local variables.

- **Important:** Local variables must be marked final.
  ```
  final JTextField textField = ...;
  ```

Timers

- Supply delay, action listener

```
ActionListener listener = ...;
final int DELAY = 1000; // 1000 millisec = 1 sec
Timer t = new Timer(DELAY, listener);
t.start();
```

- Action listener called when delay elapsed.

FrameTester.java

Drawing Shapes

- `paintIcon` method receives graphics context of type Graphics.
- Actually a Graphics2D object in modern Java versions

```
public void paintIcon(Component c, Graphics g, int x, int y)
{
      Graphics2D g2 = (Graphics2D)g;
      . . .
}
```

Can draw any object that implements Shape interface

```
Shape s = . . .;
g2.draw(s);
```

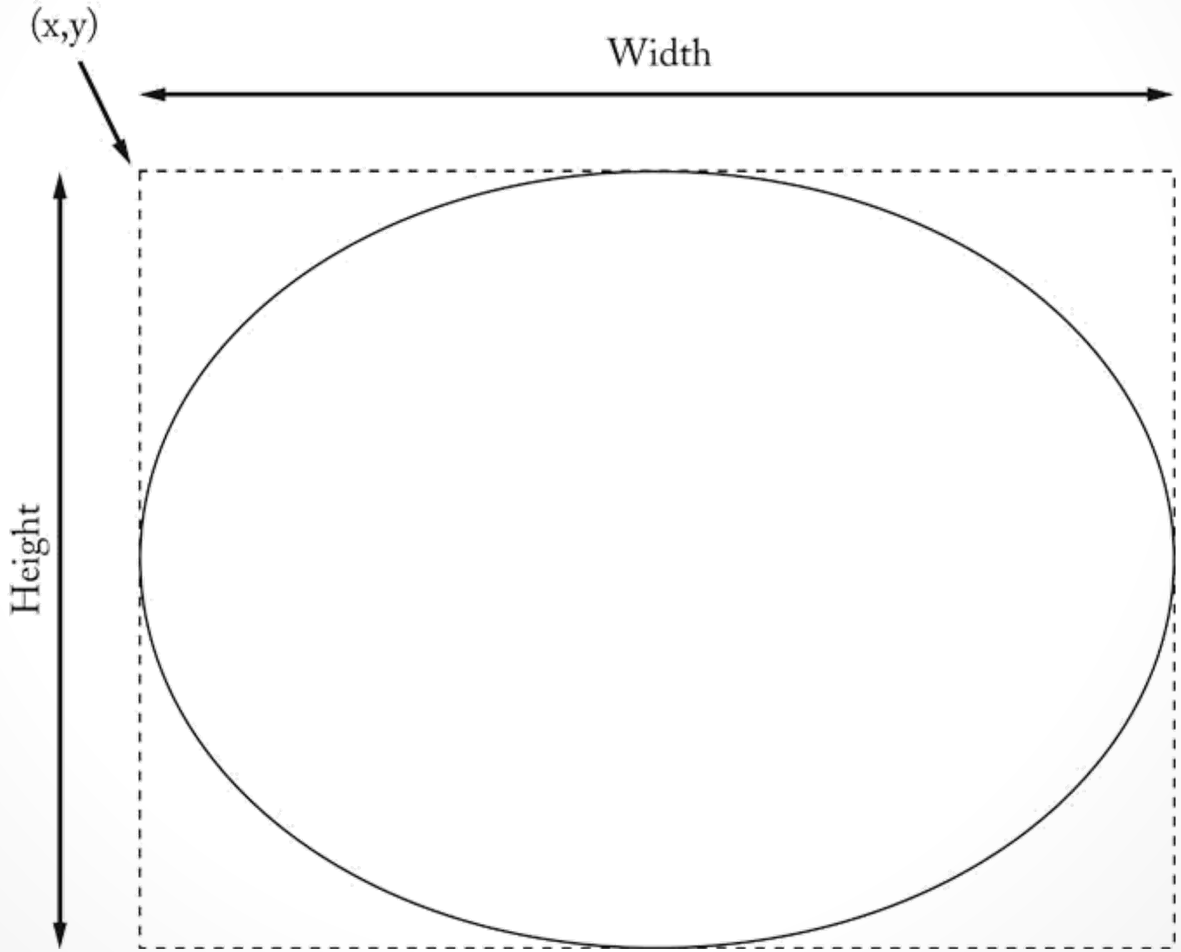
Drawing Rectangles and Ellipses

- `Rectangle2D.Double` constructed with
  - top left corner
  - width
  - height

```
g2.draw(new Rectangle2D.Double(x, y, width, height));
```

- For `Ellipse2D.Double`, specify bounding box.

Drawing Ellipses

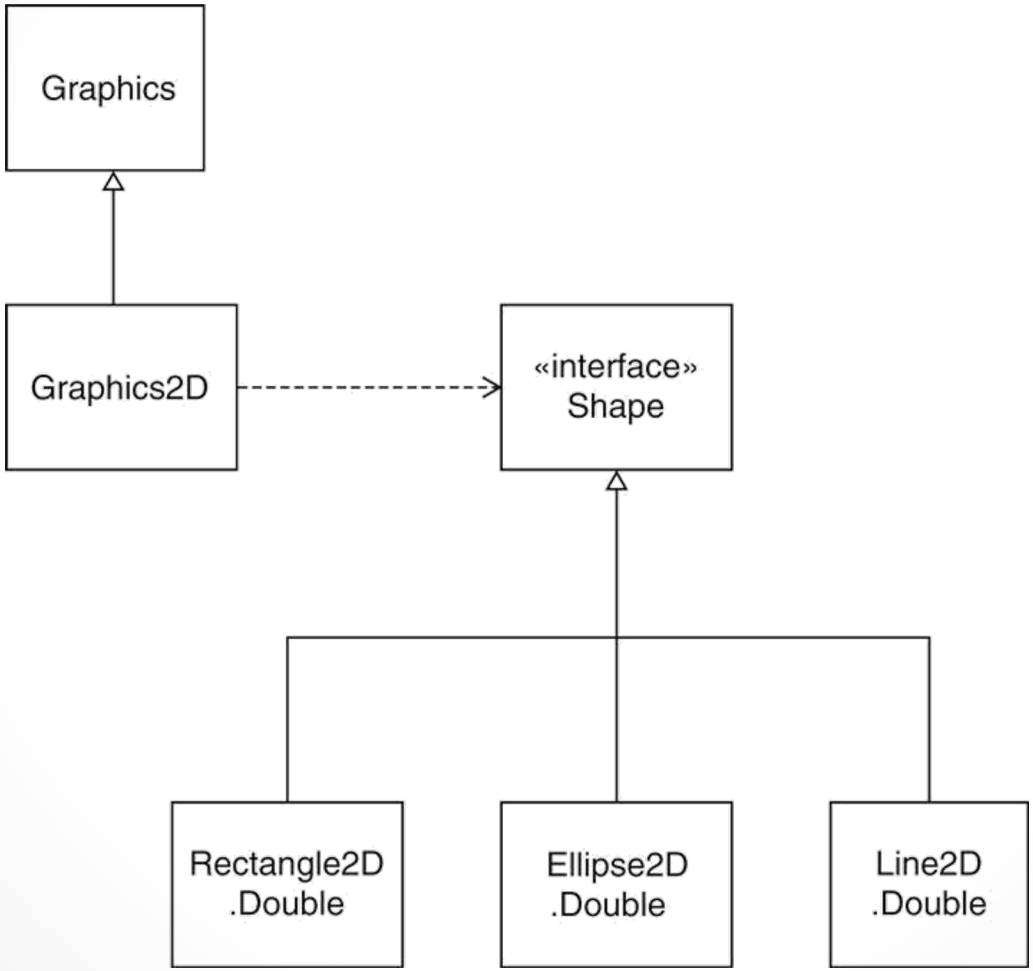


Drawing Line Segments

- `Point2D.Double` is a point in the plane
- `Line2D.Double` joins to points

```
Point2D.Double start = new Point2D.Double(x1, y1);
Point2D.Double end = new Point2D.Double(x2, y2);
Shape segment = new Line2D.Double(start, end);
g2.draw(segment);
```

Relationship Between Shape Classes



Drawing Text

- `g2.drawString(text, x, y);`
- `x, y` are base point coordinates

Drawing Cars

- Draw two cars: one in top-left corner of window, and another in the bottom right
- Compute bottom right position, inside paintComponent method:

```
int x = getWidth() - 60;

int y = getHeight() - 30;

Car car2 = new Car(x, y);
```

- **getWidth** and **getHeight** are applied to object that executes paintComponent

- If window is resized paintComponent is called and car position recomputed.

Drawing Cars (cont.)

Figure 7
The Car Component Draws Two Car Shapes

Plan Complex Shapes on Graph Paper

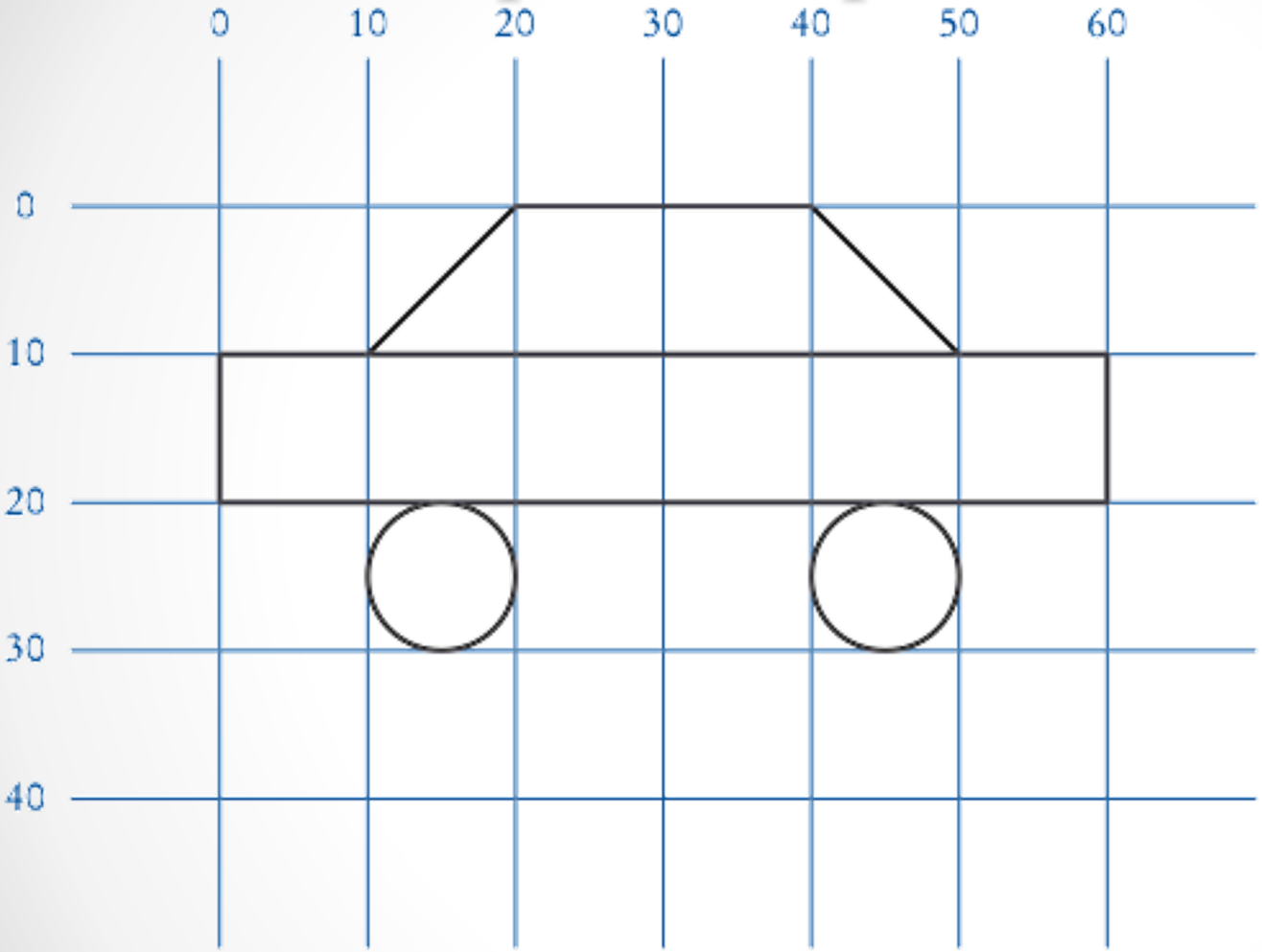


Figure 8 Using Graph Paper to Find Shape Coordinates

Classes of Car Drawing Program

- **Car:** responsible for drawing a single car
  - *Two objects of this class are constructed, one for each car*
- **CarComponent:** displays the drawing
- **CarViewer:** shows a frame that contains a CarComponent

ch03/car/Car.java

```
1  import java.awt.Graphics2D;
2  import java.awt.Rectangle;
3  import java.awt.geom.Ellipse2D;
4  import java.awt.geom.Line2D;
5  import java.awt.geom.Point2D;
6
7  /**
8     A car shape that can be positioned anywhere on the screen.
9  */
10 public class Car
11 {
12    private int xLeft;
13    private int yTop;
14
15    /**
16       Constructs a car with a given top left corner.
17       @param x the x coordinate of the top left corner
18       @param y the y coordinate of the top left corner
19    */
20    public Car(int x, int y)
21    {
22       xLeft = x;
23       yTop = y;
24    }
```

Continued

ch03/car/Car.java (cont.)

```
25
26      /**
27         Draws the car.
28         @param g2 the graphics context
29      */
30      public void draw(Graphics2D g2)
31      {
32         Rectangle body
33               = new Rectangle(xLeft, yTop + 10, 60, 10);
34         Ellipse2D.Double frontTire
35               = new Ellipse2D.Double(xLeft + 10, yTop + 20, 10, 10);
36         Ellipse2D.Double rearTire
37               = new Ellipse2D.Double(xLeft + 40, yTop + 20, 10, 10);
38
39         // The bottom of the front windshield
40         Point2D.Double r1
41               = new Point2D.Double(xLeft + 10, yTop + 10);
42         // The front of the roof
43         Point2D.Double r2
44               = new Point2D.Double(xLeft + 20, yTop);
45         // The rear of the roof
46         Point2D.Double r3
47               = new Point2D.Double(xLeft + 40, yTop);
```

Continued

ch03/car/Car.java (cont.)

```
48          // The bottom of the rear windshield
49          Point2D.Double r4
50                = new Point2D.Double(xLeft + 50, yTop + 10);
51
52          Line2D.Double frontWindshield
53                = new Line2D.Double(r1, r2);
54          Line2D.Double roofTop
55                = new Line2D.Double(r2, r3);
56          Line2D.Double rearWindshield
57                = new Line2D.Double(r3, r4);
58
59          g2.draw(body);
60          g2.draw(frontTire);
61          g2.draw(rearTire);
62          g2.draw(frontWindshield);
63          g2.draw(roofTop);
64          g2.draw(rearWindshield);
65       }
66    }
```

ch03/car/CarViewer.java

```
 1  import javax.swing.JFrame;
 2
 3  public class CarViewer
 4  {
 5     public static void main(String[] args)
 6     {
 7        JFrame frame = new JFrame();
 8
 9        frame.setSize(300, 400);
10        frame.setTitle("Two cars");
11        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
12
13        CarComponent component = new CarComponent();
14        frame.add(component);
15
16        frame.setVisible(true);
17     }
18  }
```

ch03/car/CarComponent.java

```
1  import java.awt.Graphics;
2  import java.awt.Graphics2D;
3  import javax.swing.JComponent;
4
5  /**
6     This component draws two car shapes.
7  */
8  public class CarComponent extends JComponent
9  {
10    public void paintComponent(Graphics g)
11    {
12       Graphics2D g2 = (Graphics2D) g;
13
14       Car car1 = new Car(0, 0);
15
16       int x = getWidth() - 60;
17       int y = getHeight() - 30;
18
19       Car car2 = new Car(x, y);
20
21       car1.draw(g2);
22       car2.draw(g2);
23    }
24 }
```

Filling Shapes

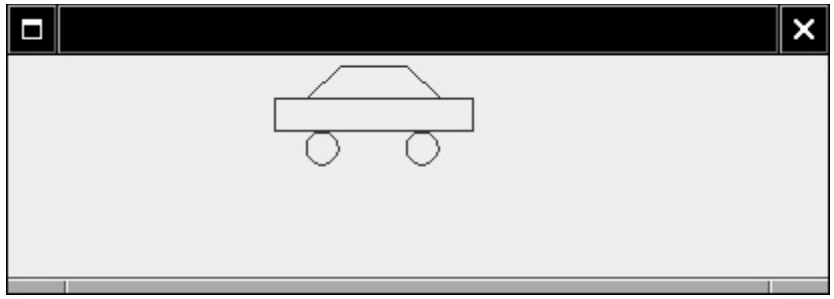
- Fill interior of shape
  `g2.fill(shape);`
- Set color for fills or strokes:
  `g2.setColor(Color.red);`

- Program that draws car

CarIcon.java
IconTester.java

Defining a New Interface Type

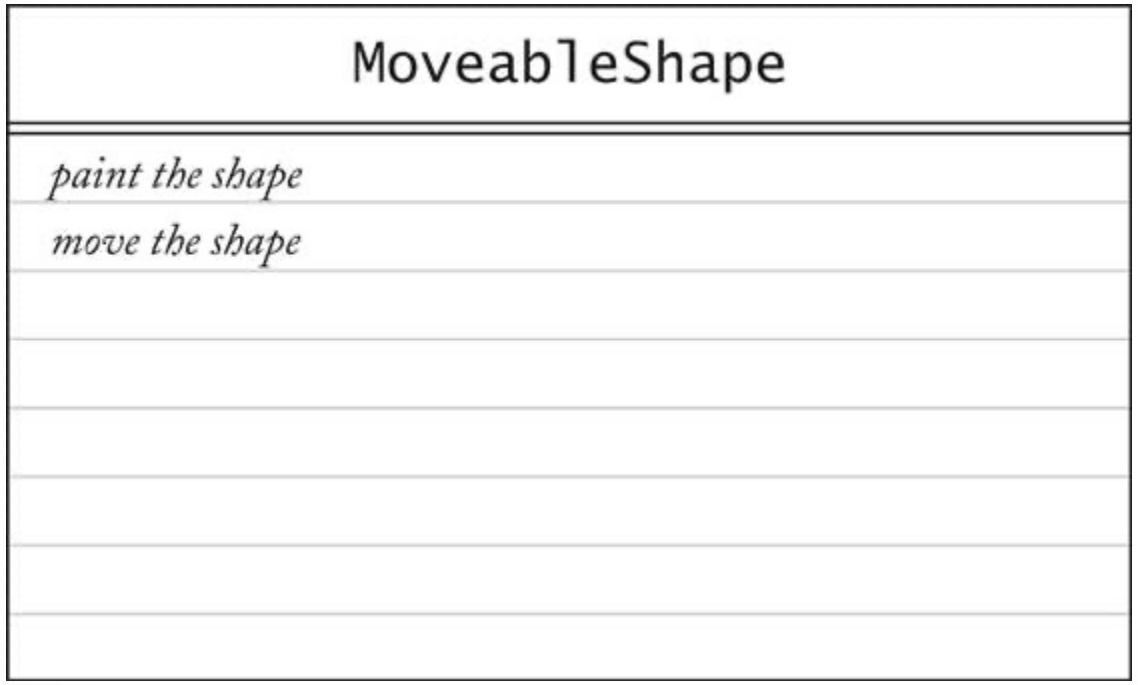
- Use timer to move car shapes.
- Draw car with `CarShape`.
- Two responsibilities:
  - Draw shape
  - Move shape
- Define new interface type `MoveableShape`



Implementing the Animation

1. Label contains icon that draws shape.
2. Timer action moves shape, calls repaint on label.
3. Label needs Icon, we have `MoveableShape`.
4. Supply `ShapeIcon` adapter class.
5. `ShapeIcon.paintIcon` calls `MoveableShape.draw`.

CRC Card for the MoveableShape Interface Type



Defining a New Interface Type

- Name the methods to conform to standard library.

```
public interface MoveableShape
{
   void draw(Graphics2D g2);
   void translate(int dx, int dy);
}
```

- CarShape class implements MoveableShape

```
public class CarShape implements MoveableShape
{
   public void translate(int dx, int dy)
   {
      x += dx;
      y += dy;
    }
    . . .
}
```

Implementing the Animation

MoveableShape.java

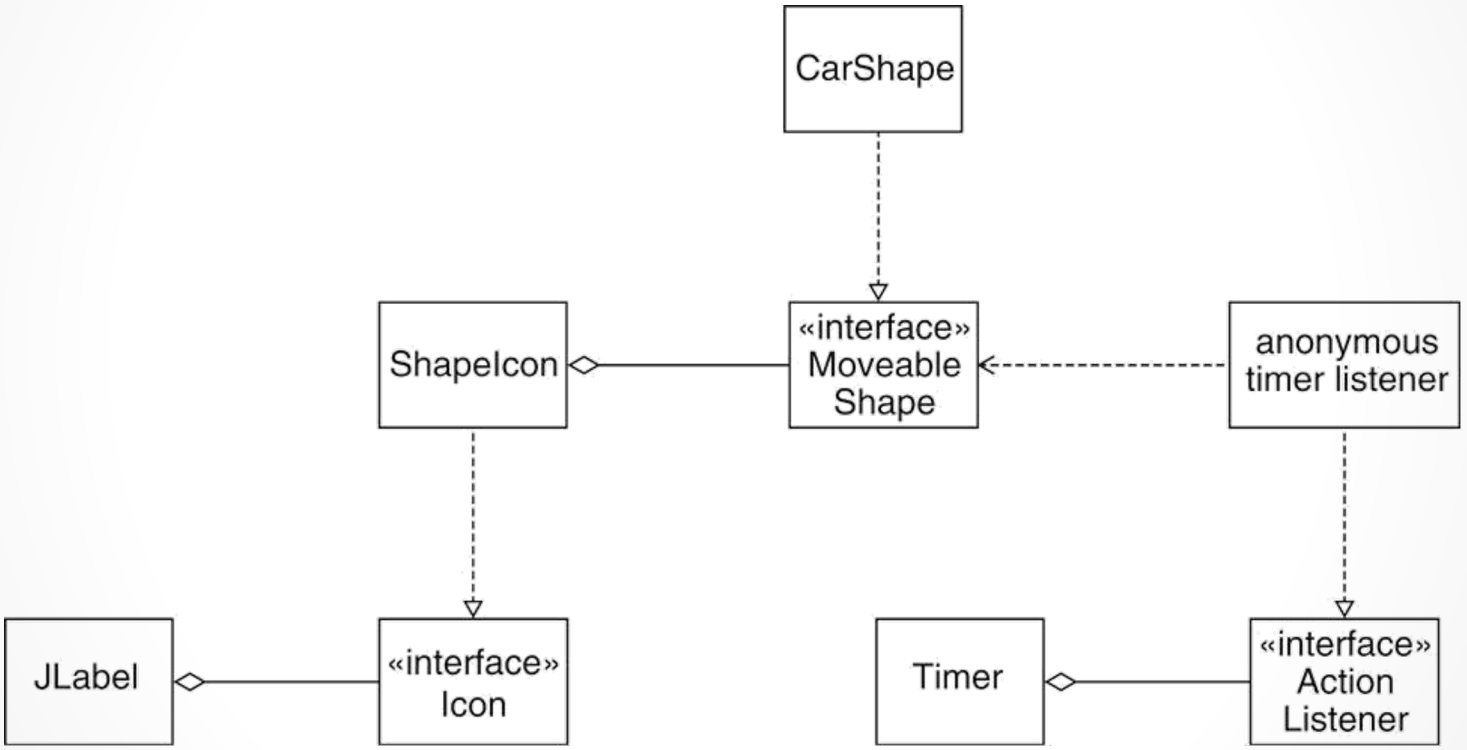
ShapeIcon.java

AnimationTester.java

CarShape.java

BusShape.java

Implementing the Animation



End of Chapter 4