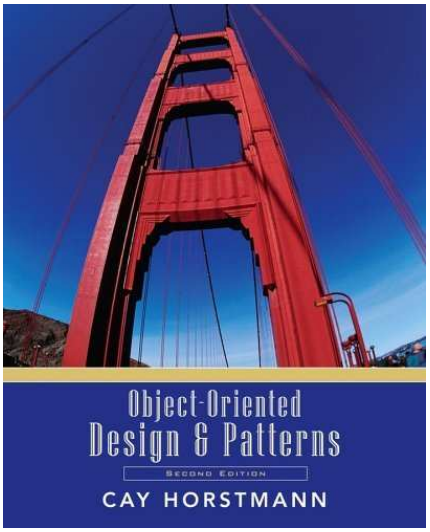
Object-Oriented Design & Patterns

Cay S. Horstmann

Chapter 4

Interfaces and Polymorphism



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

Displaying an Image

- Use `JOptionPane` to display message:

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

- Note icon to the left

Displaying an Image

- Can specify arbitrary image file

```
JOptionPane.showMessageDialog( null,  "Hello, World!", "Message", JOptionPane.INFORMATION_MESSAGE, 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

The `Icon` Interface Type

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

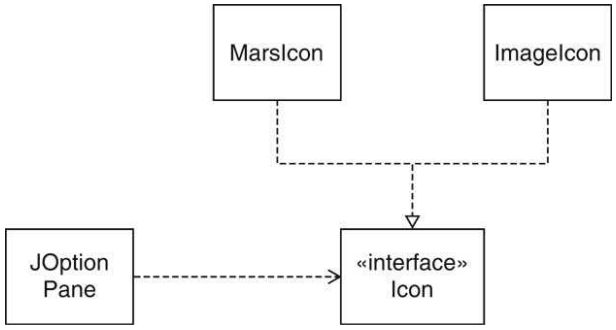

Interface Types

- No implementation
- Implementing class must supply implementation of all methods
- Ch4/icon2/MarsIcon.java
- `showMessageDialog` expects `Icon` object
- Ok to pass `MarsIcon`
- Ch4/icon2/IconTester.java

```
01: import java.awt.*;
02: import java.awt.geom.*;
03: import javax.swing.*;
04:
05: /**
06:    An icon that has the shape of the planet Mars.
07: */
08: public class MarsIcon implements Icon
09: {
10:    /**
11:       Constructs a Mars icon of a given size.
12:       @param aSize the size of the icon
13:    */
14:    public MarsIcon(int aSize)
15:    {
16:       size = aSize;
17:    }
18:
19:    public int getIconWidth()
20:    {
21:       return size;
22:    }
23:
24:    public int getIconHeight()
25:    {
26:       return size;
27:    }
28:
29:    public void paintIcon(Component c, Graphics g, int x, int y)
30:    {
31:       Graphics2D g2 = (Graphics2D) g;
32:       Ellipse2D.Double planet = new Ellipse2D.Double(x, y,
33:             size, size);
34:       g2.setColor(Color.RED);
35:       g2.fill(planet);
36:    }
37:
38:    private int size;
39: }
```

```
01: import javax.swing.*;
02:
03: public class IconTester
04: {
05:    public static void main(String[] args)
06:    {
07:       JOptionPane.showMessageDialog(
08:             null,
09:             "Hello, Mars!",
10:             "Message",
11:             JOptionPane.INFORMATION_MESSAGE,
12:             new MarsIcon(50));
13:       System.exit(0);
14:    }
15: }
16:
```


The `Icon` Interface Type and Implementing Classes



Polymorphism

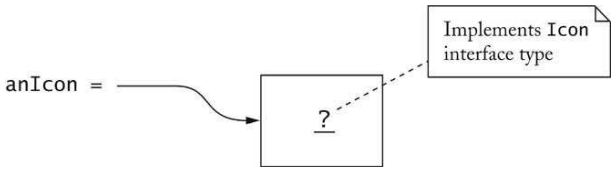
- `public static void showMessageDialog(...`**`Icon anIcon`**`)`
- `showMessageDialog` shows
  - icon
  - message
  - OK button
- `showMessageDialog` must compute size of dialog
- width = icon width + message size + blank size
- How do we know the icon width?

```
int width = anIcon.getIconWidth();
```

Polymorphism

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

A Variable of Interface Type



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

- 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` has static `sort` method:

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

- Objects in list must implement the `Comparable` interface type

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

- Interface is parameterized (like `ArrayList`)
- Type parameter is type of `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`
- `sort` method compares and rearranges elements
  `if (object1.compareTo(object2) > 0) . . .`
- `String` class implements `Comparable<String>` interface type: lexicographic (dictionary) order
- `Country` class: compare countries by area
  Ch4/sort1/Country.java
  Ch4/sort1/CountrySortTester.java


```
01: /**
02:    A country with a name and area.
03: */
04: public class Country implements Comparable<Country>
05: {
06:    /**
07:       Constructs a country.
08:       @param aName the name of the country
09:       @param anArea the area of the country
10:    */
11:    public Country(String aName, double anArea)
12:    {
13:       name = aName;
14:       area = anArea;
15:    }
16:
17:    /**
18:       Gets the name of the country.
19:       @return the name
20:    */
21:    public String getName()
22:    {
23:       return name;
24:    }
25:
26:    /**
27:       Gets the area of the country.
28:       @return the area
29:    */
30:    public double getArea()
31:    {
32:       return area;
33:    }
34:
35:
36:    /**
37:       Compares two countries by area.
38:       @param other the other country
39:       @return a negative number if this country has a smaller
40:       area than otherCountry, 0 if the areas are the same,
41:       a positive number otherwise
42:    */
43:    public int compareTo(Country other)
44:    {
```

```
45:       if (area < other.area) return -1;
46:       if (area > other.area) return 1;
47:       return 0;
48:    }
49:
50:    private String name;
51:    private double area;
52: }
```

```
01: import java.util.*;
02:
03: public class CountrySortTester
04: {
05:    public static void main(String[] args)
06:    {
07:       ArrayList<Country> countries = new ArrayList<Country>();
08:       countries.add(new Country("Uruguay", 176220));
09:       countries.add(new Country("Thailand", 514000));
10:       countries.add(new Country("Belgium", 30510));
11:
12:       Collections.sort(countries);
13:       // Now the array list is sorted by area
14:       for (Country c : countries)
15:          System.out.println(c.getName() + " " + c.getArea());
16:    }
17: }
```


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

- Ch4/sort2/CountryComparatorByName.java
  Ch4/sort2/ComparatorTester.java
- Comparator object is a *function object*
- This particular comparator object has no state
- State can be useful, e.g. flag to sort in ascending or descending order


```
01: import java.util.*;
02:
03: public class CountryComparatorByName implements Comparator<Country>
04: {
05:    public int compare(Country country1, Country country2)
06:    {
07:       return country1.getName().compareTo(country2.getName());
08:    }
09:
10: }
```

```
01: import java.util.*;
02:
03: public class ComparatorTester
04: {
05:    public static void main(String[] args)
06:    {
07:       ArrayList<Country> countries = new ArrayList<Country>();
08:       countries.add(new Country("Uruguay", 176220));
09:       countries.add(new Country("Thailand", 514000));
10:       countries.add(new Country("Belgium", 30510));
11:       Comparator<Country>  comp = new CountryComparatorByName();
12:       Collections.sort(countries, comp);
13:          // Now the array list is sorted by country name
14:       for (Country c : countries)
15:         System.out.println(c.getName() + " " + c.getArea());
16:    }
17: }
18:
```


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

- anonymous **new** expression:
  - defines anonymous class that implements `Comparator`
  - defines `compare` method of that class
  - constructs one object of that class
- Cryptic syntax for very useful feature

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

- Frame window has *decorations*
  - title bar
  - close box
  - provided by windowing system
-

```
JFrame frame = new JFrame();frame.pack();frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);frame.setVisible(true);
```


Adding Components

- Construct components

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

- Set frame layout

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

- Add components to frame

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

- Ch4/frame/FrameTester.java


```
01: import java.awt.*;
02: import javax.swing.*;
03:
04: public class FrameTester
05: {
06:    public static void main(String[] args)
07:    {
08:       JFrame frame = new JFrame();
09:
10:       JButton helloButton = new JButton("Say Hello");
11:       JButton goodbyeButton = new JButton("Say Goodbye");
12:
13:       final int FIELD_WIDTH = 20;
14:       JTextField textField = new JTextField(FIELD_WIDTH);
15:       textField.setText("Click a button!");
16:
17:       frame.setLayout(new FlowLayout());
18:
19:       frame.add(helloButton);
20:       frame.add(goodbyeButton);
21:       frame.add(textField);
22:
23:       frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
24:       frame.pack();
25:       frame.setVisible(true);
26:    }
27: }
```


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{   int 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

Accessing Variables from Enclosing Scope

- Remarkable: Inner class can access variables from enclosing scope e.g. `textField`
- Can access enclosing instance fields, local variables
- Local variables must be marked final
  `final JTextField textField = ...;`

User Interface Actions

- 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!");
  ```

- Ch4/action1/ActionTester.java


```
01: import java.awt.*;
02: import java.awt.event.*;
03: import javax.swing.*;
04:
05: public class ActionTester
06: {
07:    public static void main(String[] args)
08:    {
09:       JFrame frame = new JFrame();
10:
11:       final int FIELD_WIDTH = 20;
12:       final JTextField textField = new JTextField(FIELD_WIDTH);
13:       textField.setText("Click a button!");
14:
15:       JButton helloButton = new JButton("Say Hello");
16:
17:       helloButton.addActionListener(new
18:          ActionListener()
19:          {
20:             public void actionPerformed(ActionEvent event)
21:             {
22:                textField.setText("Hello, World!");
23:             }
24:          });
25:
26:       JButton goodbyeButton = new JButton("Say Goodbye");
27:
28:       goodbyeButton.addActionListener(new
29:          ActionListener()
30:          {
31:             public void actionPerformed(ActionEvent event)
32:             {
33:                textField.setText("Goodbye, World!");
34:             }
35:          });
36:
37:       frame.setLayout(new FlowLayout());
38:
39:       frame.add(helloButton);
40:       frame.add(goodbyeButton);
41:       frame.add(textField);
42:
43:       frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
```

```
44:       frame.pack();
45:       frame.setVisible(true);
46:    }
47: }
```

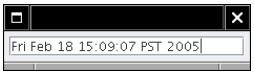

Constructing Related Actions

- Write helper method that constructs objects
- Pass variable information as parameters
- Declare parameters final

```
public static ActionListener createGreetingButtonListener(   final String message){ return new   ActionListener()   {      public void actionPerformed(ActionEvent event)      {         textField.setText(message);      }   };}
```


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
- Ch4/timer/TimerTester.java




```
01: import java.awt.*;
02: import java.awt.event.*;
03: import java.util.*;
04: import javax.swing.*;
05: import javax.swing.Timer;
06:
07: /**
08:    This program shows a clock that is updated once per second.
09: */
10: public class TimerTester
11: {
12:    public static void main(String[] args)
13:    {
14:       JFrame frame = new JFrame();
15:
16:       final int FIELD_WIDTH = 20;
17:       final JTextField textField = new JTextField(FIELD_WIDTH);
18:
19:       frame.setLayout(new FlowLayout());
20:       frame.add(textField);
21:
22:       ActionListener listener = new
23:          ActionListener()
24:          {
25:             public void actionPerformed(ActionEvent event)
26:             {
27:                Date now = new Date();
28:                textField.setText(now.toString());
29:             }
30:          };
31:       final int DELAY = 1000;
32:          // Milliseconds between timer ticks
33:       Timer t = new Timer(DELAY, listener);
34:       t.start();
35:
36:       frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
37:       frame.pack();
38:       frame.setVisible(true);
39:    }
40: }
```

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; . . .}
```

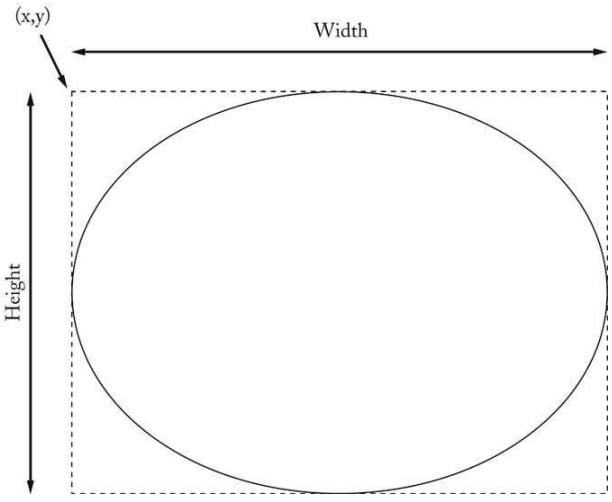
- Can draw any object that implements Shape interface

```
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

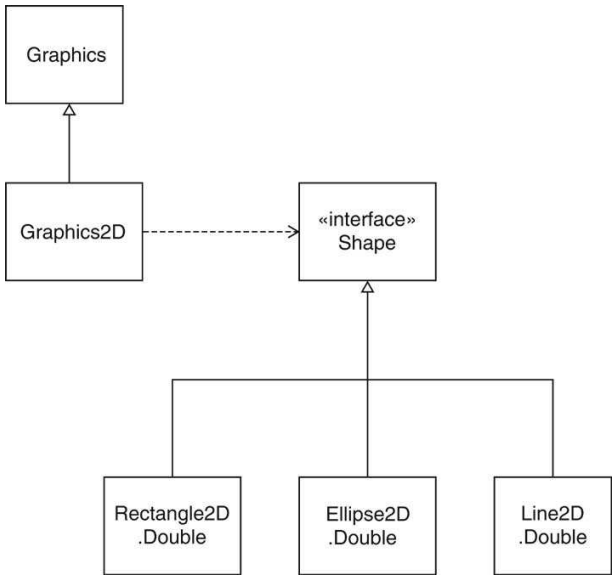


Drawing Line Segments

- 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

Filling Shapes

- Fill interior of shape

```
g2.fill(shape);
```

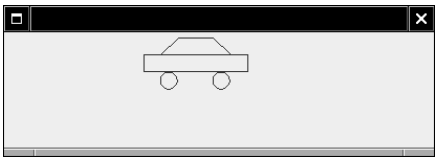
- Set color for fills or strokes:
  `g2.setColor(Color.red);`
- Program that draws car
  Ch4/icon3/CarIcon.java


```
01: import java.awt.*;
02: import java.awt.geom.*;
03: import javax.swing.*;
04:
05: /**
06:    An icon that has the shape of a car.
07: */
08: public class CarIcon implements Icon
09: {
10:    /**
11:       Constructs a car of a given width.
12:       @param width the width of the car
13:    */
14:    public CarIcon(int aWidth)
15:    {
16:       width = aWidth;
17:    }
18:
19:    public int getIconWidth()
20:    {
21:       return width;
22:    }
23:
24:    public int getIconHeight()
25:    {
26:       return width / 2;
27:    }
28:
29:    public void paintIcon(Component c, Graphics g, int x, int y)
30:    {
31:       Graphics2D g2 = (Graphics2D) g;
32:       Rectangle2D.Double body
33:             = new Rectangle2D.Double(x, y + width / 6,
34:                   width - 1, width / 6);
35:       Ellipse2D.Double frontTire
36:             = new Ellipse2D.Double(x + width / 6, y + width / 3,
37:                   width / 6, width / 6);
38:       Ellipse2D.Double rearTire
39:             = new Ellipse2D.Double(x + width * 2 / 3, y + width / 3,
40:                   width / 6, width / 6);
41:
42:       // The bottom of the front windshield
43:       Point2D.Double r1
44:             = new Point2D.Double(x + width / 6, y + width / 6);
```

```
45:       // The front of the roof
46:       Point2D.Double r2
47:             = new Point2D.Double(x + width / 3, y);
48:       // The rear of the roof
49:       Point2D.Double r3
50:             = new Point2D.Double(x + width * 2 / 3, y);
51:       // The bottom of the rear windshield
52:       Point2D.Double r4
53:             = new Point2D.Double(x + width * 5 / 6, y + width / 6);
54:
55:       Line2D.Double frontWindshield
56:             = new Line2D.Double(r1, r2);
57:       Line2D.Double roofTop
58:             = new Line2D.Double(r2, r3);
59:       Line2D.Double rearWindshield
60:             = new Line2D.Double(r3, r4);
61:
62:       g2.fill(frontTire);
63:       g2.fill(rearTire);
64:       g2.setColor(Color.red);
65:       g2.fill(body);
66:       g2.draw(frontWindshield);
67:       g2.draw(roofTop);
68:       g2.draw(rearWindshield);
69:    }
70:
71:    private int width;
72: }
73:
74:
```


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



CRC Card for the MoveableShape Interface Type

MoveableShape
paint the shape
move the shape

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

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

Implementing the Animation

- Ch4/animation/MoveableShape.java
- Ch4/animation/ShapeIcon.java
- Ch4/animation/AnimationTester.java
- Ch4/animation/CarShape.java


```
01: import java.awt.*;
02:
03: /**
04:    A shape that can be moved around.
05: */
06: public interface MoveableShape
07: {
08:    /**
09:       Draws the shape.
10:       @param g2 the graphics context
11:    */
12:    void draw(Graphics2D g2);
13:    /**
14:       Moves the shape by a given amount.
15:       @param dx the amount to translate in x-direction
16:       @param dy the amount to translate in y-direction
17:    */
18:    void translate(int dx, int dy);
19: }
```

```
01: import java.awt.*;
02: import java.util.*;
03: import javax.swing.*;
04:
05: /**
06:    An icon that contains a moveable shape.
07: */
08: public class ShapeIcon implements Icon
09: {
10:    public ShapeIcon(MoveableShape shape,
11:       int width, int height)
12:    {
13:       this.shape = shape;
14:       this.width = width;
15:       this.height = height;
16:    }
17:
18:    public int getIconWidth()
19:    {
20:       return width;
21:    }
22:
23:    public int getIconHeight()
24:    {
25:       return height;
26:    }
27:
28:    public void paintIcon(Component c, Graphics g, int x, int y)
29:    {
30:       Graphics2D g2 = (Graphics2D) g;
31:       shape.draw(g2);
32:    }
33:
34:    private int width;
35:    private int height;
36:    private MoveableShape shape;
37: }
38:
39:
```

```
01: import java.awt.*;
02: import java.awt.event.*;
03: import javax.swing.*;
04:
05: /**
06:    This program implements an animation that moves
07:    a car shape.
08: */
09: public class AnimationTester
10: {
11:    public static void main(String[] args)
12:    {
13:       JFrame frame = new JFrame();
14:
15:       final MoveableShape shape
16:             = new CarShape(0, 0, CAR_WIDTH);
17:
18:       ShapeIcon icon = new ShapeIcon(shape,
19:             ICON_WIDTH, ICON_HEIGHT);
20:
21:       final JLabel label = new JLabel(icon);
22:       frame.setLayout(new FlowLayout());
23:       frame.add(label);
24:
25:       frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
26:       frame.pack();
27:       frame.setVisible(true);
28:
29:       final int DELAY = 100;
30:       // Milliseconds between timer ticks
31:       Timer t = new Timer(DELAY, new
32:          ActionListener()
33:          {
34:             public void actionPerformed(ActionEvent event)
35:             {
36:                shape.translate(1, 0);
37:                label.repaint();
38:             }
39:          });
40:       t.start();
41:    }
42:
```

```
43:    private static final int ICON_WIDTH = 400;
44:    private static final int ICON_HEIGHT = 100;
45:    private static final int CAR_WIDTH = 100;
46: }
```

```
01: import java.awt.*;
02: import java.awt.geom.*;
03: import java.util.*;
04:
05: /**
06:    A car that can be moved around.
07: */
08: public class CarShape implements MoveableShape
09: {
10:    /**
11:       Constructs a car item.
12:       @param x the left of the bounding rectangle
13:       @param y the top of the bounding rectangle
14:       @param width the width of the bounding rectangle
15:    */
16:    public CarShape(int x, int y, int width)
17:    {
18:       this.x = x;
19:       this.y = y;
20:       this.width = width;
21:    }
22:
23:    public void translate(int dx, int dy)
24:    {
25:       x += dx;
26:       y += dy;
27:    }
28:
29:    public void draw(Graphics2D g2)
30:    {
31:       Rectangle2D.Double body
32:             = new Rectangle2D.Double(x, y + width / 6,
33:                   width - 1, width / 6);
34:       Ellipse2D.Double frontTire
35:             = new Ellipse2D.Double(x + width / 6, y + width / 3,
36:                   width / 6, width / 6);
37:       Ellipse2D.Double rearTire
38:             = new Ellipse2D.Double(x + width * 2 / 3, y + width / 3,
39:                   width / 6, width / 6);
40:
41:       // The bottom of the front windshield
42:       Point2D.Double r1
43:             = new Point2D.Double(x + width / 6, y + width / 6);
44:       // The front of the roof
```

```
45:       Point2D.Double r2
46:             = new Point2D.Double(x + width / 3, y);
47:       // The rear of the roof
48:       Point2D.Double r3
49:             = new Point2D.Double(x + width * 2 / 3, y);
50:       // The bottom of the rear windshield
51:       Point2D.Double r4
52:             = new Point2D.Double(x + width * 5 / 6, y + width / 6);
53:       Line2D.Double frontWindshield
54:             = new Line2D.Double(r1, r2);
55:       Line2D.Double roofTop
56:             = new Line2D.Double(r2, r3);
57:       Line2D.Double rearWindshield
58:             = new Line2D.Double(r3, r4);
59:
60:       g2.draw(body);
61:       g2.draw(frontTire);
62:       g2.draw(rearTire);
63:       g2.draw(frontWindshield);
64:       g2.draw(roofTop);
65:       g2.draw(rearWindshield);
66:    }
67:
68:    private int x;
69:    private int y;
70:    private int width;
71: }
```

Implementing the Animation