Chapter 24

Graphical Objects

The simple graphics we saw earlier created *screen artifacts*. A screen artifact is simply an image drawn on the screen (viewport), just as a picture can be drawn on a whiteboard. A picture on the whiteboard can be erased, or another image can be drawn over top of an existing image, but we cannot move an image on a whiteboard unless we erase the existing image and try to draw an exact copy of the original picture in a new location. The "moved" picture is *not* the original picture; it is simply a copy. A picture drawn on a whiteboard is an artifact produced by writing with a marker on the board. The picture cannot exist (outside the mind of the artist, anyway!) without the existence of the whiteboard.

Compare our marker-drawn artifact to a picture drawn on a sticky note (like a 3M Post-it® note). We can stick the note (and its associated picture) anywhere on the whiteboard. If it gets in the way of our current work on the board, we can pull it off and stick it somewhere else without redrawing the picture. The sticky note is an object that can exist independent of the whiteboard. The note can be manipulated in ways that are impossible with an artifact.

A *graphical object* is a software object that can be visualized and manipulated programmatically. Often users can directly interact with graphical objects. Examples of graphical objects include buttons.

24.1 The `GraphicalObject` Class

Our `GraphicalObject` class provides the basic functionality for graphical objects. To use a graphical object you follow these steps:

1. Create an instance of a subclass of `GraphicalObject`.
2. Add the graphical object instance to an existing viewport.

The subclass of `GraphicalObject` must override the `draw()` method in order for the graphical object to be visible within the viewport. Your subclass constructor must also set the object's width and height. `SimpleStar` (24.2) illustrates a simple graphical object that is shaped like a star. When the user clicks

The `GraphicalObject` class has many similarities to our `Viewport` class:

- It has all the methods for drawing primitive graphical shapes: lines, rectangles, polygons, etc. The methods are used just as they are in `Viewport` objects—they are called from within the graphical object's `draw()` method.
- It provides the same methods for manipulating the size and position: `setSize()` and `setLocation()`.

- It can receive input events. Methods named identically to their viewport counterparts can overridden to handle mouse activity and key presses.
- Other graphical objects can be added to a graphical object just as a graphical object can be added to a viewport.

`GraphicalStarObject` (24.1) defines a simple star-shaped graphical object:

```
import edu.southern.computing.oopj.GraphicalObject;

public class GraphicalStarObject extends GraphicalObject {
    public GraphicalStarObject(int size) {
        super(size, size);
        setCursor(HAND);
        setBackground(TRANSPARENT);
    }
    public void draw() {
        //  Code adapted from http://www.research.att.com/
        //                    sw/tools/yoix/doc/graphics/
        //                    pointInPolygon.html
        int size = getWidth(),
            xCenter = size/2,
            yCenter = size/2;
        drawPolygon(xCenter,  yCenter - round(0.50*size),
                    xCenter +  round(0.29*size),  yCenter +  round(0.40*size),
                    xCenter - round(0.47*size),  yCenter - round(0.15*size),
                    xCenter + round(0.47*size),  yCenter - round(0.15*size),
                    xCenter - round(0.29*size),  yCenter + round(0.40*size));
    }

    public void setCenter(int x, int y) {
        //  Center star at (x,y)
        super.setLocation(x - getWidth()/2, y - getHeight()/2);
    }
}
```

Listing 24.1: `GraphicalStarObject`—a simple star-shaped graphical object

The `setCenter()` method conveniently allows us to to position the star relative to its center instead of relative to the left-top corner of its bounding box. `SimpleStar` (24.2) uses the `GraphicalStarObject`:

```
import edu.southern.computing.oopj.Viewport;
import edu.southern.computing.oopj.GraphicalObject;

public class SimpleStar {
    public static void main(String[] args) {

        final GraphicalStarObject star = new GraphicalStarObject(100);
```



```
        //  Create a viewport to hold our star
        Viewport w = new Viewport("Star: Click to move", 100, 100, 600, 500) {
            public void mouseReleased() {  //  Reposition the star
                star.setCenter(getMouseX(), getMouseY());
            }
        };

        w.add(star);  //  Add the star to the viewport
    }
}
```

Listing 24.2: SimpleStar—uses the star object

Here we override the viewport's mouse released method to reposition the star's center to the location of the mouse event. Notice how the star can now be moved from one place to another within the viewport just as a post it note can be moved from one place on a whiteboard to another. The star object itself is responsible for rendering itself, not the star's client code.

One built-in capability of graphical objects allows for a more interesting effect. All graphical objects can be made movable via a setMovable() method that accepts a Boolean value: true = movable and false = immovable. The user can drag a movable graphical object around the viewport with the mouse. MovableStar (24.3) allows the user to drag the star around in the viewport:

```
import edu.southern.computing.oopj.Viewport;
import edu.southern.computing.oopj.GraphicalObject;

public class MovableStar {
    public static void main(String[] args) {

        final GraphicalStarObject star = new GraphicalStarObject(100);
        star.setMovable(true);

        //  Create a viewport to hold our star
        Viewport w = new Viewport("Star: Click to move", 100, 100, 600, 500);

        w.add(star);

    }
}
```

Listing 24.3: MovableStar—allows the user to drag the star around in the viewport

To illustrate how graphical objects can be composed, see MovableCompositeStar (24.4):

```
import edu.southern.computing.oopj.Viewport;
import edu.southern.computing.oopj.GraphicalObject;
```



```
public class MovableCompositeStar {
    public static void main(String[] args) {

        //  Make a small, movable star object
        GraphicalStarObject littleStar = new GraphicalStarObject(20);
        littleStar.setMovable(true);

        //  Make a bigger star object
        GraphicalStarObject bigStar = new GraphicalStarObject(100);
        //  Add the little star to the big star
        bigStar.add(littleStar);
        littleStar.setCenter(bigStar.getWidth()/2, bigStar.getHeight()/2);
        bigStar.setMovable(true);

        //  Create a viewport to hold our star
        Viewport w = new Viewport("Composite Stars", 100, 100, 600, 500);

        //  Add the big star to the viewport
        w.add(bigStar);
    }
}
```

Listing 24.4: MovableCompositeStar—graphical objects can be composed of other graphical objects

Here we add the little star, a graphical object in its own right, to another graphical object, the big star. Notice how the big star can be dragged taking the little star with it, and the little star can be moved within the big star without affecting the position of the big star.

ContextMenu objects can be added to graphical objects. This is handy when you want the user to change an object's property. In ChangableStar (24.5), the user can change a star's color and size:

```
import edu.southern.computing.oopj.Viewport;
import edu.southern.computing.oopj.GraphicalObject;
import edu.southern.computing.oopj.GraphicalText;
import edu.southern.computing.oopj.ContextMenu;

public class ChangableStar {
    public static void main(String[] args) {

        //  Create a viewport to hold our stars
        Viewport w = new Viewport("Changable Stars", 100, 100, 600, 500);

        GraphicalStarObject star1 = new GraphicalStarObject(100);
        star1.setMovable(true);
        star1.setContextMenu(new StarMenu(star1));
        star1.setLocation(100, 100);
        w.add(star1);
```



```
        GraphicalStarObject star2 = new GraphicalStarObject(100);
        star2.setMovable(true);
        star2.setContextMenu(new StarMenu(star2));
        star2.setLocation(200, 100);
        w.add(star2);
    }
}
```

Listing 24.5: ChangableStar—graphical objects with popup menus

The StarMenu type is a subclass of ContextMenu, as shown in StarMenu (24.6):

```
import edu.southern.computing.oopj.Viewport;
import edu.southern.computing.oopj.GraphicalObject;
import edu.southern.computing.oopj.GraphicalText;
import edu.southern.computing.oopj.ContextMenu;

public class StarMenu extends ContextMenu {
    private GraphicalStarObject star;

    public StarMenu(GraphicalStarObject star) {
        super("Red", "Blue", "Black", "Bigger", "Smaller", "Quit");
        this.star = star;
    }

    private void scale(double factor) {
        //  Current size
        int size = star.getWidth();
        //  There is a limit to the scaling
        if (size > 5 && size < 500) {
            //  Compute star's current center
            int xCenter = star.getX() + size/2,
                yCenter = star.getY() + size/2;
            //  Resize the star
            star.setSize(GraphicalObject.round(factor * size),
                         GraphicalObject.round(factor * size));
            //  Recenter the scaled star (according to new size)
            star.setLocation(xCenter - star.getWidth()/2,
                             yCenter - star.getHeight()/2);
        }
    }

    public void handler(String item) {
        if (item.equals("Red")) {
            star.setForeground(GraphicalObject.RED);
        } else if (item.equals("Blue")) {
```



```
            star.setForeground(GraphicalObject.BLUE);
        } else if (item.equals("Black")) {
            star.setForeground(GraphicalObject.BLACK);
        } else if (item.equals("Bigger")) {
            scale(1.2);
        } else if (item.equals("Smaller")) {
            scale(0.8);
        } else if (item.equals("Quit")) {
            System.exit(0);
        }
        star.visuallyUpdate();
    }
}
```

Listing 24.6: `StarMenu`—a context-sensitive popup menu for modifying `GraphicalStarObject`s

24.2 Graphical Text Objects

`GraphicalText` is a straightforward extension (that is, subclass) of `GraphicalObject`. Its constructor accepts a string argument, and a graphical text object renders itself by drawing the string. `TextGraphics` (24.7) adds some graphical text objects to a viewport:

```
import edu.southern.computing.oopj.Viewport;
import edu.southern.computing.oopj.GraphicalObject;
import edu.southern.computing.oopj.GraphicalText;

public class TextGraphics {
    public static void main(String[] args) {
        //  Create a white viewport to hold our text
        Viewport w = new Viewport("Text Graphics", 100, 100, 600, 500);
        w.setBackground(Viewport.WHITE);

        //  Create and set up the text objects
        GraphicalText text1 = new GraphicalText("String #1: Can't move me");
        text1.setLocation(10, 10);

        GraphicalText text2 = new GraphicalText("String #2: CAN move me");
        text2.setMovable(true);
        text2.setLocation(10, 30);

        GraphicalText text3 = new GraphicalText("String #3: I'm red!");
        text3.setForeground(GraphicalText.RED);
        text3.setLocation(10, 70);

        GraphicalText text4 = new GraphicalText("String #4: I'm blue!");
        text4.setForeground(GraphicalText.BLUE);
        text4.setMovable(true);
```



```
        text4.setLocation(10, 110);

        //  Add the text to the viewport
        w.add(text1);
        w.add(text2);
        w.add(text3);
        w.add(text4);
    }
}
```

Listing 24.7: `TextGraphics`—uses some movable and immobile graphical text objects

Text objects `text2` and `text4` can be moved by the user; `text1` and `text3` have fixed positions.

24.3 Graphical Buttons

A natural extension of `GraphicalText` is the `GraphicalButton` class. Wrap the graphical text in a box that responds to mouse clicks, and you have a graphical button.

24.4 Summary

- Add summary items here.

24.5 Exercises

1. Add exercises here.

