Homework4, Spring 2020

Adam Corbin (acorbin3@fau.edu)

March 2020

Contents

1 q5.1

1.1 a

a) What is the purpose of a design pattern ?

1.1.1 a answer

The purpose of a design pattern is to organize a set of behaviours in a way that the program can be easily maintained and easy to understand, and easy to expand on new behaviors or features.

1.2 b

b) When do you apply the Observer pattern ?

1.2.1 b answer

An observer pattern is used extensively when developing UI applications. For example when there is a button and the program wants to do an action when the button is pressed using an ActionListener this would use the Observer pattern.

1.3 c

c) You review a design written by somebody else for an application and you find these:

- an interface Shape with a method draw()
- a class Circle that implements Shape
- a class Rectangle that implements Shape
- a class CompoundShape that: o implements interface Shape o aggregates 0 or more Shape objects, o has an extra method called add(Shape sh) o for implementing method draw() calls the draw() method for all aggregated Shape objects. You assume that a CompoundShape object is made of multiple shapes. What design pattern is at work in this application? Explain your answer.

1.3.1 c answer

This is using a Strategy pattern because we have a base class called Shape where the other classes Circle, Rectangle, and CompoundShape are inheriting the draw method interface, but each class will individually have different implementations within that draw routine.

We also see a Composite pattern in the CompoundShape where it holds multiple shapes and calls the children's draw routine.

1.4 d

d) The TitledBorder class can give a title to a border. Consider the code

panel.setBorder(new TitledBorder(new EtchedBorder(), “Enter Option”));

What design pattern(s) are at work? Explain your answer. (a similar example is in the textbook/notes)

1.4.1 d answer

This is a decorator pattern because we are setting up a not only a Titled boarder, but a special kind using the EtchedBoarder class It would be possible to just use the TitledBoarder class which remain undecorated.

2 q5.1

Suppose you have to design a text editor class (TextEditor) that should benefit from multiple variants of a spell cheking algorithm. Users of the TextEditor class would have to supply custom versions of the spell cheking algorithm to support spelling in different languages.

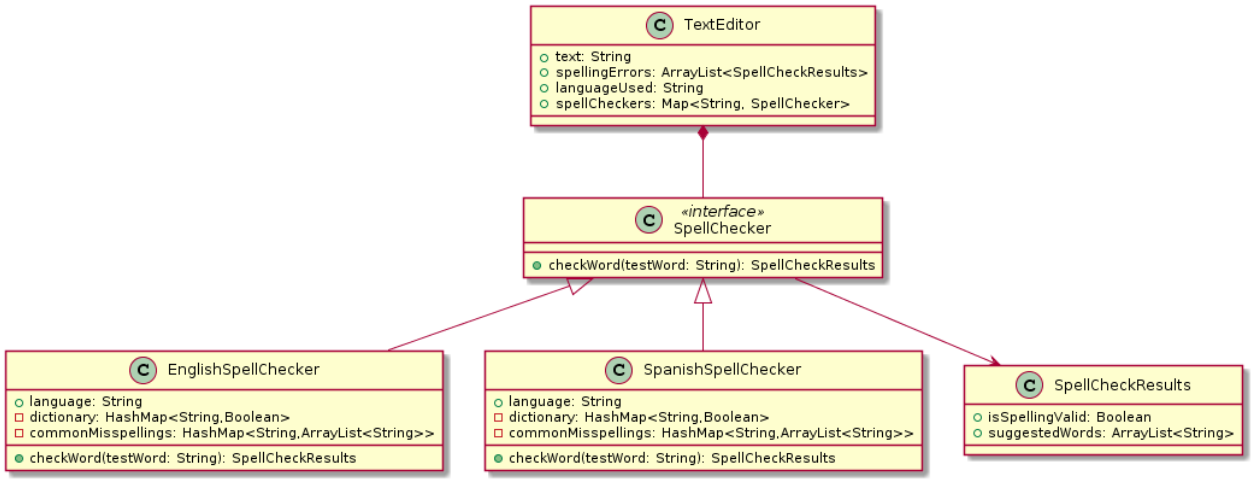
2.1 a

a) What design pattern would you use and why ?

We would want to use the strategy because we could define a core set of interfaces and have different underlying implementations to accomplish the goal for each language.

2.2 b

b) Write the UML class diagram for the design pattern as it applies to this problem.



2.3 c

c) Write a table that lists the relationship between the names from the identified design pattern and the classes/interfaces from your problem.

2.3.1 Strategy design pattern

Name in Design Pattern	Actual Name
Strategy	SpellChecker
ConcreteStrategy	EnglishSpellChecker
ConcreteStrategy	SpanishSpellChecker
doWork()	checkWord is the interface to the SpellChecker

2.4 d

d) Write the skeleton Java code for the design: definitions for classes with instance variables and methods as detailed as we can from the problem description.

2.4.1 EnglishSpellChecker.java

```
1 import java.util.ArrayList;
2 import java.util.HashMap;
3 /**
4  * This class will handle the English version of a spell checker.
5  * In general the test word will be checked if it lives in the Dictionary. If not
6  * we have a misspelling in where we will keep track of a common misspellings. If not
7  * in either then the results will be invalid spelling and no suggestions
```

```
 */
public class EnglishSpellChecker implements SpellChecker  {

    String language;
    HashMap<String,Boolean> dictionary = new HashMap();
    HashMap<String, ArrayList<String>> commonMisspellings = new HashMap();
    EnglishSpellChecker(String language){
        this.language = language;
    }

    @Override
    public SpellCheckerResults checkWord(String testWord) {

        if(this.dictionary.containsKey(testWord)){
            return new SpellCheckerResults(true,new ArrayList());
        }
        else if(this.commonMisspellings.containsKey(testWord)){
            return new SpellCheckerResults(false,this.commonMisspellings.get(testWord));
        }else{
            return new SpellCheckerResults(false,new ArrayList());
        }
    }
}
```

2.4.2 SpanishSpellChecker.java

```
import java.util.ArrayList;
import java.util.HashMap;

/**
 * This class will handle the Spanish version of a spell checker.
 * In general the test word will be checked if it lives in the Dictionary. If not
 * we have a misspelling in where we will keep track of a common misspellings. If not
 * in either then the results will be invalid spelling and no suggestions
 */
public class SpanishSpellChecker implements SpellChecker {

    String language;
    HashMap<String,Boolean> dictionary = new HashMap();
    HashMap<String, ArrayList<String>> commonMisspellings = new HashMap();
    SpanishSpellChecker(String language){
        this.language = language;
    }
    @Override
    public SpellCheckerResults checkWord(String testWord) {
        if(this.dictionary.containsKey(testWord)){
            return new SpellCheckerResults(true,new ArrayList());
        }
        else if(this.commonMisspellings.containsKey(testWord)){
            return new SpellCheckerResults(false,this.commonMisspellings.get(testWord));
        }else{
            return new SpellCheckerResults(false,new ArrayList());
        }
    }
}
```

2.4.3 SpellChecker.java

```java
/**
 * This interfaced is designed to be a base of any kind of spellchecker. Each class that
 * implements the check word would use their own implementation of how to check if the
     word is spelled correctly
 */
public interface SpellChecker {
    public SpellCheckerResults checkWord(String word);
}
```

2.4.4 SpellCheckerResults.java

```java
import java.util.ArrayList;

/**
 * This class is used to represent if a word is spelled correctly and if not a list of
 * suggested words that might be the correct spelling
 */
public class SpellCheckerResults {

    Boolean isValid;
    ArrayList<String> suggestedWords;

    SpellCheckerResults(Boolean isValid, ArrayList<String> suggestedWords){
        this.isValid = isValid;
        this.suggestedWords = suggestedWords;
    }
}
```

2.4.5 TextEditor.java

```java
import java.util.ArrayList;
import java.util.HashMap;

public class TextEditor {
    HashMap<String,SpellChecker> spellCheckers = new HashMap<>();
    String text = "";
    ArrayList<SpellCheckerResults> SpellingErrors = new ArrayList<>();
    String languageUsed = "English";

    /**
     * This class will create 2 spell checkers where they are indexed by the english
         language string
     * This class would be used by using a specific spell checker by iterating over the
         text string
     * to check a single word
     * */
    TextEditor(){
        spellCheckers.put("English",new EnglishSpellChecker("English"));
        spellCheckers.put("Spanish",new SpanishSpellChecker("Spanish"));
    }
}
```

3 q5.3

3.1 BarIcon

```java
import javax.swing.*;
import java.awt.*;

/**
 * This class is used as way to display objects on the screen with a color.
 */
public class BarIcon implements Icon {

    int x,y,width, height;
    Color color;

    BarIcon(int x, int y, int width, int height, Color color){
        System.out.println(y );
        this.width = width;
        this.height = height;
        this.color = color;
        this.x = x;
        this.y = y;
    }

    @Override
    public void paintIcon(Component c, Graphics g, int x, int y) {
        Graphics2D g2 = (Graphics2D) g;
        Rectangle rect2 = new Rectangle(this.x,this.y,width,height);
        g2.setColor(color);
        g2.fill(rect2);
    }

    @Override
    public int getIconWidth() {
        return width;
    }

    @Override
    public int getIconHeight() {
        return height;
    }
}
```

3.2 BarModel

```java
public class BarModel {
    int barWidth = 0;

    /**
     * This class is used to keep track of the width of the Bar
     * @param barWidth initialized value to how big the Bar should be
     */
    BarModel(int barWidth){
        this.barWidth = barWidth;
    }
```

```
11
12      /**
13       * @param updatedWidth - value to update the internal barWidth
14       */
15      public void updateWidth(int updatedWidth){
16          this.barWidth = updatedWidth;
17      }
18  }
```

3.3 GUIController

```
1   import javax.swing.*;
2   import javax.swing.border.Border;
3   import javax.swing.border.CompoundBorder;
4   import javax.swing.border.EmptyBorder;
5   import java.awt.*;
6   import java.awt.event.KeyEvent;
7   import java.awt.event.KeyListener;
8   import java.util.ArrayList;
9   import java.util.Random;
10
11  public class GUIController {
12      /**
13       * In the GUI Controller it will create 3 JTextField, and 3 BarIcons to be added to
             the GUIView
14       * It will also create 3 BarModels which store the value of the bar height
15       */
16      public static void main(String[] args) {
17          GUIView guiView = new GUIView();
18          Random rand = new Random();
19          ArrayList<BarModel> barModelArray = new ArrayList<>();
20          for (int i = 0; i < 3 ; i++) {
21              barModelArray.add(new BarModel(rand.nextInt(100)));
22          }
23
24          int height = 50;
25
26          createBarRow(guiView, barModelArray.get(0), height, 1, Color.red);
27          createBarRow(guiView, barModelArray.get(1), height, 2, Color.green);
28          createBarRow(guiView, barModelArray.get(2), height, 3, Color.blue);
29
30          guiView.repack();
31      }
32
33      /**
34       * This function will create and add JTextFields and BarIcons to the GUIView
35       * There will also have some KeyListners when the JTExtFields are edited. When that
             occurs
36       * the BarModle will update. From that value, the BarIcon will update its width and
             tell the GUIView to refresh
37       *
38       * @param guiView  - the view to add the JTextFields and BarIcons to
39       * @param barModel - the mode where the BarIcon heigths and text values are stored
40       * @param height - The height of how big the BarIcons should be
41       * @param index - index used to edit the holders within the GuiView
```

```
42      * @param color - The color of the BarIcon should be when created
43      */
44     static void createBarRow(GUIView guiView, BarModel barModel, int height, int index,
           Color color) {
45         int maxWidth = 250;
46         BarIcon barIcon = new BarIcon(15,0,maxWidth,height, color);
47         guiView.barIcons.add(barIcon);
48         barIcon.width = (int) (barModel.barWidth/100.0 * maxWidth);
49         JLabel jLabel = new JLabel(barIcon);
50         JTextField textField = new JTextField(3);
51         textField.setPreferredSize(new Dimension(50,50));
52         Border empty = new EmptyBorder(10, 10, 10, 10);
53         Border compound = new CompoundBorder(textField.getBorder(), empty);
54         textField.setEnabled(true);
55         textField.setBorder(compound);
56         textField.setText(String.valueOf(barModel.barWidth));
57         guiView.textFields.add(textField);
58 
59         final int localIndex = index - 1; // Index 0 is the label for the range 0-100
60         textField.addKeyListener(new KeyListener() {
61             @Override
62             public void keyTyped(KeyEvent e) {
63 
64                 updateBarIcon(guiView, barModel, localIndex, maxWidth);
65             }
66 
67             @Override
68             public void keyPressed(KeyEvent e) {
69                 updateBarIcon(guiView, barModel, localIndex, maxWidth);
70             }
71 
72             @Override
73             public void keyReleased(KeyEvent e) {
74                 updateBarIcon(guiView, barModel, localIndex, maxWidth);
75             }
76         });
77 
78         guiView.textPanel.add(textField);
79 
80         guiView.barPanel.add(jLabel);
81         guiView.frame.repaint();
82     }
83 
84     /**
85      * This function will retrieve the text value and update the BarModel object. There
           is also a test to ensure that
86      * the values are numeric and between 0 and 100(inclusive)
87      * @param guiView - Used to find the text value within the text fields
88      * @param barModel - Used to update the barWidth value
89      * @param localIndex - Used to index into the correct BarIcons and TextField arrays
           in the GUIView
90      * @param maxWidth - Used to come up with the scaled value to display on the screen.
           Values from 0-100 was too small
91      *                  on the screen so we have a max size to compute based on the
```

```
         percentage of how big it should be
92       */
93      private static void updateBarIcon(GUIView guiView, BarModel barModel, int
            localIndex, int maxWidth) {
94          try {
95              int size = Integer.parseInt(guiView.textFields.get(localIndex).getText());
96              if (0 <= size && size <= 100) {
97                  barModel.updateWidth(size);
98                  guiView.barIcons.get(localIndex).width = (int) (((double)
                        barModel.barWidth / 100.0) * maxWidth);
99                  guiView.frame.repaint();
100             }
101         }catch (Exception e){
102             System.out.println("ERROR: Parsing Int failed. Be sure to only use
                    integers");
103         }
104     }
105 }
```

3.4 GUIView

```
1  import javax.swing.*;
2  import java.awt.*;
3  import java.util.ArrayList;
4
5  public class GUIView {
6      JFrame frame = new JFrame();
7      JPanel masterPanel = new JPanel(new GridLayout(2,3));
8      JPanel textPanel = new JPanel(new GridLayout(3,1));
9      JPanel barPanel = new JPanel(new GridLayout(3,1));
10     ArrayList<JTextField> textFields = new ArrayList();
11     ArrayList<BarIcon> barIcons = new ArrayList();
12
13     /**
14      * This is the view of the MVC. It handles holding the text panel and the bar panel
            and laying out the pieces
15      */
16     GUIView(){
17         JLabel label = new JLabel("Keep numbers [0,100]");
18         masterPanel.add(label);
19         JLabel label2 = new JLabel(""); // This is only needed because of the Grid layout
20         masterPanel.add(label2);
21
22
23         masterPanel.add(textPanel);
24         masterPanel.add(barPanel);
25         frame.add(masterPanel);
26         frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
27         frame.pack();
28         frame.setVisible(true);
29         frame.setSize(750, 400);
30     }
31
32     /**
33      * After new items get added to the frame its good to repack and set the size back
```

```
         to default
34      */
35     public void repack(){
36         frame.pack();
37         frame.setSize(750,400);
38     }
39 }
```

4 q6.1

4.1 a

a) Explain the purpose of abstract classes in no more than 15 lines.

An abstract class is used to create a base set of interfaces with possible instance variables. This is really great when you might need to create many different kinds of objects that are different but there is a core set of functionality that you want to retain. An Abstract Class can also have implemented methods so the children classes dont need to reinvent the wheel. This works great in a hierarchy mapping of many classes that might want to inherit other methods or fields from the parent class.

4.2 b

b) Give an example for a situation when an abstract class cannot be used in a Java program and an interface is the only choice.

Java doesn't have multiple inheritance so you can only extend a class with 1 abstract class. However, you can implement as many interfaces as you want to a class.

4.3 c

c) GeneralPath collects shapes and is itself a shape. What design pattern does it implement ? Explain.

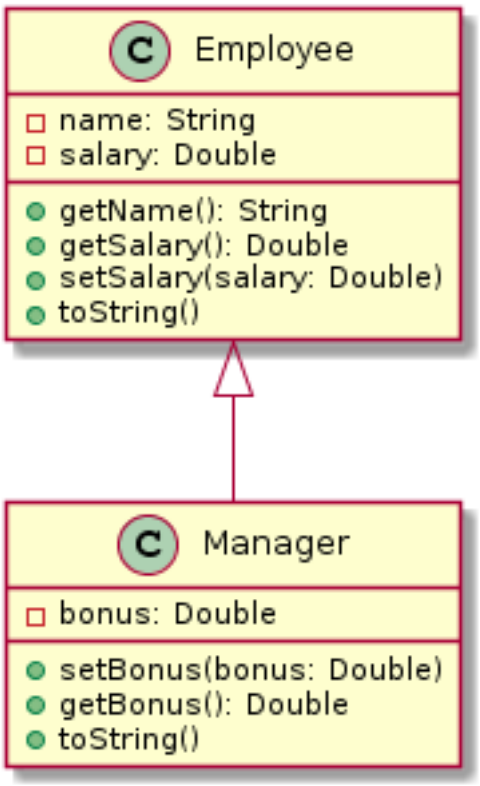
This is an instance of a composite pattern since it contains instances of children classes. The GeneralPath is the main container that holds multiple other shapes and as a whole is considered one object.

5 6.2

5.1 Template Method design pattern

Name in Design Pattern	Actual Name
AbstractClass	Employee
ConcreteClass	Manager
templateMethod()	toString()
primitiveOp1()	getName(),getSalary(), setSalary()

5.2 UML



5.3 Employee.java

```java
/**
 * This is the base class
 */
public class Employee {
    private String name;
    private Double salary;

    Employee(String name){
        this.name = name;
    }

    public String getName(){
        return this.name;
    }

    public void setSalary(Double salary) {
        this.salary = salary;
    }
    public Double getSalary() {
        return salary;
    }

    /**
     * This is the template method in which any class that extends this class will
         inherit this toString method
```

```
     * @return
     */
    public String toString() {
        return "Employee{" +
                "name='" + this.getName() + '\'' +
                ", salary=" + this.getSalary() +
                '}';
    }
}
```

5.4 Manager.java

```
public class Manager extends Employee {
    private Double bonus;
    Manager(String name) {
        super(name);
    }

    public Double getBonus() {
        return bonus;
    }

    public void setBonus(Double bonus) {
        this.bonus = bonus;
    }


}
```

6 q6.3

6.1 SelectableShape.java

```
import java.awt.*;

/**
   A shape that manages its selection state.
*/
public abstract class SelectableShape implements SceneShape
{
   public void setSelected(boolean b)
   {
      selected = b;
   }

   public boolean isSelected()
   {
      return selected;
   }

   public void drawSelection(Graphics2D g2)
   {
      //Capture the default stroke to be used to reset after the dashed objects get drawn
      g2.setColor(Color.black);
      Stroke defaultStroke = g2.getStroke();
```

```
23      draw(g2);
24
25      if(this.isSelected()){
26         g2.setColor(Color.BLUE);
27         float[] dash1 = { 5.0f };
28         g2.setStroke(new BasicStroke(2.0f,
29                 BasicStroke.CAP_BUTT, BasicStroke.JOIN_MITER, 5.0f, dash1, 0.0f));
30         //Get the bounds of the object, make it a little bit larger and align up to
               boarder to be at the
31         // bottom right corner
32         Rectangle rect = this.getPath().getBounds();
33         rect.height = rect.height + 5;
34         rect.width = rect.width + 5;
35         rect.x = rect.x - 4;
36         rect.y = rect.y - 4;
37         g2.draw(rect);
38
39         //Create the 4 corners
40         Rectangle rect1 = new Rectangle(rect.x-3,rect.y-3,6,6);
41         g2.fill(rect1);
42         Rectangle rect2 = new Rectangle(rect.x-3,rect.y-3 + rect.height,6,6);
43         g2.fill(rect2);
44         Rectangle rect3 = new Rectangle(rect.x-3  + rect.width,rect.y-3 +
               rect.height,6,6);
45         g2.fill(rect3);
46         Rectangle rect4 = new Rectangle(rect.x-3 + rect.width,rect.y-3,6,6);
47         g2.fill(rect4);
48      }
49
50      //Reset the color and stroke back to normal. Without this you might cause
            subsequent objects
51      // that might not be selected to use the selected color and stroke.
52      g2.setColor(Color.black);
53      g2.setStroke(defaultStroke);
54   }
55
56   private boolean selected;
57 }
```

6.2 CarShape.java

```
1  import java.awt.*;
2  import java.awt.geom.*;
3
4  /**
5     A car shape.
6  */
7  public class CarShape extends CompoundShape
8  {
9     /**
10       Constructs a car shape.
11       @param x the left of the bounding rectangle
12       @param y the top of the bounding rectangle
13       @param width the width of the bounding rectangle
14    */
```

```
15    public CarShape(int x, int y, int width)
16    {
17       Rectangle2D.Double body
18          = new Rectangle2D.Double(x, y + width / 6,
19             width - 1, width / 6);
20       Ellipse2D.Double frontTire
21          = new Ellipse2D.Double(x + width / 6, y + width / 3,
22             width / 6, width / 6);
23       Ellipse2D.Double rearTire
24          = new Ellipse2D.Double(x + width * 2 / 3,
25             y + width / 3,
26             width / 6, width / 6);
27
28       // The bottom of the front windshield
29       Point2D.Double r1
30          = new Point2D.Double(x + width / 6, y + width / 6);
31       // The front of the roof
32       Point2D.Double r2
33          = new Point2D.Double(x + width / 3, y);
34       // The rear of the roof
35       Point2D.Double r3
36          = new Point2D.Double(x + width * 2 / 3, y);
37       // The bottom of the rear windshield
38       Point2D.Double r4
39          = new Point2D.Double(x + width * 5 / 6, y + width / 6);
40       Line2D.Double frontWindshield
41          = new Line2D.Double(r1, r2);
42       Line2D.Double roofTop
43          = new Line2D.Double(r2, r3);
44       Line2D.Double rearWindshield
45          = new Line2D.Double(r3, r4);
46
47       add(body);
48       add(frontTire);
49       add(rearTire);
50       add(frontWindshield);
51       add(roofTop);
52       add(rearWindshield);
53    }
54 }
```

6.3 CompoundShape.java

```
1  import java.awt.*;
2  import java.awt.geom.*;
3
4  /**
5     A scene shape that is composed of multiple geometric shapes.
6  */
7  public abstract class CompoundShape extends SelectableShape
8  {
9     public CompoundShape()
10    {
11       path = new GeneralPath();
12    }
```

```
13
14    protected void add(Shape s)
15    {
16       path.append(s, false);
17    }
18
19    public boolean contains(Point2D aPoint)
20    {
21       return path.contains(aPoint);
22    }
23
24    public void translate(int dx, int dy)
25    {
26       path.transform(
27             AffineTransform.getTranslateInstance(dx, dy));
28    }
29
30    public Rectangle getPath(){
31       return path.getBounds();
32    }
33
34    public void draw(Graphics2D g2)
35    {
36       g2.draw(path);
37    }
38
39    private GeneralPath path;
40 }
```

6.4 HouseShape.java

```
1  import java.awt.*;
2  import java.awt.geom.*;
3
4  /**
5     A house shape.
6  */
7  public class HouseShape extends CompoundShape
8  {
9     /**
10       Constructs a house shape.
11       @param x the left of the bounding rectangle
12       @param y the top of the bounding rectangle
13       @param width the width of the bounding rectangle
14    */
15    public HouseShape(int x, int y, int width)
16    {
17       Rectangle2D.Double base
18          = new Rectangle2D.Double(x, y + width, width, width);
19
20       // The left bottom of the roof
21       Point2D.Double r1
22          = new Point2D.Double(x, y + width);
23       // The top of the roof
24       Point2D.Double r2
```

```
         = new Point2D.Double(x + width / 2, y);
      // The right bottom of the roof
      Point2D.Double r3
         = new Point2D.Double(x + width, y + width);

      Line2D.Double roofLeft
         = new Line2D.Double(r1, r2);
      Line2D.Double roofRight
         = new Line2D.Double(r2, r3);

      add(base);
      add(roofLeft);
      add(roofRight);
   }
}
```

6.5 SceneComponent.java

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.MouseAdapter;
import java.awt.event.MouseEvent;
import java.awt.event.MouseMotionAdapter;
import java.util.ArrayList;

/**
   A component that shows a scene composed of shapes.
*/
public class SceneComponent extends JComponent
{
   public SceneComponent()
   {
      shapes = new ArrayList<SceneShape>();

      addMouseListener(new
         MouseAdapter()
         {
            public void mousePressed(MouseEvent event)
            {
               mousePoint = event.getPoint();
               for (SceneShape s: shapes)
               {
                  if (s.contains(mousePoint))
                     s.setSelected(!s.isSelected());
               }
               repaint();
            }
         });

      addMouseMotionListener(new
         MouseMotionAdapter()
         {
            public void mouseDragged(MouseEvent event)
            {
               Point lastMousePoint = mousePoint;
```

```
38                mousePoint = event.getPoint();
39                for (SceneShape s : shapes)
40                {
41                   if (s.isSelected())
42                   {
43                      double dx = mousePoint.getX() - lastMousePoint.getX();
44                      double dy = mousePoint.getY() - lastMousePoint.getY();
45                      s.translate((int) dx, (int) dy);
46                   }
47                }
48                repaint();
49             }
50          });
51    }
52
53    /**
54       Adds an shape to the scene.
55       @param s the shape to add
56    */
57    public void add(SceneShape s)
58    {
59       shapes.add(s);
60       repaint();
61    }
62
63    /**
64       Removes all selected shapes from the scene.
65    */
66    public void removeSelected()
67    {
68       for (int i = shapes.size() - 1; i >= 0; i--)
69       {
70          SceneShape s = shapes.get(i);
71          if (s.isSelected()) shapes.remove(i);
72       }
73       repaint();
74    }
75
76    public void paintComponent(Graphics g)
77    {
78       super.paintComponent(g);
79       Graphics2D g2 = (Graphics2D) g;
80       for (SceneShape s : shapes)
81       {
82          s.draw(g2);
83          if (s.isSelected())
84             s.drawSelection(g2);
85       }
86    }
87
88    private ArrayList<SceneShape> shapes;
89    private Point mousePoint;
90 }
```

6.6 SceneEditor.java

```
1  import java.awt.*;
2  import java.awt.geom.*;
3  import java.awt.event.*;
4  import javax.swing.*;
5
6  /**
7     A program that allows users to edit a scene composed
8     of items.
9  */
10 public class SceneEditor
11 {
12    public static void main(String[] args)
13    {
14       JFrame frame = new JFrame();
15       frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
16
17       final SceneComponent scene = new SceneComponent();
18
19       JButton houseButton = new JButton("House");
20       houseButton.addActionListener(new
21          ActionListener()
22          {
23             public void actionPerformed(ActionEvent event)
24             {
25                scene.add(new HouseShape(20, 20, 50));
26             }
27          });
28
29       JButton carButton = new JButton("Car");
30       carButton.addActionListener(new
31          ActionListener()
32          {
33             public void actionPerformed(ActionEvent event)
34             {
35                scene.add(new CarShape(20, 20, 50));
36             }
37          });
38
39       JButton removeButton = new JButton("Remove");
40       removeButton.addActionListener(new
41          ActionListener()
42          {
43             public void actionPerformed(ActionEvent event)
44             {
45                scene.removeSelected();
46             }
47          });
48
49       JPanel buttons = new JPanel();
50       buttons.add(houseButton);
51       buttons.add(carButton);
52       buttons.add(removeButton);
53
```

```
      frame.add(scene, BorderLayout.CENTER);
      frame.add(buttons, BorderLayout.NORTH);

      frame.setSize(300, 300);
      frame.setVisible(true);
   }
}
```

6.7 SceneShape.java

```
import java.awt.*;
import java.awt.geom.*;

/**
   A shape that is a part of a scene.
*/
public interface SceneShape
{
   /**
      Draws this item.
      @param g2 the graphics context
   */
   void draw(Graphics2D g2);
   /**
      Draws the selection adornment of this item.
      @param g2 the graphics context
   */
   void drawSelection(Graphics2D g2);
   /**
      Sets the selection state of this item.
      @param b true if this item is selected
   */
   void setSelected(boolean b);
   /**
      Gets the selection state of this item.
      @return true if this item is selected
   */
   boolean isSelected();
   /**
      Translates this item by a given amount.
      @param dx the amount to translate in x-direction
      @param dy the amount to translate in y-direction
   */
   void translate(int dx, int dy);
   /**
      Tests whether this item contains a given point.
      @param p a point
      @return true if this item contains p
   */
   boolean contains(Point2D p);

   Rectangle getPath();

}
```