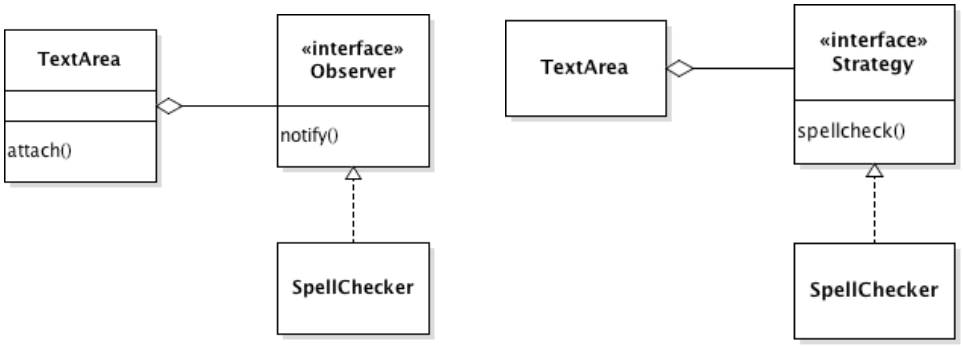
Chapter 5

5.1

a. Design Patterns make it easy to solve problems where a design pattern can be applied. Using a design pattern that was successful in the past, there is a good chance that you will benefit from it as well.
b. The Observer Pattern is used whenever an object needs to be notified of changes of the state of another object.
c. The Composite Design Pattern is at work because a compound shape itself is a shape which contains multiple shapes.
d. The Decorator Pattern is being used here as Panel class is being decorated by the TitledBorder class.

5.2

a. Both the Observer Pattern and Strategy Pattern will be used. The Observer Pattern is used because with every input the spell checking object needs to be notified. The Strategy Pattern is used because the spell checking object needs to be able to supply different spellcheck algorithms.
b.



c.

Name in Design Pattern	Actual Name
Subject	TextArea
Observer	ActionListener
ConcreteObserver	The class that implements the ActionListener interface type
attach( )	addActionListener
notify( )	actionPerformed

Name in Design Pattern	Actual Name
Context	TextArea
Strategy	Spellchecker
ConcreteStrategy	A class that implements the Comparator interface type
doWork( )	spellcheck( )

5.3

Grapher.java

```
package grapher;

import java.awt.EventQueue;
import javax.swing.JFrame;

public class Grapher {

	/**
	 * @param args
	 */
	public static void main(String[] args) {
		EventQueue.invokeLater(new Runnable (){
			public void run (){
				GrapherFrame frame = new GrapherFrame();
				frame.setTitle("Grapher");
				frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
				frame.setVisible(true);
			}
		});
	}

}
```

Data.java

```
package grapher;

import javax.swing.event.*;

/**
 * @author Gregory Prosper
 * Class that creates Data Object for Graph
 */
public class Data {

	private int data1;
	private int data2;
	private ChangeListener listener;

	/**
	 * Creates Data Object with graph 1 and 2 starting at 0
	 */
	public Data() {
		this.data1 = 0;
		this.data2 = 0;
	}

	/**
	 * Changes data for graph 1
```

```
	 * @param i number for 1st bar on graph
	 */
	public void changeData1(int i) {
		this.data1 = i;
		ChangeEvent event = new ChangeEvent(this);
		this.listener.stateChanged(event);
	}

	/**
	 * Gets number for first bar
	 * @return number for first bar
	 */
	public int getData1() {
		return this.data1;
	}

	/**
	 * Changes data for bar 2
	 * @param i number for 2st bar on graph
	 */
	public void changeData2(int i) {
		this.data2 = i;
		ChangeEvent event = new ChangeEvent(this);
		this.listener.stateChanged(event);
	}

	/**
	 * Gets number for second bar
	 * @return number for second bar
	 */
	public int getData2() {
		return this.data2;
	}

	/**
	 * Adds ChangeListener to Data Object
	 * @param listener ChangeListner to be added to Data Object
	 */
	public void addListener(ChangeListener listener) {
		this.listener = listener;
	}
}
```

GrapherFrame.java

```
package grapher;

import java.awt.BorderLayout;
import javax.swing.*;
import javax.swing.event.*;

/**
 * @author Gregory Prosper A frame which contains a TextPanel and GraphComponent
 * */
public class GrapherFrame extends JFrame {
```

```
private JPanel mainPanel;
private TextPanel textPanel;
private GraphComponent graph;
private JLabel label;
private Data data;

/**
 * Creates Grapher Frame
 */
public GrapherFrame() {

	// Create Data Object
	data = new Data();

	// Create ChangeListener for Data Objec
	ChangeListener listener = new ChangeListener() {
		public void stateChanged(ChangeEvent event) {
			graph.update(data.getData1(), data.getData2());
		}
	};

	// Add Listener to Data Object
	data.addListener(listener);

	// initialize components
	mainPanel = new JPanel();
	label = new JLabel("Choose numbers between 0 to 100", JLabel.CENTER);
	graph = new GraphComponent(data.getData1(), data.getData2());
	textPanel = new TextPanel();

	// Create textfield 1 for textfield panel and add Document Listener to
	// it
	final JTextField field1 = new JTextField("0", 10);
	field1.getDocument().addDocumentListener(new DocumentListener() {

		@Override
		public void removeUpdate(DocumentEvent e) {
			String s = field1.getText();
			if (s.length() > 0) {
				data.changeData1(Integer.parseInt(s));
			} else
				data.changeData1(0);
		}

		@Override
		public void insertUpdate(DocumentEvent e) {
			String s = field1.getText();
			if (s.length() > 0) {
				try {
					int number = Integer.parseInt(s);
					if (number > 100 || number < 0)
						throw new NumberFormatException();
					data.changeData1(number);
				} catch (NumberFormatException ex) {
					JOptionPane.showMessageDialog(null,
							"Choose numbers between 0 to 100");
```

```
                                SwingUtilities.invokeLater(new Runnable() {

                                        @Override
                                        public void run() {
                                                // TODO Auto-generated method stub
                                                field1.setText("0");
                                        }
                                });
                        }
                } else
                        data.changeData1(0);
        }

        @Override
        public void changedUpdate(DocumentEvent e) {
                // TODO Auto-generated method stub

        }

});

// Create textfield 2 for textfield panel and add Document Listener to
// it
final JTextField field2 = new JTextField("0", 10);
field2.getDocument().addDocumentListener(new DocumentListener() {

        @Override
        public void removeUpdate(DocumentEvent e) {
                String s = field2.getText();
                if (s.length() > 0) {
                        data.changeData2(Integer.parseInt(s));
                } else
                        data.changeData2(0);

        }

        @Override
        public void insertUpdate(DocumentEvent e) {
                String s = field2.getText();
                if (s.length() > 0) {
                        try {
                                int number = Integer.parseInt(s);
                                if (number > 100 || number < 0)
                                        throw new NumberFormatException();
                                data.changeData2(number);
                        } catch (NumberFormatException ex) {
                                JOptionPane.showMessageDialog(null,
                                                "Choose numbers between 0 to 100");

                                SwingUtilities.invokeLater(new Runnable() {

                                        @Override
                                        public void run() {
                                                // TODO Auto-generated method stub
                                                field2.setText("0");
                                        }
                                });
```

```
                                }
                        } else
                                data.changeData2(0);
                }

                @Override
                public void changedUpdate(DocumentEvent e) {
                        // TODO Auto-generated method stub

                }

        });

        // Add text fields to textPanel
        textPanel.addTextField(field1);
        textPanel.addTextField(field2);

        // Set mainPanel layout
        mainPanel.setLayout(new BorderLayout());

        // Add components to mainPanel
        mainPanel.add(this.label, BorderLayout.NORTH);
        mainPanel.add(this.textPanel, BorderLayout.WEST);
        mainPanel.add(this.graph, BorderLayout.CENTER);
        add(mainPanel);
        setSize(500, 150);
        setResizable(false);
    }
}
```

TextPanel.java

```
package grapher;

import java.awt.*;
import javax.swing.JPanel;
import javax.swing.JTextField;

/**
 * @author Gregory Prosper
 * A Panel which contains text fields
 */
public class TextPanel extends JPanel {

    /**
     * Creates TextPanel with 2 rows and 1 column
     */
    public TextPanel() {
        setLayout(new GridLayout(2, 1));
    }

    /**
     * Adds text field to panel
     * @param field text field to be added to panel
     */
    public void addTextField(JTextField field) {
        add(field);
    }
```

```
}
```

GraphComponent.java

```
package grapher;

import java.awt.*;
import java.awt.geom.Rectangle2D;
import java.util.*;
import javax.swing.JComponent;

/**
 * @author Gregory Prosper
 * Component which draws bar graphs
 */
public class GraphComponent extends JComponent {

	private final int DEFAULT_WIDTH = 300;
	private final int DEFAULT_HEIGHT = 200;
	private Timer timer = new Timer();
	private Rectangle2D rect1;
	private Rectangle2D rect2;
	private int data1;
	private int data2;
	private boolean isAnimating;

	/**
	 * Creates Graph with bars initialized to x1 and x2
	 * @param x1 number for bar 1
	 * @param x2 number for bar 2
	 */
	public GraphComponent(int x1, int x2) {
		this.data1 = x1;
		this.data2 = x2;
	}

	/* (non-Javadoc)
	 * @see javax.swing.JComponent#paintComponent(java.awt.Graphics)
	 */
	public void paintComponent(Graphics g) {
		Graphics2D g2 = (Graphics2D) g;

		g2.setPaint(Color.RED);
		rect1 = new Rectangle2D.Double(10, 7, toPercentage(this.data1)
				* DEFAULT_WIDTH, 40);
		g2.draw(rect1);
		g2.fill(rect1);

		g2.setPaint(Color.BLUE);
		rect2 = new Rectangle2D.Double(10, 65, toPercentage(this.data2)
				* DEFAULT_WIDTH, 40);
		g2.draw(rect2);
		g2.fill(rect2);
	}

	/* (non-Javadoc)
	 * @see javax.swing.JComponent#getPreferredSize()
	 */
	public Dimension getPreferredSize() {
```

```
		return new Dimension(DEFAULT_WIDTH, DEFAULT_HEIGHT);
	}

	/**
	 * Updates bar graph values
	 * @param x1 value for bar 1
	 * @param x2 value for bar 2
	 */
	public void update(int x1, int x2) {
		if (isAnimating){
			this.timer.cancel();
			this.timer = new Timer();
		}
		this.timer.schedule(new Animate(x1, x2), 0, 10);
	}

	/**
	 * @author Gregory Prosper
	 * TimerTask which animates the update of bar graphs
	 */
	private class Animate extends TimerTask {

		/**
		 * Creates Animate object with numbers to be updated to
		 * @param x1 value for bar 1
		 * @param x2 value for bar 2
		 */
		public Animate(int x1, int x2) {
			this.x1 = x1;
			this.x2 = x2;
		}

		/* (non-Javadoc)
		 * @see java.util.TimerTask#run()
		 */
		@Override
		public void run() {
			isAnimating = true;

			if (this.x1 > data1) {
				data1++;
			} else if (this.x1 < data1) {
				data1--;
			}

			if (this.x2 > data2) {
				data2++;
			} else if (this.x2 < data2) {
				data2--;
			}

			if (this.x1 == data1 && this.x2 == data2) {
				cancel();
				isAnimating = false;
			}

			repaint();
		}
```

```
                private int x1;
                private int x2;
        }

        /**
         * Changes regular number into percentage
         * @param i number to be converted
         * @return regular number as percentage
         */
        private float toPercentage(int i) {
                return (float) i / (float) 100;
        }
}
```

Chapter 6

6.1

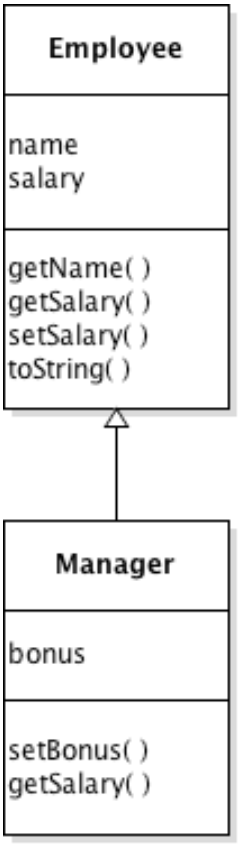
a. An abstract class allows you to extract common behavior between classes. For example, Student and Employee classes would inherit from an Abstract class Person.

b. If you have Circle Shape class that already inherits from shape class, additional functionality cannot be added through an abstract class but through an interface. Objects can only inherit from one Object but can implement multiple interfaces.

c. The Composite Design Pattern is at work because GeneralPath itself is a shape which contains multiple shapes.

6.2

Name in Design Pattern	Actual Name
SuperClass	Employee
SubClass	Manager
teplateMethod( )	toString( )
primitiveOp1( ) primitiveOp2( )	getName( ) getSalary( )



Employee.java

```
public class Employee {

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

	public String getName() {
		return name;
	}

	public void setName(String name) {
		this.name = name;
	}

	public double getSalary() {
		return salary;
	}

	public void setSalary(double salary) {
```

```
                this.salary = salary;
        }

        public String toString(){
                return getName() + " " + this.salary + " " + getSalary();
        }

        private double salary;
        private String name;
}
```

Manager.java

```
public class Manager extends Employee {

        public Manager(String name) {
                super(name);
        }

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

        @Override
        public double getSalary() {
                return this.bonus + super.getSalary();
        }

        private double bonus;
}
```

6.3

SceneEditor.java

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

/**
   A program that allows users to edit a scene composed
   of items.
*/
public class SceneEditor
{
   public static void main(String[] args)
   {
      JFrame frame = new JFrame();
      frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

      final SceneComponent scene = new SceneComponent();

      JButton houseButton = new JButton("House");
      houseButton.addActionListener(new
         ActionListener()
         {
            public void actionPerformed(ActionEvent event)
            {
               scene.add(new HouseShape(20, 20, 50));
            }
         });

      JButton carButton = new JButton("Car");
      carButton.addActionListener(new
         ActionListener()
         {
            public void actionPerformed(ActionEvent event)
            {
               scene.add(new CarShape(20, 20, 50));
            }
         });

      JButton removeButton = new JButton("Remove");
      removeButton.addActionListener(new
         ActionListener()
         {
            public void actionPerformed(ActionEvent event)
            {
               scene.removeSelected();
            }
         });

      JPanel buttons = new JPanel();
      buttons.add(houseButton);
      buttons.add(carButton);
      buttons.add(removeButton);
```

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

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

SceneComponent.java

```
import java.awt.*;
import java.awt.event.*;
import java.awt.geom.*;
import javax.swing.*;
import java.util.*;

/**
   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;
               mousePoint = event.getPoint();
               for (SceneShape s : shapes)
               {
                  if (s.isSelected())
                  {
                     double dx = mousePoint.getX() - lastMousePoint.getX();
                     double dy = mousePoint.getY() - lastMousePoint.getY();
                     s.translate((int) dx, (int) dy);
                  }
```

```
            }
            repaint();
         }
      });
   }

   /**
      Adds an shape to the scene.
      @param s the shape to add
   */
   public void add(SceneShape s)
   {
      shapes.add(s);
      repaint();
   }

   /**
      Removes all selected shapes from the scene.
   */
   public void removeSelected()
   {
      for (int i = shapes.size() - 1; i >= 0; i--)
      {
         SceneShape s = shapes.get(i);
         if (s.isSelected()) shapes.remove(i);
      }
      repaint();
   }

   public void paintComponent(Graphics g)
   {
      super.paintComponent(g);
      Graphics2D g2 = (Graphics2D) g;
      for (SceneShape s : shapes)
      {
         s.draw(g2);
         if (s.isSelected())
            s.drawSelection(g2);
      }
   }

   private ArrayList<SceneShape> shapes;
   private Point mousePoint;
}
```

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
   */

   Rectangle2D getBounds();

   boolean contains(Point2D p);
}
```

SelectableShape.java

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

/**
 * 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) {
                Rectangle2D bounds = getBounds();
```

```
                bounds.setRect(bounds.getMinX() - 5, bounds.getMinY() - 5,
                                bounds.getWidth() + 10, bounds.getHeight() + 10);

                final float dash1[] = {4.0f};
                final BasicStroke dashed = new BasicStroke(2.0f,
                                BasicStroke.CAP_BUTT, BasicStroke.JOIN_MITER, 10.0f, dash1,
                                0.0f);

                Graphics2D g2d = (Graphics2D) g2.create();

                g2d.setStroke(dashed);
                g2d.setColor(Color.BLUE);
                g2d.draw(bounds);

                Rectangle2D corner1 = new Rectangle2D.Double(bounds.getMinX() - 3,
bounds.getMinY() - 3, 6, 6);
                Rectangle2D corner2 = new Rectangle2D.Double(bounds.getMaxX() - 3,
bounds.getMinY() - 3, 6, 6);
                Rectangle2D corner3 = new Rectangle2D.Double(bounds.getMinX() - 3,
bounds.getMaxY() - 3, 6, 6);
                Rectangle2D corner4 = new Rectangle2D.Double(bounds.getMaxX() - 3,
bounds.getMaxY() - 3, 6, 6);

                g2.setColor(Color.BLUE);
                g2.fill(corner1);
                g2.fill(corner2);
                g2.fill(corner3);
                g2.fill(corner4);


                g2.draw(corner1);
                g2.draw(corner2);
                g2.draw(corner3);
                g2.draw(corner4);


                g2.setColor(Color.BLACK);

        }

        private boolean selected;
}
```

CompoundShape.java

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

/**
   A scene shape that is composed of multiple geometric shapes.
*/
public abstract class CompoundShape extends SelectableShape
{
   public CompoundShape()
   {
      path = new GeneralPath();
```

```
   }

   protected void add(Shape s)
   {
      path.append(s, false);
   }

   public boolean contains(Point2D aPoint)
   {
      return path.contains(aPoint);
   }

   public void translate(int dx, int dy)
   {
      path.transform(
            AffineTransform.getTranslateInstance(dx, dy));
   }

   public void draw(Graphics2D g2)
   {
      g2.draw(path);
   }

   public Rectangle2D getBounds(){
          return path.getBounds2D();
   }

   private GeneralPath path;
}
```

CarShape.java

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

/**
   A car shape.
*/
public class CarShape extends CompoundShape
{
   /**
      Constructs a car shape.
      @param x the left of the bounding rectangle
      @param y the top of the bounding rectangle
      @param width the width of the bounding rectangle
   */
   public CarShape(int x, int y, int width)
   {
      Rectangle2D.Double body
         = new Rectangle2D.Double(x, y + width / 6,
            width - 1, width / 6);
      Ellipse2D.Double frontTire
         = new Ellipse2D.Double(x + width / 6, y + width / 3,
            width / 6, width / 6);
      Ellipse2D.Double rearTire
         = new Ellipse2D.Double(x + width * 2 / 3,
```

```
            y + width / 3,
            width / 6, width / 6);

      // The bottom of the front windshield
      Point2D.Double r1
         = new Point2D.Double(x + width / 6, y + width / 6);
      // The front of the roof
      Point2D.Double r2
         = new Point2D.Double(x + width / 3, y);
      // The rear of the roof
      Point2D.Double r3
         = new Point2D.Double(x + width * 2 / 3, y);
      // The bottom of the rear windshield
      Point2D.Double r4
         = new Point2D.Double(x + width * 5 / 6, y + width / 6);
      Line2D.Double frontWindshield
         = new Line2D.Double(r1, r2);
      Line2D.Double roofTop
         = new Line2D.Double(r2, r3);
      Line2D.Double rearWindshield
         = new Line2D.Double(r3, r4);

      add(body);
      add(frontTire);
      add(rearTire);
      add(frontWindshield);
      add(roofTop);
      add(rearWindshield);
   }
}
```

HouseShape.java

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

/**
   A house shape.
*/
public class HouseShape extends CompoundShape
{
   /**
      Constructs a house shape.
      @param x the left of the bounding rectangle
      @param y the top of the bounding rectangle
      @param width the width of the bounding rectangle
   */
   public HouseShape(int x, int y, int width)
   {
      Rectangle2D.Double base
         = new Rectangle2D.Double(x, y + width, width, width);

      // The left bottom of the roof
      Point2D.Double r1
         = new Point2D.Double(x, y + width);
      // The top of the roof
```

```
      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);
   }
}
```