Name: Aldrick Gardiner

Course: COP 4331 003

Date: November 1st 2015

Homework 4

Question 5.1

Answer these questions:

a) What is the purpose of a design pattern?

The purpose of a design pattern is to have simple and elegant solutions to specific problems in object-oriented software design, to increase software **reuse** and **flexibility** and its haves the ability to be developed and evolved over time.

b) When do you apply the Observer pattern?

The observer pattern is applied when there is an object that will change states and there are other dependent objects that are connected that's will be notified and update automatically.

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:
    - implements interface Shape
    - aggregates 0 or more Shape objects,
    - has an extra method called add(Shape sh)
    - 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.

The design pattern that's at work in this application is composite. Composite patterns uses primitive objects can be grouped into composite objects, and the composites themselves are considered primitive objects. In this instances the class CompoundShape is consider the composite objects and the circle and rectangle class would be consider the primitive class, they will be use to create different shapes within the class.

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)

The design pattern that's at work in this code is Decorator. The code above satisfied all the context of the decorator pattern.

- Enhancement of the behavior of a border by adding a title
- A title border is a border so therefore it be used in the same way of any other border
- Borders don't have methods for adding titles
- Other decorations of borders are possible

Question 5.2

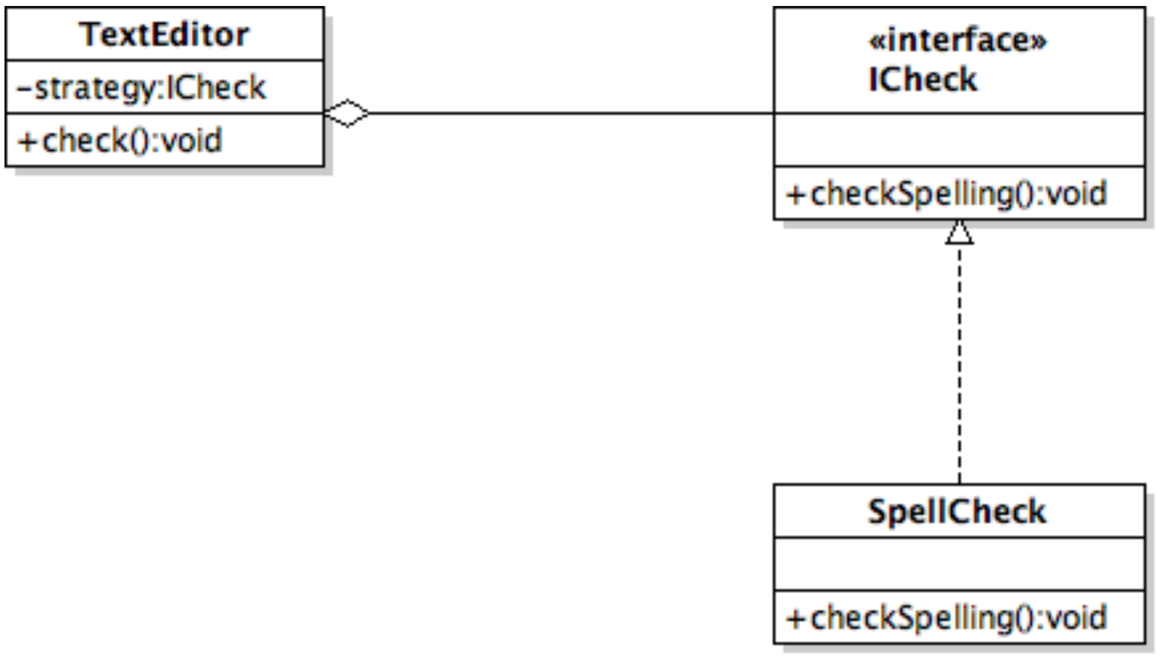
Suppose you have to design a text editor class (TextEditor) that should
benefit from multiple variants of a spell cheking algorithm. Users of the
TextEditor class would have to supply custom versions of the spell cheking
algorithm to support spelling in different languages.

a) What design pattern would you use and why?

The design pattern I would use in this instance is Strategy. Strategy in this case fits the description of the design the best because you apply a strategy pattern whenever you want to allow a client to supply and algorithm and in this case the users of the class will have to supply different algorithms. Also this class passes all of the context for strategy where

- The class benefits from different variants of an algorithm
- Clients will supply custom versions of the algorithm

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



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

Name in Design Pattern	Actual Name
Context	TextEditor
Strategy	ICheck
Concrete Strategy	SpellCheck
doWork()	checkSpelling()

Question 5.3

Write a program that displays editable bar graphs. The GUI has two vertical panels.
The left panel contains textfields with numbers. The right panel contains corresponding
horizontal bar graphs that graphically show the value of the numbers on the left. Use a rectangle
with the width equal to the numbers from the textfield.

Filename: BarChart.java

```
import java.awt.Color;
import java.awt.Component;
import java.awt.Graphics;
import java.awt.Graphics2D;
import java.awt.geom.Rectangle2D;
import javax.swing.Icon;

public class BarChart implements Icon {

    private final Color barColor;
    private int barWidth;
    private final int barHeight;

   public BarChart(Color color) {

       barWidth = 200;
       barHeight = 30;
       barColor = color;

   }
    @Override
   public void paintIcon(Component c, Graphics g, int x, int y)
   {
       Graphics2D g2 = (Graphics2D) g;
       Rectangle2D.Double r = new Rectangle2D.Double(x, y,
barWidth, barHeight );
```

```
        g2.setColor(barColor);
        g2.fill(r);
    }

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

     @Override
    public int getIconHeight()
    {
        return barHeight;
    }

    void setWidth (int width)
    {
        barWidth = width;
    }

}
```

Filename: BarChartTester

```
import java.awt.BorderLayout;
import java.awt.Color;
import java.awt.FlowLayout;
import java.awt.event.KeyEvent;
import java.awt.event.KeyListener;
import javax.swing.BorderFactory;
import javax.swing.BoxLayout;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JPanel;
import javax.swing.JTextField;
import javax.swing.SwingConstants;

/**
 *
 * @author Ace
 */
public class BarChartTester {

        /** The entry main method
      * @param args */
    public static void main(String[] args) {

        JFrame frame = new JFrame();
        Color colors [] = new Color[]{Color.red, Color.green,
Color.blue};
        BarChart bars[] = new BarChart [3];
        JTextField textfields[] = new JTextField[3];
        JLabel labels [] = new JLabel[3];

        JPanel m = new JPanel();
        m.setLayout(new FlowLayout());

        JLabel tlabel = new JLabel ("Select from 1-100");


        JPanel p1 = new JPanel();
        p1.setLayout(new BoxLayout(p1,BoxLayout.Y_AXIS));


        JPanel p2 = new JPanel();
        p2.setLayout(new BoxLayout(p2, BoxLayout.Y_AXIS));

        for (int i = 0; i <= 2; i++)
        {
            BarChart bar = new BarChart(colors[i]);
            JLabel label = new JLabel(bar);
            JTextField Tfs = new JTextField();
            bars[i] = bar;
            labels[i] = label;

label.setBorder(BorderFactory.createEmptyBorder(5,5,5,5));
            label.setHorizontalAlignment(SwingConstants.LEFT);
            textfields[i] = Tfs;
            Tfs.setColumns(2);
            p1.add(Tfs);
            p2.add(label);
            Tfs.addKeyListener(new KeyListener() {

             @Override
                 public void keyTyped(KeyEvent k) {
                     //throw new
UnsupportedOperationException("Not supported yet."); //To change
body of generated methods, choose Tools | Templates.
                 }

                 @Override
                 public void keyPressed(KeyEvent k) {
                     //throw new
UnsupportedOperationException("Not supported yet."); //To change
body of generated methods, choose Tools | Templates.
                 }

                 @Override
                 public void keyReleased(KeyEvent k) {
                     try {

                      int width = Integer.parseInt(Tfs.getText());
                      if(width >100) {width = 100;}
                      if (width < 0) {width = 0;}
                      bar.setWidth((int) (width*2));
                      label.repaint();
                      Tfs.setText(Integer.toString(width));
                    } catch (Exception e){

                    }
                 }

             });
            }



        m.add(p1);
        m.add(p2);
        frame.add(tlabel, BorderLayout.NORTH);
        frame.add(m,BorderLayout.WEST);
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.pack();
        frame.setTitle("Absolute Positioning Demo");
        frame.setSize(400, 200);
        frame.setVisible(true);
    }

}
```

Question 6.1

Answer these questions:
a) Explain the purpose of abstract classes in no more than 15 lines.

The purpose of abstract classes is to function as base classes, which can be extended by subclasses to create a full implementation. Abstract classes improve the situation by preventing a developer from instantiating the base class, because a developer has marked it as having missing functionality. It also provides compile-time safety so that you can ensure that any classes that extend your abstract class provide the bare minimum functionality to work, and you don't need to worry about putting stub methods that inheritors somehow have to magically know that they have to override a method in order to make it work.

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.

In a case where an abstract class is not favorable to use and its best to use an interface is the only choice is implementing multiple inheritances. Java does not allow multiple inheritances. In Java, a class can only derive from one class, whether it’s abstract or not. However, a class can implement multiple interfaces – which could be considered as an alternative to for multiple inheritances. So, one major difference is that a Java class can inherit from only one abstract class, but can implement multiple interfaces.

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

The design pattern that GeneralPath implements is Composite pattern. The CompoundShape class that was implement in the question in chapter 5 makes use of the GeneralPath class in the standard library therefore if CompoundShape uses the GeneralPath to implement its code and the CompoundShape is of a Composite Design pattern then GeneralPath must be of a Composite design also.

Question 6.2

Consider the Employee class hierarchy from the beginning of Chapter 6.
Use the Template Method design pattern to implement a Template Method
for the Employee class:

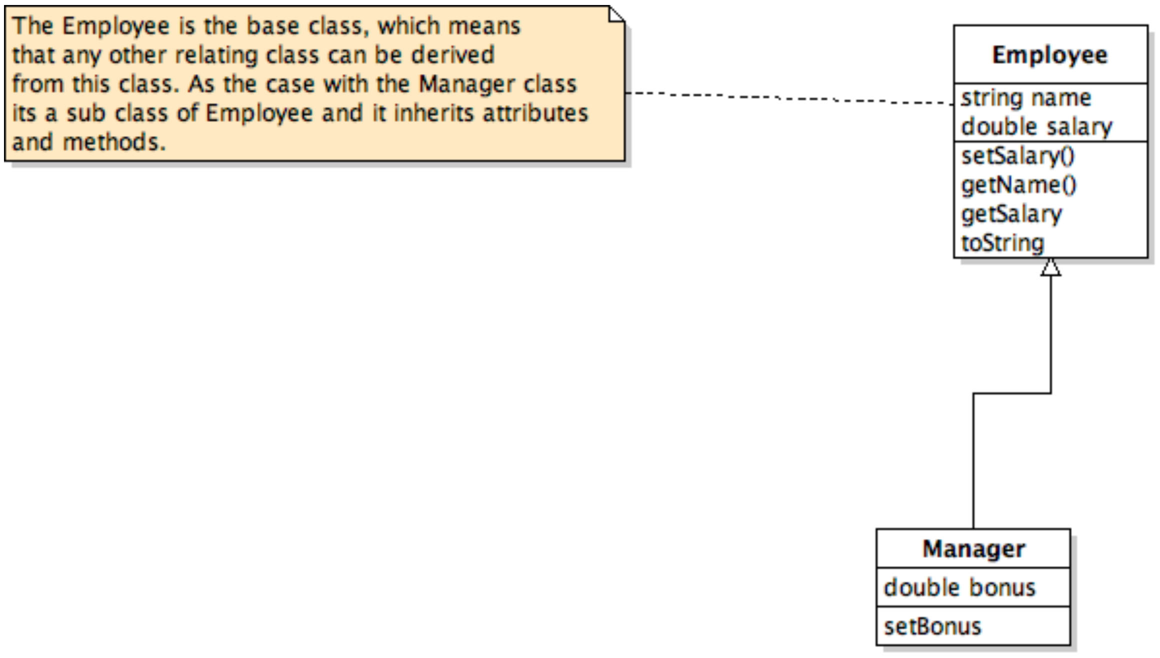
```
      String toString() {...}
```

that generates a string with employee's name, base salary (the salary field),
and the total salary (as returned by getSalary()).

Describe the mappings from the pattern context to the Employee problem domain.
Show the UML class diagram.The diagram needs to describe relationships between
the base class Employee and for one subclass, Manager.
attributes, and methods involved in implementing toString().
Write the Java code for classes Employee and Manager.

In terms of the mapping from the pattern context to the Employee problem domain is

- The String toString() method is applicable for multiple types, the toString returns the value of multiple types, double and string in this case.
- The toString() method can be broken down into primitive operations. the toString method in this case uses other methods that return different data types and combine it so it all return in a one statement.
- In toString() method the order of which the methods call other method is not type restricted where it has to be in a particular type order.


The Employee is the base class, which means that any other relating class can be derived from this class. As the case with the Manager class its a sub class of Employee and it inherits attributes and methods.
Employee
string name
double salary
setSalary()
getName()
getSalary
toString
Manager
double bonus
setBonus

Filename:Employee.java

```
public class Employee {

   public Employee(String aName) { name = aName; }
   public void setSalary(double aSalary) { salary = aSalary; }
   public String getName() { return name; }
   public double getSalary() { return salary; }
   public String toString()
   {
       return "Employee's name: "+this.getName()+"\n"+"Base
Salary: "+this.salary+"\n"+"Total Salary:
"+this.getSalary()+"\n";
   }
   String name;
   private double salary;

}
```

Filename:Manager.java

```
public class Manager extends Employee {


    public Manager(String aName) {
        super(aName); // Calls superclass constructor to
initialize private fields of superclass
        bonus = 0;
    }
    public void setBonus(double aBonus) { bonus = aBonus; } //
new method
    public double getSalary() { return super.getSalary() + bonus;
} // overrides Employee method
    private double bonus; // new field


}
```

Change the SceneEditor application from directory
code/Ch6/scene3
(available at
http://wisenet.fau.edu/class/cop4331/book-code/Ch6/scene3/)
such that a SceneShape object in selected state is drawn with a dashed blue rectangle
around it, with small blue squares (side size=6 pixels) marking the corners.
A screenshot how it should look is available at
http://wisenet.fau.edu/class/cop4331/SelectableShape-selected.png

Hints:
You may use method GeneralPath.getBounds2D() to get the enclosing rectangle.
Use method Graphics2D.setStroke(Stroke s) with a BasicStroke parameter.

Filename: 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()
   {
      super();
      path = new GeneralPath();
      bounds2D = path.getBounds2D();
   }

   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);
      bounds2D = path.getBounds2D();
      x = bounds2D.getX() - 5;
      y = bounds2D.getY() - 5;
      width = bounds2D.getWidth() + 10;
      height = bounds2D.getHeight() + 10;
      g2.setColor(Color.blue);
      Rectangle2D.Double square1 = new Rectangle2D.Double(x,
y,6,6);
      g2.draw(square1);
      g2.fill(square1);

      Rectangle2D.Double square2 = new
Rectangle2D.Double((x+width-5), y,6,6);
      g2.draw(square2);
      g2.fill(square2);

      Rectangle2D.Double square3 = new Rectangle2D.Double(x,
(y+height-5),6,6);
      g2.draw(square3);
      g2.fill(square3);


      Rectangle2D.Double square4 = new
Rectangle2D.Double((x+width-5), (y+height-5),6,6);
      g2.draw(square4);
      g2.fill(square4);
      g2.setStroke(dashed);
      g2.draw(new Rectangle2D.Double(x, y,width,height));
      g2.setStroke(new BasicStroke());
      g2.setColor(Color.black);

   }


   private GeneralPath path;
   Rectangle2D bounds2D;
   private transient Stroke stroke;
   double x = 0;
   double y = 0;
   double width = 0;
   double height = 0;
   final static float dash1[] = {10.0f};
   final BasicStroke dashed = new BasicStroke(1.0f,
BasicStroke.CAP_BUTT, BasicStroke.JOIN_MITER, 10.0f, dash1,
0.0f);

}
```