Homework 3

5.1

a) This pattern is used to increase the functionality of an object or enhance its behavior. This allows the ability to add functionality to an existing method, while also keeping the ability to use the existing ones.
b) This is used when the program is expecting a changed or event. This allows the program to do multiple task when this event is triggered. The observer class supplies method to attach observers that in turn have the ability to perform the mentioned tasks.
c) This is using the composite pattern since is taking a interface and then customizing it for the other classes as needed. The other key feature is that a single call of a particular method depending of the class of the implementation.

5.2

a) Decorator design pattern would be used here. There is common functionality that needs to be supplied by the programmer but at the same time some common functionalities can and should be used. This will keep an independence of other languages as well as common methods without changes available to use. Also keeps the main structure of the editor intact keeping expected data, methods and behaviors pristine which helps smooth runtime.

TextEditor.java

```java
package q5_2;

import java.util.Arrays;
import java.util.HashMap;

public class TextEditor {
    private HashMap<String, SpellCheck> dictionaries =  new HashMap<>();
    private String initialLanguage = "English";

    public void addDictionary(SpellCheckImpl spellCheck, String languageName){}
}
```

SpellCheck.java

```java
package q5_2;

import java.util.HashMap;

public interface SpellCheck {
    void addWord(String word);
    void ignoreWord(String word);
    void checkWord(String word);
    HashMap<String, String> getDictionary(String language);

}
```

SpellCheckImpl.java

```java
package q5_2;

import java.util.HashMap;

public class SpellCheckImpl implements SpellCheck {
    @Override
    public void addWord(String word) {

    }

    @Override
    public void ignoreWord(String word) {

    }

    @Override
    public void checkWord(String word) {

    }

    @Override
    public HashMap<String, String> getDictionary(String language) {
        return null;
    }
}
```

JapaneseSpellCheck.java

```java
package q5_2;
```

```java
public class JapaneseSpellCheck extends SpellCheckImpl {
    public void checkClassicJapanese(String Word){

    }
}
```

ArabSpellCheck.java

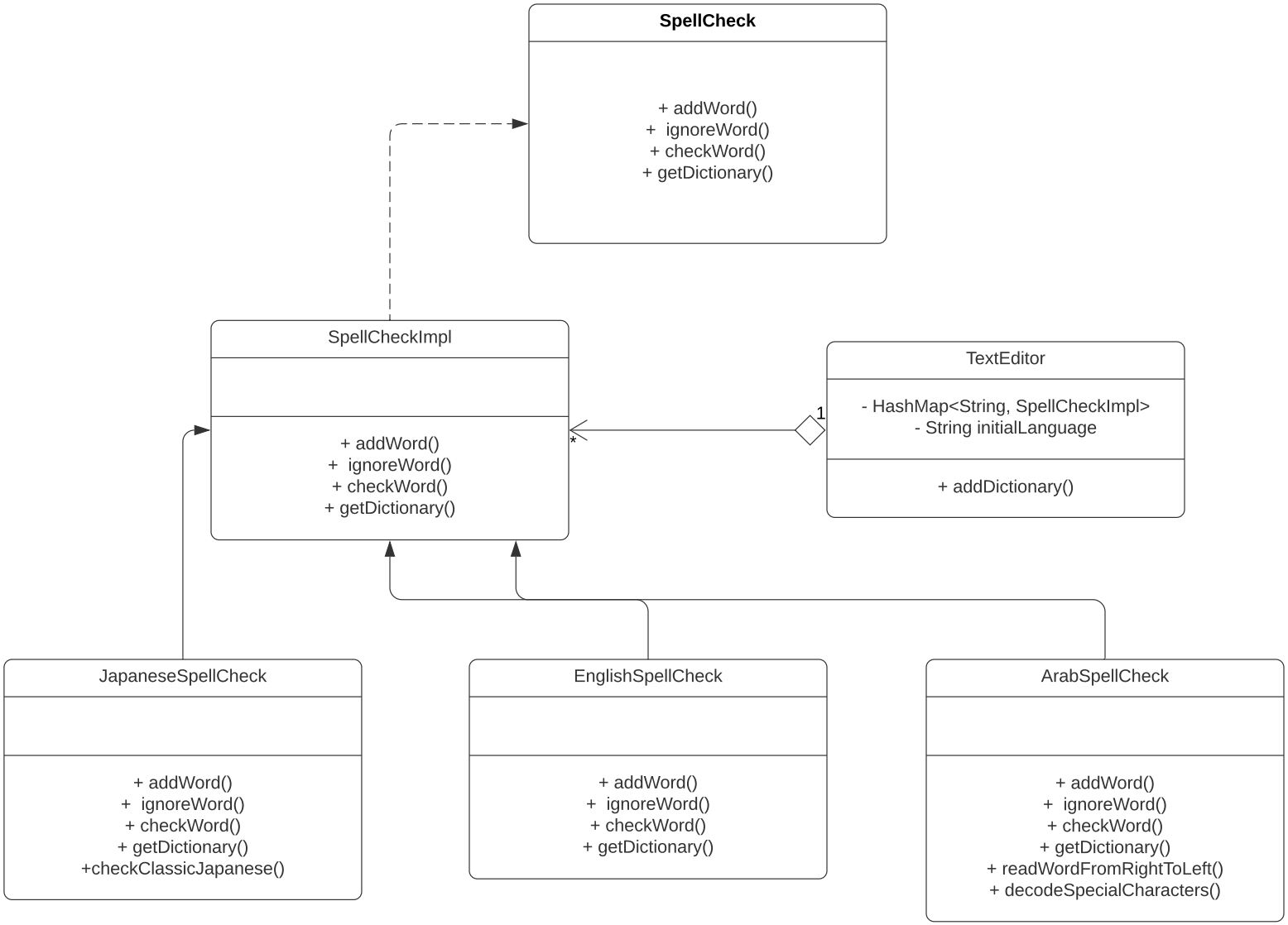
```java
package q5_2;

public class ArabSpellCheck extends SpellCheckImpl {

    public void readWordRightToLeft(String word) {
        //changes the direction the word is read and checked
    }

    public void decodeSpecialChracters(){

    }
}
```

EnglishSpellCheck.java

```java
package q5_2;

public class EnglishSpellCheck extends SpellCheckImpl {

}
```

Q5.2

Diego Gutierrez | March 6, 2020



5.3

GraphTester.java

```
package q5_3;

public class GraphTester {
    public static void main(String[] args) {
         GraphView graphView = new GraphView();
    }
}
```

GraphViewer.java

```
package q5_3;


import javax.swing.*;
import javax.swing.border.EmptyBorder;
import java.awt.*;
import java.awt.event.KeyEvent;
import java.awt.event.KeyListener;

public class GraphView {

    private GraphController controller= new GraphController();
    public  final  Color[] colors = new Color[]{Color.RED, Color.YELLOW, Color.BLUE};

    public GraphView(){
        GraphModel model = new GraphModel();
        JFrame frame  = new JFrame();
        JPanel mainPanel = new JPanel();
        mainPanel.setLayout(new GridLayout(4, 1));

        JPanel instructions = new JPanel();
        JLabel instruction = new JLabel("Keep Numbers [1,100]");
        instruction.setHorizontalAlignment(JLabel.LEFT);
        instructions.add(instruction);
        mainPanel.add(instructions);

        for (int x = 0; x < colors.length; x++){
            JPanel row = createRow(colors[x], model.getWidth());
            mainPanel.add(row);
        }

        frame.setSize(700, 200);


        frame.add(mainPanel);
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setVisible(true);
    }

    public JPanel createRow(Color color, int width){
        JPanel row = new JPanel();
        row.setBorder(new EmptyBorder(6,6,6,6));
        row.setLayout(new BorderLayout());
        JTextField text = new JTextField(5);
        GraphIcon icon = new GraphIcon(color, width);
```

```
        text.addKeyListener(
                new KeyListener() {
                    public void keyTyped(KeyEvent e) {}
                    public void keyPressed(KeyEvent e) {}
                    public void keyReleased(KeyEvent e) {
                        controller.updateBar(text.getText(), icon);
                        row.repaint();
                    }
                }
        );
        row.add(text, BorderLayout.WEST);
        JLabel label = new JLabel(icon);
        label.setHorizontalAlignment(JLabel.LEFT);
        row.add(label, BorderLayout.CENTER);
        return row;
    }
}
```

GraphModel.java

```
package q5_3;

public class GraphModel {
    private  int width = 100;

    public GraphModel(){ }

    public int getWidth(){ return width; }
}
```

GraphController.java

```
package q5_3;

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

public class GraphController {
    private int maxWidth = 500;

    public void updateBar(String perc, GraphIcon icon){
        int newWidth = 0;
        try{
            newWidth = Integer.parseInt(perc);
            float p = (int) newWidth;
            if (newWidth <= 100) {
                p = (p / 100);
                newWidth = (int) (p * maxWidth);
            }else {
                newWidth = maxWidth;
            }
            icon.setWidth(newWidth);

        }catch (Exception e){
           icon.setWidth(0);
        }
    }

}
```

GraphIcon.java

```java
package q5_3;

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

public class GraphIcon implements Icon {
    int heigth = 50;
    int width;
    Color color;

    GraphIcon(Color color, int width){
        this.width = width;
        this.color = color;
    }

    public void setWidth(int width){ this.width = width; }

    @Override
    public void paintIcon(Component c, Graphics g, int x, int y) {
        Graphics2D g2 = (Graphics2D) g;
        Rectangle2D.Double icon =  new Rectangle2D.Double(x,y, width, heigth);
        g2.setColor(this.color);
        g2.fill(icon);
    }

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

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

5.4

ColorController.java

```
package q5_4;

import javax.swing.*;
import javax.swing.event.ChangeEvent;
import javax.swing.event.ChangeListener;
import java.util.Dictionary;

public class ColorController implements ChangeListener {
    JSlider slider;
    String color;
    ColorModel colorModel;

    public ColorController(JSlider slider, String color, ColorModel model){
        this.slider = slider;
        this.color = color;
        colorModel = model;
    }

    public void stateChanged(ChangeEvent e) {
        colorModel.updateColor(color, slider.getValue());
    }
}
```

ColorIcon.java

```
package q5_4;

import javax.swing.*;
import java.awt.*;
import java.awt.geom.Ellipse2D;
import java.awt.geom.Rectangle2D;

public class ColorIcon implements Icon {
    int size = 200;
    Graphics2D g2;
    private Color color = new Color(0,0,0);

    public void setIconColor(Color color){
        this.color = color;
        g2.setColor(this.color);
    }

    @Override
    public void paintIcon(Component c, Graphics g, int x, int y) {
        g2 = (Graphics2D) g;
        Ellipse2D.Double icon = new Ellipse2D.Double(x, y, size, size);
        g2.setColor(this.color);
        g2.fill(icon);
    }

    @Override
    public int getIconWidth() {
        return size;
    }
```

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

ColorModel.java

```
package q5_4;

import java.awt.*;
import java.util.HashMap;

public class ColorModel {
    private  int sliderMax = 255;
    private int sliderMin= 0;
    private int sliderInit = 0;
    private HashMap<String, Integer> colors;
    private ColorView view;

    public  ColorModel(ColorView view){
        colors = new HashMap<>();
        colors.put("Red", sliderInit);
        colors.put("Green", sliderInit);
        colors.put("Blue", sliderInit);
        this.view = view;
    }

    public void updateColor(String color, int value){
        colors.put(color, value);
        Color newColor = new Color(
                colors.get("Red"),
                colors.get("Green"),
                colors.get("Blue")
        );
        view.updateIconColor(newColor);

    }

    public int getSliderMax() { return sliderMax; }

    public int getSliderMin() { return sliderMin; }

    public HashMap<String, Integer> getColors(){ return colors; }

}
```

ColorTester.java

```
package q5_4;

public class ColorTester {
    public static void main(String[] args) {
        new ColorView();
;    }
}
```

ColorView.java

```
package q5_4;

import javax.swing.*;
import javax.swing.border.EmptyBorder;
import java.awt.*;
import java.awt.geom.GeneralPath;

public class ColorView {

    private  ColorModel model;
    private JFrame frame;
    ColorIcon icon;

    public ColorView() {
        model = new ColorModel(this);
        frame  = new JFrame();
        JPanel mainPanel = new JPanel();
        mainPanel.setLayout(new GridLayout(2, 1));

        JPanel panel = new JPanel();
        panel.setLayout(new BorderLayout());
        icon = new ColorIcon();
        JLabel iconLabel =new JLabel(icon);
        panel.add(iconLabel);
        mainPanel.add(panel, BorderLayout.CENTER);

        JPanel sliders = new JPanel();
        sliders.setLayout(new GridLayout(3,1));
        model.getColors().forEach((color, value) ->{
            JPanel row = createRow(color, value);
            sliders.add(row);
        });
        mainPanel.add(sliders);

        frame.setSize(700, 700);
        frame.add(mainPanel);
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setVisible(true);

    }


    public JPanel createRow(String color, int value){
        JPanel row = new JPanel();
        ColorController controller;
        row.setPreferredSize(new Dimension(700, 20));
        row.setBorder(new EmptyBorder(6,6,6,6));
        row.setLayout(new BorderLayout());
        JSlider slider = new
JSlider(JSlider.HORIZONTAL,model.getSliderMin(),model.getSliderMax(),value);
        controller = new ColorController(slider, color, model);
        slider.setPaintTrack(true);
        slider.setPaintTicks(true);
        slider.setPaintLabels(true);
        slider.setMajorTickSpacing(15);
        slider.addChangeListener(controller);
        row.add(slider, BorderLayout.CENTER);
```

```
        JLabel label = new JLabel(color);
        label.setHorizontalAlignment(JLabel.LEFT);
        row.add(label, BorderLayout.WEST);
        return row;
    }

    public void updateIconColor(Color color){
        icon.setIconColor(color);
        frame.repaint();
    }
}
```

6.1

a) The main purpose of an abstract is to layout a certain step in a behavior that is common. They do not implement some or none of the methods layout in the class, this is done by the implementation. The also can require some properties and members that must be implemented by the sub-class that implements.
b) An interface would be used when is expected to be used by multiple unrelated objects and expect some type of behavior such as serialization. The interface is used by several unrelated classes but share the same behavior. This is also helpful since the author of the interface does not care about the implementation, just the behavior and is able to be implemented in several was but at the same time these implementations are shared.
c) The GeneralPath is using the composite design pattern. This classes treats other objects as primitive object such as doubles from it parent class Path2D. It also applies the component from the same interface/abstract class.

6.2

Employee.java

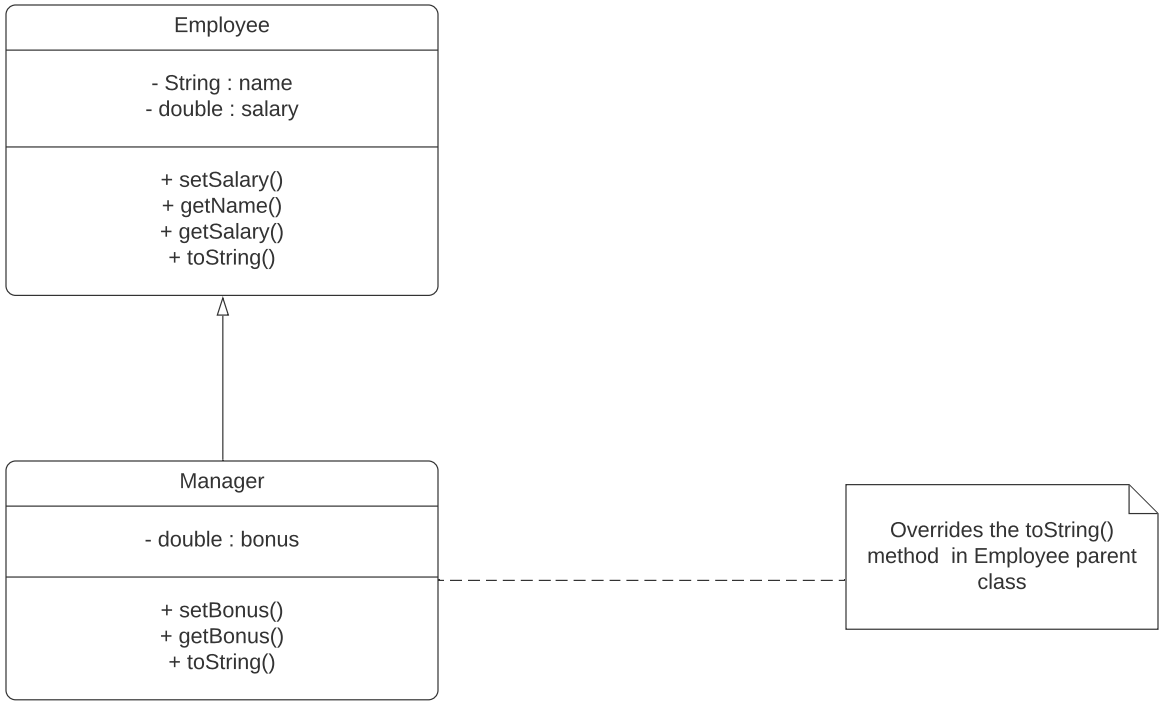
```
package q6_2;

public class Employee {
    private String name;
    private double salary;

    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 "\n ***** Employee Info *****" +
                "\n[Name: " + name + "]" +
                "\n[Base Salary: " + salary +
                "]";
    }
}
```

Manager.java

```
package q6_2;

public class Manager extends Employee {
    private double bonus;

    public Manager(String aName) { super(aName);}
    public void setBonus(double aBonus) { bonus = aBonus; } // new method
    public double getSalary() { return super.getSalary() + bonus; } // overrides
Employee method
    public String toString(){
        return super.toString() +
            "\n[Total Salary: " + getSalary() + "]";
    }

    public static void main(String[] args) {
        Manager manager = new Manager("Doe, Joe");
        manager.setSalary(100000);
        manager.setBonus(20000);
        System.out.println(manager.toString());

    }
}
```

Q6.2

Diego Gutierrez | March 6, 2020



7.1

Pair.java

```
package q7_1;

import java.io.Serializable;

public class Pair<K,V> implements Serializable, Cloneable {

    private K key;
    private V value;

    public Pair(K k, V v){
        key = k;
        value = v;
    }

    public K k(){ return  key; }
    public V v(){ return value; }

    public boolean equals(Object obj){
        if (this == obj) return  true;
        if (obj == null) return true;
        if (getClass() != obj.getClass()) return  false;
        if (this.hashCode() == obj.hashCode()) return true;
        Pair pair = (Pair) obj;
        return this.k() == pair.k() && this.v() == pair.v();
    }

    public  int hashCode(){
        return key.hashCode() +value.hashCode();
    }

    public String toString(){
        return getClass() +
                "[key=" + key.toString() +
                ", value=" + value.toString() +
                "]";
    }

    public Object clone(){
        try {
            return super.clone();
        } catch (CloneNotSupportedException e) {
            return  null;
        }
    }
}
```

PairTest.java

```
package q7_1;

import java.io.*;

public class PairTest {
    public static void main(String[] args) throws Exception {
        Pair<String, Integer> pair = new Pair<>("PIN", 1234);
```

```
        Pair<String, String> pair1 = new Pair<>("Passcode", "P@$$Cod3");
        System.out.println("Testing equals method: Result sould be false");
        System.out.println(pair.equals(pair1));

        System.out.println("Testing clone method: Result sould be true");
        Pair<String, Integer> clone = (Pair<String, Integer>) pair.clone();
        System.out.println(pair.equals(clone));

        System.out.println("Testing serializations: Results sould be true");
        Pair[] secrets = new Pair[2];
        secrets[0] = pair;
        secrets[1] = pair1;

        ObjectOutputStream out = new ObjectOutputStream(new
FileOutputStream("PairTest.dat"));
        out.writeObject(secrets);
        out.close();

        ObjectInputStream in = new ObjectInputStream(new
FileInputStream("PairTest.dat"));
        Pair[] savedSecrets = (Pair[]) in.readObject();
        in.close();

        for (int i = 0; i < secrets.length; i++) {
            System.out.println(secrets[i].equals(savedSecrets[i]));
        }
    }
}
```

7.2

Utils.java

```
package q7_2;

        import q7_1.Pair;

        import java.util.*;

public class Utils {
    public static <K extends Comparable<K>,V> List<Pair<K,V>> sortPairCollection
(Collection<Pair<K,V>> col) {
        List<Pair<K,V>> result = new ArrayList<>();
        result.addAll(col);
        Comparator<Pair<K,V>> keyNameComparator = Comparator.comparing(Pair::k,
Comparable::compareTo);
        Collections.sort(result, keyNameComparator);
        return result;
    }

    public static void main(String[] args) {
        Collection<Pair<String, Integer>> pairStack = new Stack<Pair<String,
Integer>>() {
            {
                add(new Pair("Bella", 2));
                add(new Pair("Dude", 3));
            }
        };
        Collection<Pair<String, Integer>> pairsList = new LinkedList<Pair<String,
Integer>>() {
            {
                add(new Pair("Ronaldo", 2));
                add(new Pair( "Messi", 3));
                add(new Pair( "Bella", 5));
                add(new Pair( "Dude", 1));
            }
        };
        Collection<Pair<Boolean, String>> booleanList = new LinkedList<Pair<Boolean,
String>>() {
            {
                add(new Pair(true, "Bella"));
                add(new Pair(false, "Dude"));
            }
        };
        System.out.println(sortPairCollection(pairStack));
        System.out.println(sortPairCollection(pairsList));
        System.out.println(sortPairCollection(booleanList));
    }
}
```

8.1

Eraser.java

```
package q8_1;

/**
 * Invariant: Name most not equal an empty string or null value at any time.
 */
public class Eraser implements Tool {
    private final String name;

    /**
     * Creates Eraser object with name given on construction.
     * @param name Name of new Eraser.
     * @throws IllegalArgumentException if name is an empty sting or null value.
     */
    public Eraser(String name) throws IllegalArgumentException{
        if (!legalArgumentCheck(name)) {
            throw new IllegalArgumentException("The name is set to invalid value " +
name);
        }
        this.name = name;
        assert  legalArgumentCheck(name);
    }

    /**
     * {@inheritDoc}
     */
    @Override
    public String getName(){ return name; }

    /**
     * {@inheritDoc}
     */
    @Override
    public void use() throws IllegalArgumentException {
        System.out.println("Using " + this.getClass().getSimpleName() + ": " + name);
    }

    /**
     * {@inheritDoc}
     */
    @Override
    public boolean legalArgumentCheck(String name) {
        return (name != null || name.length() > 0);
    }
}
```

Pencil.java

```
package q8_1;

/**
 * Invariant: Name most not equal an empty string or null value at any time.
 */
public class Pencil implements Tool {
    private final String name;

    /**
```

```
     * Creates Pencil object with name given on construction.
     * @param name Name of new Pencil.
     * @throws IllegalArgumentException if name is an empty sting or null value.
     */
    public Pencil(String name ){
        if (!legalArgumentCheck(name)) {
            throw new IllegalArgumentException("The name is set to invalid value " +
name);
        }
        this.name = name;
    }

    /**
     * {@inheritDoc}
     */
    @Override
    public String getName() {
        return name;
    }

    /**
     * {@inheritDoc}
     */
    @Override
    public void use() {
        System.out.println("Using " + this.getClass().getSimpleName() + ": " + name);
    }

    /**
     * {@inheritDoc}
     */
    @Override
    public boolean legalArgumentCheck(String name) {
        return (name != null || name.length() > 0);
    }
}
```

Ruler.java

```
package q8_1;

/**
 * Invariant: Name most not equal an empty string or null value at any time.
 */
public class Ruler implements Tool {
    private final String name;

    /**
     * Creates Pencil object with name given on construction.
     * @param name Name of new Pencil.
     * @throws IllegalArgumentException if name is an empty sting or null value.
     */
    public Ruler(String name){
        if (!legalArgumentCheck(name)) {
            throw new IllegalArgumentException("The name is set to invalid value " +
name);
        }
        this.name = name;
    }
```

```
    /**
     * {@inheritDoc}
     */
    @Override
    public String getName() {
        return name;
    }

    /**
     * {@inheritDoc}
     */
    @Override
    public void use() {
        System.out.println("Using " + this.getClass().getSimpleName() + ": " + name);
    }

    /**
     * {@inheritDoc}
     */
    @Override
    public boolean legalArgumentCheck(String name) {
        return (name != null || name.length() > 0);
    }
}
```

Tool.java

```
package q8_1;

public interface Tool extends Cloneable {
    /**
     * Get the tool name that was set duting the class contrution.
     * @precondition Name !=  null && name != ""
     * @return String with name value.
     */
    String getName();

    /**
     * @precondition Name !=  null && name != ""
     * Prints to console "Using Tool: Name"
     */
    void use();

    /**
     * Invariant check if the name is a valid value.
     * @param name Name to be check for invariant.
     * @return Returns false is the invariant fails.
     */
    boolean legalArgumentCheck(String name);
}
```

Toolbox.java

```
package q8_1;

/**
 * Inveriant: The objects on toolbox can't not be null.
 */
public interface Toolbox {
```

```
    /**
     * Add new Tool to to the Toolbox HashMap
     * @precondition Tool != Null
     * @param tool Tool object to be added to the tool box.
     * @throws NullPointerException if the tool object that is being added is null.
     */
    void add(Tool tool);

    /**
     * Gets Tool from Toolbox HaspMa
     * @precondition None of the tools in the HashMap are set to Null.
     * @param toolName String of the name of the requsted tool.
     * @return Tool of obejct to be requested.
     */
    Tool get(String toolName);
}
```

ToolboxImpl.java

```
package q8_1;

import java.util.HashMap;
import java.util.Map;

public class ToolboxImpl implements Toolbox {
    private static Map<String, Tool> toolbox = new HashMap<>();

    /**
     * {@inheritDoc}
     */
    @Override
    public void add(Tool tool) {
        if (tool == null ) throw new NullPointerException("Parameter tool is set to
null");
        toolbox.put(tool.getName(), tool);
    }

    /**
     * {@inheritDoc}
     */
    @Override
    public Tool get(String toolName) {
        Tool tool =  toolbox.get(toolName);
        if (tool == null){
            System.out.println("Tool is not in toolbox");
            return null;
        } else return tool;
    }
}
```

ToolTest.java

```
package q8_1;

public class ToolTest {
    public static void main(String[] args) {
        ToolboxImpl toolbox = new ToolboxImpl();
        toolbox.add(new Pencil("Number 2"));
        toolbox.add(new Eraser("Dry"));
```

```
        toolbox.add(new Ruler("Metric"));

        toolbox.get("Dry").use();
        toolbox.get("Number 2").use();
        toolbox.get("Metric").use();
        toolbox.get("Test");
    }
}
```

Q8.1

Diego Gutierrez | March 6, 2020