Dwayne Fraser COP 4331 003
Homework # 3

Problem # 1

Homework 3

3.1

a) Encapsolation is used by making all the data in the class private. Encapsulation allows implemented code to be modified without breaking other code.

b) A Method Contract is written in order to establish what is to take place within a Method. It specifies the Before and After, and is good practice for organization.

c) An Immutable class can not be modified. This can be useful when working with code that is not going to change because you want to avoid the possibility of errors if the code becomes changed as you add more. If a string Java class was Mutable, changes can occur that will break other code.

d) The Law of Demeter Attempts to Prevent Side-Effects by only using the variables within a ~~friend~~ neighbor object. To Prevent reaching into an object to gain access to a third object

e) Lack of Consistency in class design causes a loss of Productivity.

Problem #2

```
/*
DWAYNE FRASER
HOMEWORK 3.2
 */

package q2;

public class NVector {

    /**
    * constructor, takes dimension n and sets all elements to 0: NVector(int n)
     * @param n
    */
    public NVector(int n) {
            vector = new double[n];
    }

    /**
    * constructor, takes another NVector and copies all data from "other": NVector(NVector other)
     * @param other
    */
    public NVector(NVector other) {
            vector = (double[])other.vector.clone();
    }


    /**
    * VARARG constructor
     * @param a
    */
    public NVector(double... a) {
            vector = (double[])a.clone();
    }

    /**
       @return vector's size.
    */
    public int length() {
            return vector.length;
    }

    /**
    * returns element with index i
     * @param i
     * @return
    */
    public double get(int i) {
        return vector[i];
    }
```

```java
/**
* Compares two NVector Objects
 * @param o
 */
@Override
public boolean equals(Object o) {

        return this == o;
}

/**
* a method that returns a new copy of an NVector with just one element changed
 * @param i
 * @param x
 * @return
*/
public NVector set(int i, double x) {

    NVector vec = new NVector(this);

        vec.vector[i] = x;

        return vec;
}

/**
* add, returns a new NVector with the sum of this vector and the other
 * @param other
 * @return
*/
public NVector sum(NVector other) {

        NVector vec = new NVector(this.length());

        for (int i=0; i<length(); i++) {
            vec.vector[i] = this.get(i) + other.get(i);
        }
        return vec;
}

/**
returns a double with the scalar product of this vector and another NVector
 * @param other
 * @return
*/
public double sprod(NVector other) {
    double sprod = 0;

        for (int i=0; i<length(); i++) {
            sprod += this.get(i) * other.get(i);
        }
```

```
        return sprod;
    }


    /**
    return a string representation
    */
    @Override
    public String toString() {
        StringBuilder str = new StringBuilder();

        for (int i=0; i<this.length(); i++) {
            str.append(this.vector[i]);
        }

        return str.toString();
    }



    private final double[] vector;
};


/*
DWAYNE FRASER
HOMEWORK 3.2
 */

package q2;

import org.junit.jupiter.api.Test;
import static org.junit.jupiter.api.Assertions.*;

public class TestNVector {
    @Test
    public void testConstructor() {
        double x = 1;
       double y = 2;
       double z = 3;

        NVector vec = new NVector(x, y, z);

        assertTrue(vec.get(0) == x && vec.get(1) == y && vec.get(2) == z);
    }

    @Test
    public void testEquals() {
        double x = 1;
       double y = 2;
```

```
        NVector a = new NVector(x, y);
        NVector b = new NVector(x, y);

        assertTrue(a.equals(b));
    }

    @Test
    public void testGet() {
        double x = 1;
       double y = 2;
       double z = 3;

        NVector vec = new NVector(x, y, z);

        assertTrue(vec.get(0) == x && vec.get(1) == y && vec.get(2) == z);

    }

    @Test
    public void testSum() {
        double x = 1;
       double y = 2;


        NVector a = new NVector(x, y);


        NVector vec = a.sum(a);

        NVector newVec = new NVector(x+y);

        assertTrue(vec.equals(newVec));
    }

    @Test
    public void testSprod() {
        double x = 1;
       double y = 2;

        NVector a = new NVector(x, y);

        double vec = a.sprod(a);
        double newVec = x*y;
        assertTrue(vec == newVec);
    }


    @Test
    public void testToString() {
        double x = 1;
       double y = 2;
```

```
        NVector a = new NVector(x, y);

        String vec = a.toString();

        String newVec = Double.toString(x) + Double.toString(y);

        assertTrue(vec.equals(newVec));
    }
}

/*
DWAYNE FRASER
HOMEWORK 3.2
 */

package q2;

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

        assert new NVector(1, 2, 3, 4).equals(new NVector(1, 2, 3, 4)) : "failed";

        NVector vec = new NVector(1);
        NVector vec2 = vec.set(0, 1);

        NVector vec3 = new NVector(2, 3, 4);
        assert vec3.equals(vec) : "test 1 failed";



        NVector vecSum = vec.sum(vec2);

        NVector sumVec = new NVector(7, 8, 9);

        assert sumVec.equals(vecSum) : "sum() failed";

      double prod = vec.sprod(vec2);
        assert prod == 7 : "sprod() failed";

        System.out.println("sum=" + vecSum);
        System.out.println("sprod=" + prod);
    }
}
```

Problem #3

```
/*
DWAYNE FRASER
HOMEWORK 3.3
 */

package q3;
import java.util.*;
import java.util.Date;

public class Car {
/**
   Constructor
   @param make
   @param model
   @param date
*/
    public Car(String make, String model, Date date) {

        this.make = make;
        this.model = model;
        this.date = date;
    }

    /**
      @return car make
     */
    public String make() {
        return this.make;
    }

    /**
      @return car model
     */
    public String model() {
        return this.model;
    }

    /**

      @return build date
     */
    public Date date() {
        return (Date)date.clone();
    }

    /**
```

```
   @return string from object
 */
@Override
public String toString() {

    StringBuilder sb = new StringBuilder();

    sb.append(make());

    sb.append(model());

    sb.append(date().toString());

    return sb.toString();
}

/**
   @return new comparator object
*/
public static Comparator<Car> getCompByMakeModel() {

    return (Car c1, Car c2) -> {
      if (c1.make().equals(c2.make())) {
         return c1.model().compareTo(c2.model());
      }
      return c1.make().compareTo(c2.make());
   };
};



/**
 * @return
*/
public static Comparator<Car> getCompByDate() {
    return (Car c1, Car c2) -> c1.date().compareTo(c2.date());
}


private final String make;
private final String model;
private final Date date;

   public static void main(String[] args) {

    ArrayList<Car> cars = new ArrayList<>();

    cars.add(new Car("Ford", "Focus", new Date()));
    cars.add(new Car("Chevy", "Colorado", new Date()));
```

```
        cars.add(new Car("Toyota", "Camry", new Date()));



        Collections.sort(cars, Car.getCompByMakeModel());

       cars.forEach(x -> {
          System.out.println(x);
       });

        Collections.sort(cars, Car.getCompByDate());

       cars.forEach(x -> {
          System.out.println(x);
       });
    }
}
```

Problem #4

```
/*
DWAYNE FRASER
HOMEWORK 3.4
 */

package q4;

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

/**
   Icon Color Class
*/
public class IconColor implements Icon
{
   /**
      Constructor.
      @param width
      @param color
   */
   public IconColor(int width, Color color)
   {
         this.width = width;
         this.color = color;
   }

   /**
      @return the icon width
   */
```

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

    /**
       @return the icon height
    */
    @Override
    public int getIconHeight()
    {
        return width;
    }

    /**
       Paint Icon
    */
    @Override
    public void paintIcon(Component c, Graphics g, int x, int y)
    {
        Graphics2D g2 = (Graphics2D)g;
        double r = 100;

        Rectangle2D el = new Rectangle2D.Double(x + width/2 - r, y + width/2 - r, width/2,
width/2);
        g2.setColor(color);
        g2.fill(el);
    }

    /**
    * Sets Icon Color
     * @param color
    */
    public void setColor(Color color)
    {
        this.color = color;
    }

    private Color color;
    private final int width;
}

/*
DWAYNE FRASER
HOMEWORK 3.4
 */

package q4;

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

```
import java.awt.event.*;

public class Frame
{
   /*
   *   Main Class
    */
   public static void main(String[] args)
   {

      final IconColor icon = new IconColor(300, Color.YELLOW);

         final JLabel label = new JLabel(icon);

         JFrame frame = new JFrame();

         JPanel panel = new JPanel();

         panel.setLayout(new FlowLayout());

         Color []colors = new Color[]{Color.RED, Color.YELLOW, Color.BLUE};

         String []colorNames = new String[]{"Red", "Yellow", "Blue"};

         JButton []buttons = new JButton[colors.length];

         int x;

         for (x = 0; x < colors.length; x++) {

            final Color c = colors[x];

            buttons[x] = new JButton(colorNames[x]);

            ActionListener actionListener = (ActionEvent action) -> {
            icon.setColor(c);
            label.repaint();
         };


            buttons[x].addActionListener(actionListener);
            panel.add(buttons[x]);
         }

         frame.add(label, BorderLayout.CENTER);
         frame.add(panel, BorderLayout.SOUTH);

         frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
         frame.pack();
         frame.setVisible(true);
   }
```

```
}
```