Eyan Eubanks COP4331 003

3.1) - a) Explain how encapsulation is used for the implementation of the Day class from the textbook and what is achieved by using that principle (i.e. advantages versus not using it).

Encapsulation is used in the book for the Day class to create a day/ arithmetic object that can handel date arithmetic and give information about current day, distance from a certain day and more. Encapsulation does this by hiding and preventing data being changed. Encapsulation hence benefits the day class is hiding unrelevant data and methods to the program user. This helps for if implementation of the object is changed later on, the program user won't be confused about a attribute not being available, or a unimportant method that they used to be gone because they never had access to it in the first place. This allows for easy changes to software or depreciation of old methods. Another aspect of encapsulation would be having data and methods be bundled together. This isolates classes from each other and nutralizes dependences on other objects. Lets say that another class has been changed that Day was built ontop of. As long as the dependency class follows the rules of Encapsalation this change would not matter as long as that dependency class still retains its methods that were called for the Day object. Thus, preventing debuging and error checking to fix code.

b) What is the role of a contract when we write a method ?

The role of a contract is to make sure that when a service is called by a program user, that the service is complete. The contract assures the program user that this method will perform this certain action as long as the preconditions are met.

C) What are the advantages of making a class immutable ? Discuss in principle, then in the context of the String Java class.

The advantages of making a class immutable is preventing program users from changing or tampering with attributes of objects. This causes unintended user of a class by a program user, and can create unintended output and produce incorrect results. An example could be the string class. If users were able to change strings, then a mutator function would allow for users to make changes to the state of the string object, and cause it to do unpredictable things because preconditions have been changed. This ultimately breaks encapsulation because the data being changed outside of the object.

d) What is the argument of the Law of Demeter ? Explain why we should (try to) follow it.

The Law of Demeter states that an object can only act on its own instance fields, parameters, and objects its constructs with new. Hence, no other object should be able to alter its internal state. We should try to follow this because if other objects can change its state when it wants, then the object's precondition and states can be altered to produce unexpected behavior. This is not good for the program creator or the program user.

e) Explain why a lack of consistency in class design is detrimental.

The lack of consistency in class design is detrimental because it creates confusion. If method naming or implementation has a certain way its design that is recognisable, and then something related is changed to something unrelated, it can cause confusion with class users and create inconvienience.

3.2)

Nvector.java

```
package src;

public class Nvector {

 /**
  * @param n, the capacity of the vector
  * @precondition None
  * @postcondition The vector will be initalized with a capacity of n with 0's
at each index, Nvector > 0
  * @invariant Nvector will always be > 0
  */
 public Nvector(int n) {
   vector = new double[n];
   System.out.println(vector[0]);
 }

 /**
  * @param n, this parameter is an arbitrary number of parameters. These
parameters are then converted into a Nvector
  * @precondition None
  * @postcondition Nvector created with parameters passed to the constructor
  * @invariant
  */
 public Nvector(double... n) {
   vector = new double[n.length];
   for (int i = 0; i < n.length; i++) {
     vector[i] = n[i];
     System.out.println(n[i]);
   }
 }

 /**
  * @param vec, is another Nvector that will be coppied into this current's
objects field
  * @precondition the parameter Nvector needs to be initalized and > 0
  * @postcondition the current object is not equivalent to the parameter
Nvector
  */
 public Nvector(Nvector vec) {
```

```
    vector = new double[vec.length()];
    for (int i = 0; i < vec.length(); i++) {
      vector[i] = vec.get(i);
      System.out.println(vec.get(i));
    }
  }

  /**
   * @param i, is the index of a value in the Nvector
   * @return returns the value corresponding to the index
   * @precondition 0 < i < Nvector.size(), Nvector.size > 0
   * @postcondition The value at the index place is returned as a double
   */
  public double get(int i) {
    return vector[i];
  }

  /**
   * @return returns length of the Nvector
   * @precondition Nvector.size > 0
   * @postcondition the vectors length is returned as an int
   */
  public int length() {
    return vector.length;
  }

  /**
   * @param vec, another Nvector
   * @return Will return wheather current Nvector, and parameter Nvector are
equivalent
   * @precondition For Nvectors to be considered to be equal, need to be same
length,
   * @postcondition Nvectors will either be equivalent which is true returned,
or vice versa
   */
  public boolean isEqual(Nvector vec) {
    if (this.length() != vec.length()) {
      return false;
    }
    for (int i = 0; i < this.length(); i++) {
      if (vector[i] != vec.get(i)) {
        return false;
      }
    }
    return true;
  }

  /**
   * @param i, the index that will be replaced in the copy
```

```
 * @param x, the value that will replace the value at index I
 * @return a new Nvector with the value at that index replaced with x
 * @precondition The current Nvector > 0, 0 <= i < Nvector.length() - 1
 * @postcondition, same as @return
 */
public Nvector set(int i, double x) {
  if (i >= 0 && i < vector.length) {
    double temparry[] = new double[vector.length];

    for (int z = 0; z < vector.length; z++) {
      if (i == z) {
        temparry[i] = x;
      } else {
        temparry[z] = this.get(z);
      }
    }
    return new Nvector(temparry);
  } else {
    System.out.println("You have not choosen a valid index");
    System.exit(1);
  }
  return null;
}

/**
 * @precondition Nvector > 0, this.length() == vec.length()
 * @param vec another Nvector that will be added to current Nvector
 * @return returns a new Nvector that is the sum of both Nvectors
 * @postcondition new Nvect with sum of both Nvectors
 */
public Nvector add(Nvector vec){
  if(this.length() == vec.length()){
    double tempary[] = new double[vector.length];
    for(int i = 0; i < vector.length; i++){
      tempary[i] = vector[i] + vec.get(i);
    }
    return new Nvector(tempary);
  }
  System.out.println("These two Nvectors are not the same size");
  System.exit(1);
  return null;
}

/**
 * @precondition Nvector > 0, this.length() == vec.length()
 * @param vec Nvector that will be multiplied by another Nvector
 * @return the product of the Nvectors indexes summed up
 * @postCondition same as @return
 */
```

```
public double sprod(Nvector vec){
  if(this.length() == vec.length()){
    double sprod = 0.0;
    for(int i = 0; i < vector.length; i++){
      sprod += vector[i] * vec.get(i);
    }
    return sprod;
  }
  System.out.println("These two Nvectors are not the same size");
  System.exit(1);
  return 0.0;
}

/**
 * @precondition Nvector > 0
 * @return returns a string that has all elements of Nvector
 * @postcondition Same as @return
 */
public String toString(){
  String tostr = "[";
  for(int i = 0; i < vector.length; i++){
    if(i == 0){
      tostr += Double.toString(vector[i]);
    }else {
      tostr += "," + Double.toString(vector[i]);
    }
  }
  tostr += "]";
  return tostr;
}

public static void main(String[] args) {
  Nvector test = new Nvector(5);

  Nvector test2 = new Nvector(9, 3, 2, 1, 5, 3);

  Nvector test3 = new Nvector(test2);

  System.out.println(test3.isEqual(test3));
  System.out.println(test3.isEqual(test));

  System.out.println("end of boolean testing");

  Nvector test4 = test3.set(2, 5.6);

  System.out.println("end of set testing");

  test4.add(test3);
```

```
    System.out.println("end of set testing");

    Nvector prodtest = new Nvector(1,1,1,1,1);
    Nvector prodtest1 = new Nvector(3,1,5,6,9);
    System.out.println(prodtest.sprod(prodtest1));

    System.out.println("end of sprod testing");
    System.out.println(test4.toString());
    return;
  }

  private double[] vector;
};
```

NvectorTest.java

```
package src;

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

class NvectorTest {

  @org.junit.jupiter.api.Test
  void VARARGNvector(){
    Nvector testvec = new Nvector(5,6,2,1,2,3,23.9,8.76);
    double controlvec[] = {5,6,2,1,2,3,23.9,8.76};
    for(int i = 0; i < testvec.length();i++){
      assertEquals(controlvec[i], testvec.get(i));
    }
  }


  @org.junit.jupiter.api.Test
  void get() {
    Nvector testvec = new Nvector(3,4,1,2);
    assertEquals(4, testvec.get(1));
  }

  @org.junit.jupiter.api.Test
  void isEqual() {
    Nvector testvec = new Nvector(3,4,1,2);
    Nvector testvec2 = new Nvector(3,4,1,2);
    assertEquals(true, testvec.isEqual(testvec2));
  }

  @org.junit.jupiter.api.Test
  void add() {
    Nvector testvec = new Nvector(3,4,1,2);
    Nvector testvec2 = new Nvector(3,4,1,2);
    assertEquals( "[6.0,8.0,2.0,4.0]", testvec.add(testvec2).toString());
  }
```

```
 @org.junit.jupiter.api.Test
 void sprod() {
   Nvector testvec = new Nvector(3,4,1,2);
   Nvector testvec2 = new Nvector(3,4,1,2);
   assertEquals(30, testvec.sprod(testvec2));
 }

 @org.junit.jupiter.api.Test
 void NvectoString() {
   Nvector testvec = new Nvector(3,4,1,2);
   Nvector testvec2 = new Nvector(3,4,1,2);
   assertEquals("[3.0,4.0,1.0,2.0]", testvec.toString());
 }
}
```

4.1)

Car.java

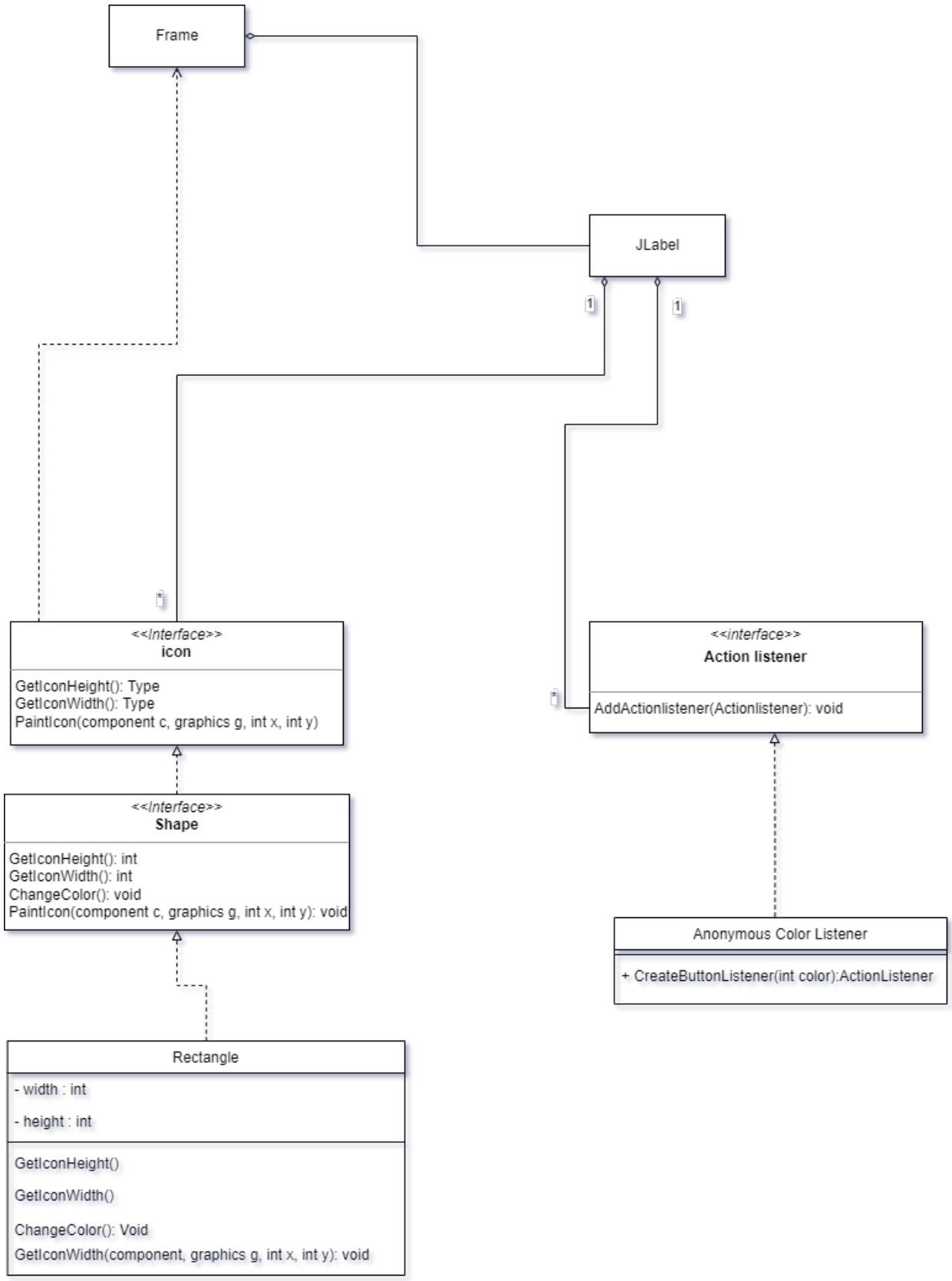
```
import java.util.*;
import java.util.Collections;


public class Car {

 /**
  * @precondition, none
  * @postcondition, object's attributes are initalized
  * @param make, is the make of the car (Toyota, Ford)
  * @param model, is the model of the car (Corolla, Bronco)
  * @param whenBuilt, is the date when the car was built
  */
 public Car(String make, String model, Date whenBuilt){
   this.make = make;
   this.model = model;
   this.whenBuilt = whenBuilt;
 }

 /**
  *
  * @param args
  */
 public static void main(String[] args){
     Date carDate = new Date((long)1104541261);
     Date carDate1 = new Date((long)1102938912);
     Date carDate2 = new Date((long)1102387823);
     Date carDate3 = new Date((long)1103872093);

     ArrayList<Car> CarArray = new ArrayList<Car>();
     Car currcar = new Car("Toyota", "Corolla", carDate);
     Car currcar1 = new Car("Ford", "F150", carDate1);
```

```
        Car currcar2 = new Car("Nissan", "Altima", carDate2);
        Car currcar3 = new Car("Toyota", "Rav4", carDate3);

        CarArray.add(currcar);
        CarArray.add(currcar1);
        CarArray.add(currcar2);
        CarArray.add(currcar3);

        Comparator<Car> carcomparator = Car.getCompByMakeModel();
        Comparator<Car> carDatecomparator = Car.getCompByDate();

        Collections.sort(CarArray, carcomparator);

        for(Car cariter : CarArray) {
            cariter.toStrinG();
            System.out.println("-----------------------------------------");
        }


System.out.println("*****************************************************");
        Collections.sort(CarArray, carDatecomparator);

        for(Car cariter : CarArray) {
            cariter.toStrinG();
            System.out.println("-----------------------------------------");
        }
    }

    /**
     * @precondition, make needs to be initalized
     * @postcondition, none
     * @return returns make of current Car
     */
    public String getmake(){
      return this.make;
    }

    /**
     * @precondition, model needs to be initalized
     * @postcondition, none
     * @return returns model of current Car
     */
    public String getmodel(){
      return this.model;
    }

    /**
     * @precondition, Date needs to be initalized
     * @postcondition, a cloned date is returned
```

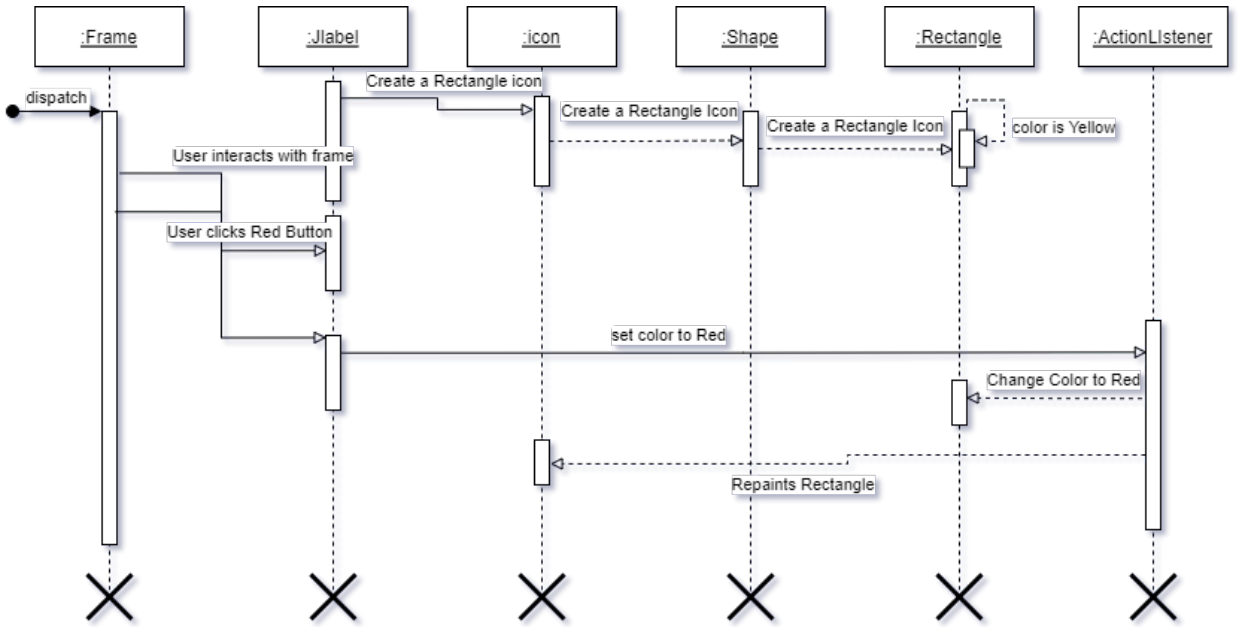
```
  * @return returns a clone of when the Car was built
  */
 public Date getwhenBuilt(){
   return (Date)this.whenBuilt.clone();
 }

 /**
  * @precondition, Make, model, and Date when built needs to be returned
  * @postcondition, none
  */
 public void toStrinG(){
   System.out.println("make:" + this.make + "\nModel: " + this.model + "\nWhen
Built: " + this.getwhenBuilt());
 }

 /**
  * @precondition, none
  * @postcondition, none
  * @return returns a Comparator object for comparison of cars by Make & Model
  */
 public static java.util.Comparator<Car> getCompByMakeModel(){
   return new Comparator<Car>(){
     public int compare(Car car1, Car car2){
       String car1comp = car1.getmake() + " " + car1.getmodel();
       String car2comp = car2.getmake() + " " + car2.getmodel();
       return car1comp.compareTo(car2comp);
     }
   };
 }
 /**
  * @precondition, none
  * @postcondition, none
  * @return returns a Comparator object for comparison of cars by Date
  */
 public static java.util.Comparator<Car> getCompByDate(){
   return new Comparator<Car>(){
     public int compare(Car car1, Car car2){
       return car1.getwhenBuilt().compareTo(car2.getwhenBuilt());
     }
   };
 }

 private String make;
 private String model;
 private Date whenBuilt;
}
```

4.2

a)


Frame
JLabel
1
1
*
*
<<Interface>>
icon
GetIconHeight(): Type
GetIconWidth(): Type
PaintIcon(component c, graphics g, int x, int y)
<<interface>>
Action listener
AddActionlistener(Actionlistener): void
<<Interface>>
Shape
GetIconHeight(): int
GetIconWidth(): int
ChangeColor(): void
PaintIcon(component c, graphics g, int x, int y): void
Anonymous Color Listener
+ CreateButtonListener(int color):ActionListener
Rectangle
- width : int
- height : int
GetIconHeight()
GetIconWidth()
ChangeColor(): Void
GetIconWidth(component, graphics g, int x, int y): void

b)



c)

Rectangle.java

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

public class Rectangle implements Shape{
    /**
     * @precondition none
     * @postcondition new Rectangle Icon that's color is yellow
     * Initiates Rectangle with width, height, size, and rectangle color
to yellow
     * @param width, width of object in pixels
     * @param height, height of object in pixels
     * @param size, area of object in pixels
     */
    public Rectangle(int width, int height, int size){
        this.width = width;
        this.height = height;
        this.size = size;
        this.CurrColor = Color.yellow;
    }

    /**
     * @precondition, none
     * @postcondition, Current Icons height is returned
     * @return, returns Icons height in pixels
     */
    public int getIconHeight(){
        return height;
    }
```

```
        /**
        * @precondition, none
        * @postcondition, Current Icons width is returned
        * @return, returns Icons width in pixels
        */
        public int getIconWidth(){
            return width;
        }
        /**
         * @precondition, the icon has a color, (initially yellow)
         * @postcondition, the icons color will be changed to the given Color
value
         * @return, none
         */
        public void changeColor(Color C){
            this.CurrColor = C;
        }
        /**
         * @precondition, Icon needs to be initalized
         * @postcondition, The Icon needs to be painted and held within a
Jlabel object
         * @return, none
         */
        public void paintIcon(Component c, Graphics g, int x, int y){
            Graphics2D g1 = (Graphics2D) g; // typecast abstract graphics to
2D graphics
            Rectangle2D.Double box = new Rectangle2D.Double(x, y, width,
height);  // create rectangle
            g1.setColor(CurrColor);
            g1.fill(box);
        }
        private int width;
        private int height;
        private int size;

        private Color CurrColor;

    }
```

Shape.java

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

    public interface Shape extends Icon{

        /**
         * @precondition, none
         * @postcondition, Current Icons height is returned
         * @return, returns Icons height in pixels
```

```
         */
        public int getIconHeight();
        /**
         * @precondition, none
         * @postcondition, Current Icons width is returned
         * @return, returns Icons width in pixels
         */
        public int getIconWidth();
        /**
         * @precondition, the icon has a color, (initially yellow)
         * @postcondition, the icons color will be changed to the given Color
value
         * @return, none
         */
        public void changeColor(Color C);
        /**
         * @precondition, Icon needs to be initalized
         * @postcondition, The Icon needs to be painted and held within a
Jlabel object
         * @return, none
         */
        public void paintIcon(Component c, Graphics g, int x, int y);

    }
```

Frame.java

```
    import java.awt.*;
    import java.awt.event.ActionEvent;
    import java.awt.event.ActionListener;
    import javax.swing.*;

    public class Frame{
        /**
         * Is Entry point into program
         * @param args
         */
        public static void main(String[] args){

            JFrame frame = new JFrame();
            Frame myframe = new Frame();

            frame.setVisible(true); // set to visible

            frame.setLayout(new FlowLayout()); // set layout



            // add rectangle for the label
            Rectangle MyRect = new
Rectangle(ICON_WIDTH,ICON_HEIGHT,ICON_SIZE);
```

```
            JLabel label = new JLabel(MyRect);

            // add 3 buttons for the label
            JButton Buttons[] = new JButton[3];
            for(int i = 0; i < 3; i++){
                final Color c = myframe.Colors[i];
                Buttons[i] = myframe.createButton(i, MyRect, label);
            }

            // add buttons and labels
            frame.add(label);
            for(int i = 0; i < 3; i++){
                frame.add(Buttons[i]);
            }

            frame.pack();   // sets smallest size to display components

            frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE); // set on
close condition

        }

        /**
         * This function takes the index of an array. The button with the
related actionlistener is returned
         * @precondition , a MyRect and Jlabel must be instandtiated.
         * @postfondition , a JButton with a ActionListener attached must be
returned.
         * @param i , i determines which color is being set to actionlistener
         * @return , returns a JButton object to be stored in button array
         */
        public JButton createButton(int i, Rectangle MyRect, JLabel label){
            String colorname = "";
            final Color colorchoice = Colors[i];
            if(i == 0) {
                colorname = "Red";
            }
            else if(i == 1) {
                colorname = "Blue";
            }
            else{
                colorname = "Yellow";
            }
            JButton newButton = new JButton(colorname);
            // attach Action listener to newly created button
            newButton.addActionListener(createchangeColorListener(colorchoice,
MyRect, label));
            return newButton;
        }
```

```
        /**
         * @precondition , The color needs to be 1 of the 3 possible colors
in the Colors array, MyRect needs to be defined
         * and JLabel must already be initalized and painted.
         * @postcondition , a ActionListener that repaints the Rect with the
desired Color is returned.
         * @param C the color that the colorlistener will set to the
Rectangle.
         * @param MyRect , the rectangle that is to changeColor.
         * @param label , the label that will be repainted.
         * @return , returns a new ActionListner object with the
actionPreformed attached.
         */
        public static ActionListener createchangeColorListener(Color C,
Rectangle MyRect, JLabel label){
            return new ActionListener()
            {
                public void actionPerformed(ActionEvent e){
                    MyRect.changeColor(C);  // change color
                     label.repaint();
                }
            };
        }

        // pixel measurements
        private static final int ICON_WIDTH = 100;
        private static final int ICON_HEIGHT = 100;
        private static final int ICON_SIZE = 10000;

        public Color[] Colors = {Color.RED, Color.BLUE, Color.YELLOW};
    }
```

4.3

CarShape.java

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

    /**
      This program implements an animation that moves
      a car shape.
    */
    public class AnimationTester
    {
      public static void main(String[] args)
      {
         ShapeIcon.objImplementsMS = new ArrayList<>();
```

```
            JFrame frame = new JFrame("My frame");
            // created a JPanel which is a simpler item than a JFrame
            JPanel panel = new JPanel();

            MovableShape shape[] = new MovableShape[4];
            for(int i = 0; i < 4; i++) {
               final CarShape tempCarShape = new CarShape(0, i, CAR_WIDTH);
               shape[i] = tempCarShape;
            }

            ShapeIcon icon[] = new ShapeIcon[4];
            for(int i = 0; i < shape.length; i++){
               final ShapeIcon tempShapeIcon = new ShapeIcon(shape[i],
ICON_WIDTH, ICON_HEIGHT);
               icon[i] = tempShapeIcon;
            }

            JLabel label[] = new JLabel[4];
            for(int i = 0; i < shape.length; i++){
               final JLabel templabel = new JLabel(icon[i]);
               label[i] = templabel;
               panel.add(label[i]);
            }

            System.out.println("There are " + ShapeIcon.objImplementsMS.size() +
                    " classes initialized that implement MovableShape. There
addresses are:");
            for( int i = 0; i < ShapeIcon.objImplementsMS.size(); i++) {
               System.out.println(ShapeIcon.objImplementsMS.get(i));
            }
            // set the layout of the panel to a BoxLayout that is formulated
for change of Y axis
            panel.setLayout(new BoxLayout(panel,BoxLayout.Y_AXIS));

            // add the panel to the frame
            frame.add(panel);
            frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
            frame.pack();
            frame.setVisible(true);

            final int DELAY = 100;
            // Milliseconds between timer ticks
            Timer t = new Timer(DELAY, new
               ActionListener()
               {
                  public void actionPerformed(ActionEvent event)
                  {
                     for(int i = 0; i < shape.length; i++){
                        shape[i].translate(3, 0);
```

```
                label[i].repaint();
            }

         }
      });
      t.start();
   }

   private static final int ICON_WIDTH = 400;
   private static final int ICON_HEIGHT = 100;
   private static final int CAR_WIDTH = 100;
}
```

MovableShape.java

```
import java.awt.*;

/**
   A shape that can be moved around.
*/
public interface MovableShape
{
   /**
      Draws the shape.
      @param g2 the graphics context
   */
   void draw(Graphics2D g2);
   /**
      Moves the shape 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);
}
```

ShapeIcon.java

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

/**
   An icon that contains a moveable shape.
*/
public class ShapeIcon implements Icon
{
   public ShapeIcon(MovableShape shape,
      int width, int height)
   {
      this.shape = shape;
      this.width = width;
      this.height = height;
      objImplementsMS.add(shape);
```

```
   }

   public int getIconWidth()
   {
      return width;
   }

   public int getIconHeight()
   {
      return height;
   }

   public void paintIcon(Component c, Graphics g, int x, int y)
   {
      Graphics2D g2 = (Graphics2D) g;
      shape.draw(g2);
   }

   // get the other objects that impelment MovableShape

   private int width;
   private int height;
   private MovableShape shape;

   public static ArrayList<MovableShape> objImplementsMS;
}
```

AnimationTester.java

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

/**
   An icon that contains a moveable shape.
*/
public class ShapeIcon implements Icon
{
   public ShapeIcon(MovableShape shape,
      int width, int height)
   {
      this.shape = shape;
      this.width = width;
      this.height = height;
      objImplementsMS.add(shape);
   }

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

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

  public void paintIcon(Component c, Graphics g, int x, int y)
  {
     Graphics2D g2 = (Graphics2D) g;
     shape.draw(g2);
  }

  // get the other objects that impelment MovableShape

  private int width;
  private int height;
  private MovableShape shape;

  public static ArrayList<MovableShape> objImplementsMS;
}
```