Autoren: Marius Birk
Pieter Vogt
Tutor: Florian Brandt

Abgabe: 23.06.2020, 12:00 Uhr

Smileys:

A1	A2	A3	∑

Objektorientierte Modellierung und Programmierung
Abgabe Uebungsblatt Nr.09

(Alle allgemeinen Definitionen aus der Vorlesung haben in diesem Dokument bestand, es sei den sie erhalten eine explizit andere Definition.)

Aufgabe 1

```java
package sample;
import javafx.application.Application;
import javafx.beans.value.ChangeListener;
import javafx.collections.ObservableList;
import javafx.fxml.FXMLLoader;
import javafx.geometry.Insets;
import javafx.scene.Parent;
import javafx.scene.Scene;
import javafx.scene.layout.Background;
import javafx.scene.layout.BackgroundFill;
import javafx.scene.layout.CornerRadii;
import javafx.scene.paint.Color;
import javafx.stage.Stage;
import javafx.scene.shape.Rectangle;
import javafx.scene.control.Slider;
import javafx.scene.layout.VBox;

public class Main extends Application {

    @Override
    public void start(Stage primaryStage) throws Exception{
        Parent root = FXMLLoader.load(getClass().getResource(
           "sample.fxml"));
        //Initialize Components
        Rectangle rect = new Rectangle();
        Slider red = new Slider();
        Slider green = new Slider();
        Slider blue = new Slider();
        VBox vBox = new VBox();
        ObservableList list = vBox.getChildren();

        rect.setHeight(160);
        rect.setWidth(300);

        red.setMin(0);
        red.setMax(255);
        red.setValue(0);
        red.setShowTickLabels(true);
        red.setShowTickMarks(true);
        red.setMajorTickUnit(25);
        red.setMinorTickCount(5);
        red.setBlockIncrement(25);
        red.setBackground(new Background(new BackgroundFill(
           Color.RED, CornerRadii.EMPTY, Insets.EMPTY)));

        green.setMin(0);
        green.setMax(255);
        green.setValue(0);
        green.setShowTickLabels(true);
        green.setShowTickMarks(true);
        green.setMajorTickUnit(25);
        green.setMinorTickCount(5);
        green.setBlockIncrement(25);
        green.setBackground(new Background(new BackgroundFill
           (Color.GREEN, CornerRadii.EMPTY, Insets.EMPTY)));

        blue.setMin(0);
        blue.setMax(255);
        blue.setValue(0);
        blue.setShowTickLabels(true);
        blue.setShowTickMarks(true);
        blue.setMajorTickUnit(25);
        blue.setMinorTickCount(5);
        blue.setBlockIncrement(25);
        blue.setBackground(new Background(new BackgroundFill(
           Color.BLUE, CornerRadii.EMPTY, Insets.EMPTY)));

        ChangeListener<Object> updateListener = (obs,
           oldValue, newValue) -> {
            int cRed   = (int) red.getValue();
            int cGreen   = (int) green.getValue();
            int cBlue   = (int) blue.getValue();
            rect.setFill(Color.rgb(cRed, cGreen, cBlue));
        };

        red.valueProperty().addListener(updateListener);
        green.valueProperty().addListener(updateListener);
        blue.valueProperty().addListener(updateListener);

        vBox.setSpacing(10);
        vBox.setMargin(rect, new Insets(20,20,20,20));
        vBox.setMargin(red, new Insets(20,20,20,20));
        vBox.setMargin(green, new Insets(20,20,20,20));
        vBox.setMargin(blue, new Insets(20,20,20,20));
```

```java
        list.addAll(rect, red, green, blue);

        Scene scene = new Scene(vBox);
        primaryStage.setTitle("Color-Mixer");
        primaryStage.setScene(scene);
        primaryStage.show();
    }


    public static void main(String[] args) {
        launch(args);
    }
}
```

Aufgabe 2

```java
 import javafx.application.Application;
import javafx.event.EventHandler;
import javafx.scene.Scene;
import javafx.scene.layout.GridPane;
import javafx.scene.paint.Color;
import javafx.scene.shape.Rectangle;
import javafx.scene.shape.StrokeType;
import javafx.stage.Stage;



public class Lights extends Application {

    public static void main(String[] args) {
        launch(args);
    }
    static int randomNumber(){
        int size = (int)(Math.random()*10);

        while (size < 2){
            size = (int)(Math.random()*10);
        }
        return size;
    }

    @Override
    public void start(Stage primaryStage) {
        int size = randomNumber();
        Rectangle[][] arrRect = new Rectangle[size][size];
        for(int j= 0; j< arrRect.length;j++){
            for(int i = 0; i < arrRect.length; i++)
            {
                Rectangle rect = new Rectangle();
                rect.setFill(Color.WHITE);
                rect.setWidth(100);
                rect.setHeight(100);
                rect.setStrokeType(StrokeType.INSIDE);
                rect.setStroke(Color.BLACK);
                arrRect[i][j] = rect;
            }
        }

        GridPane grid = new GridPane();
        for(int j=0; j<arrRect.length;j++){
            for(int i = 0; i<arrRect.length;i++){
                grid.add(arrRect[j][i], j, i);
            }
        }
        for(int j =0; j<arrRect.length;j++){
            for(int i = 0; i<arrRect.length;i++){
                int finalI = i;
                int finalJ = j;
                arrRect[i][j].setOnMouseClicked(new
                   EventHandler<javafx.scene.input.MouseEvent
                   >() {
                    @Override
                    public void handle(javafx.scene.input.
                       MouseEvent mouseEvent) {
                        if (arrRect[finalI][finalJ].getFill()
                            == Color.YELLOW) {
                            arrRect[finalI][finalJ].setFill(
                               Color.WHITE);

                        } else {
                            arrRect[finalI][finalJ].setFill(
                               Color.YELLOW);
                        }
                        try {
                            if (arrRect[finalI][finalJ + 1].
                               getFill() == Color.YELLOW) {
                                arrRect[finalI][finalJ + 1].
                                   setFill(Color.WHITE);
                            } else {
                                arrRect[finalI][finalJ + 1].
                                   setFill(Color.YELLOW);
                            }
                            if (arrRect[finalI + 1][finalJ].
                               getFill() == Color.YELLOW) {
                                arrRect[finalI + 1][finalJ].
                                   setFill(Color.WHITE);
                            } else {
```

```java
                                arrRect[finalI + 1][finalJ].
                                   setFill(Color.YELLOW);
                            }
                        } catch (
                           ArrayIndexOutOfBoundsException e)
                           {

                        }
                        if (arrRect[finalI][finalJ - 1].
                           getFill() == Color.YELLOW) {
                            arrRect[finalI][finalJ - 1].
                               setFill(Color.WHITE);
                        } else {
                            arrRect[finalI][finalJ - 1].
                               setFill(Color.YELLOW);
                        }
                        if (finalI > 1) {
                            if (arrRect[finalI - 1][finalJ].
                               getFill() == Color.YELLOW) {
                                arrRect[finalI - 1][finalJ].
                                   setFill(Color.WHITE);
                            } else {
                                arrRect[finalI - 1][finalJ].
                                   setFill(Color.YELLOW);
                            }
                        }
                    }
                });
            }
        }


        Scene scene = new Scene(grid, 200, 100);

        primaryStage.setHeight((double) size*arrRect[0][0].
           getHeight()+60);
        primaryStage.setWidth((double) size*arrRect[0][0].
           getWidth()+40);
        primaryStage.setTitle("Lights");
        primaryStage.setScene(scene);
        primaryStage.show();
    }
}
```

Aufgabe 3

Captain

```java
public abstract class Captain {

 protected Ship ship;

 public Captain(Ship ship) {
  super();
  this.ship = ship;
 }

 /**
  * Gibt ein Kommando an das Schiff.
  * Dieses Kommando wird erst auf der Konsole ausgegeben
  * und anschliessend wird die entsprechende Methode des
  * Schiffs aufgerufen.
  */
 public abstract void commandShip();

}
```

Klasse Observable

```java
import java.lang.reflect.Array;
import java.util.ArrayList;

public abstract class Observable implements Observer{
 private ArrayList<Observer> observers;
 public Observable() {
  observers = new ArrayList<>();
 }

 public void addObserver(Observer obs){
  observers.add(obs);
 }
 public void removeObserver(Observer obs){
  observers.remove(obs);
 }

 public void setChanged(ShipEvent what){

 }
 public void clearChanged(){

 }
 public boolean isChanged(){
  return true;
 }
 public void notifyObservers(ShipEvent what){
  for(Observer o : observers){
    o.update(this, what);
   }
  }
  @Override
  public void update(Observable who, ShipEvent what) {
   who.setChanged(what);
  }
 }
```

Klasse Ship

```
 public class Ship extends Observable{
    private ShipEvent what;
    public void setShipEvent(ShipEvent what){
        this.what = what;
        notifyObservers(what);
    }
}
```

Drunken Pirate

```
 import java.beans.PropertyChangeListener;
import java.util.Properties;
import java.util.Random;

public class DrunkenPirate extends Captain{
    private int lastPick=0;
    private boolean cannon = false;
    private boolean sails = false;


    public DrunkenPirate(Ship ship) {
        super(ship);
    }

    @Override
    public void commandShip() {
        int pick = new Random().nextInt(ShipEvent.values().
           length);
        while(pick==lastPick){
            pick = new Random().nextInt(ShipEvent.values().
               length);
        }
        if(ShipEvent.values()[pick].equals(ShipEvent.
           SET_SAILS)){
            if(sails == false){
                System.out.println(ShipEvent.values()[pick]);
                sails= true;
            }
        }else if(ShipEvent.values()[pick].equals(ShipEvent.
           STRIKE_SAILS)){
            if(sails==true){
                System.out.println(ShipEvent.values()[pick]);
                sails = false;
            }
        }else if(ShipEvent.values()[pick].equals(ShipEvent.
           LOAD_CANNONS)){
            if(cannon==false){
                System.out.println(ShipEvent.values()[pick]);
                cannon=true;
            }
        }else if(ShipEvent.values()[pick].equals(ShipEvent.
           FIRE_CANNONS)){
            if(cannon==true){
                System.out.println(ShipEvent.values()[pick]);
                cannon=false;
            }
        }else if(ShipEvent.values()[pick].equals(ShipEvent.
           NO_EVENT)||ShipEvent.values()[pick].equals(
           ShipEvent.TURN_LEFT)||ShipEvent.values()[pick].
           equals(ShipEvent.TURN_RIGHT)){
            System.out.println(ShipEvent.values()[pick]);
        }
        lastPick = pick;
    }

}
```

ShipLog

```
 public class ShipLog implements Observer{
    @Override
    public void update(Observable who, ShipEvent what) {
    }
}
```