LIGHTS OUT

Softwareentwicklung AU09

Appel, Freudenthaler, Langela, Tieber

Inhaltsverzeichnis

Aufgaben

Geschulte Kompetenzen

Anwendung von GIT-Hub, GUI Programmierung (MVC), Verknüpfung von GIT-Hub mit der Eclipse API.
Testen mit Beispielen.

Simple Aufgaben

Erstellen von GIT-Hub repository, Einteilung der Gruppen

Unsere Ziele waren:
Kreieren von einem Simplen „lights out“ Spiel mit Hilfe von Java.
Erstellen von einer Sinnvollen GUI. Diese GUI soll ein Feld von 7 Buttons mal 7 Buttons erstellen. Nur 5 mal 5 davon sind sichtbar.
Wenn die inneren 5 mal 5 Felder Schwarz sind soll eine Ausgabe den Sieg über das Spiel mitteilen.
Beim Start des Programmes sollen jedes Mal zufällige Felder gelb gefärbt sein.
Beim Druck auf einen Button soll das Feld sowie die 4 Felder darum den Zustand ändern.

Teams

Bilde ein Team mit 3-4 Personen. Jedes Teammitglied hat eine Individuelle Aufgabe, jeder hat seinen Code zu dokumentieren und zu testen.

Notengebung

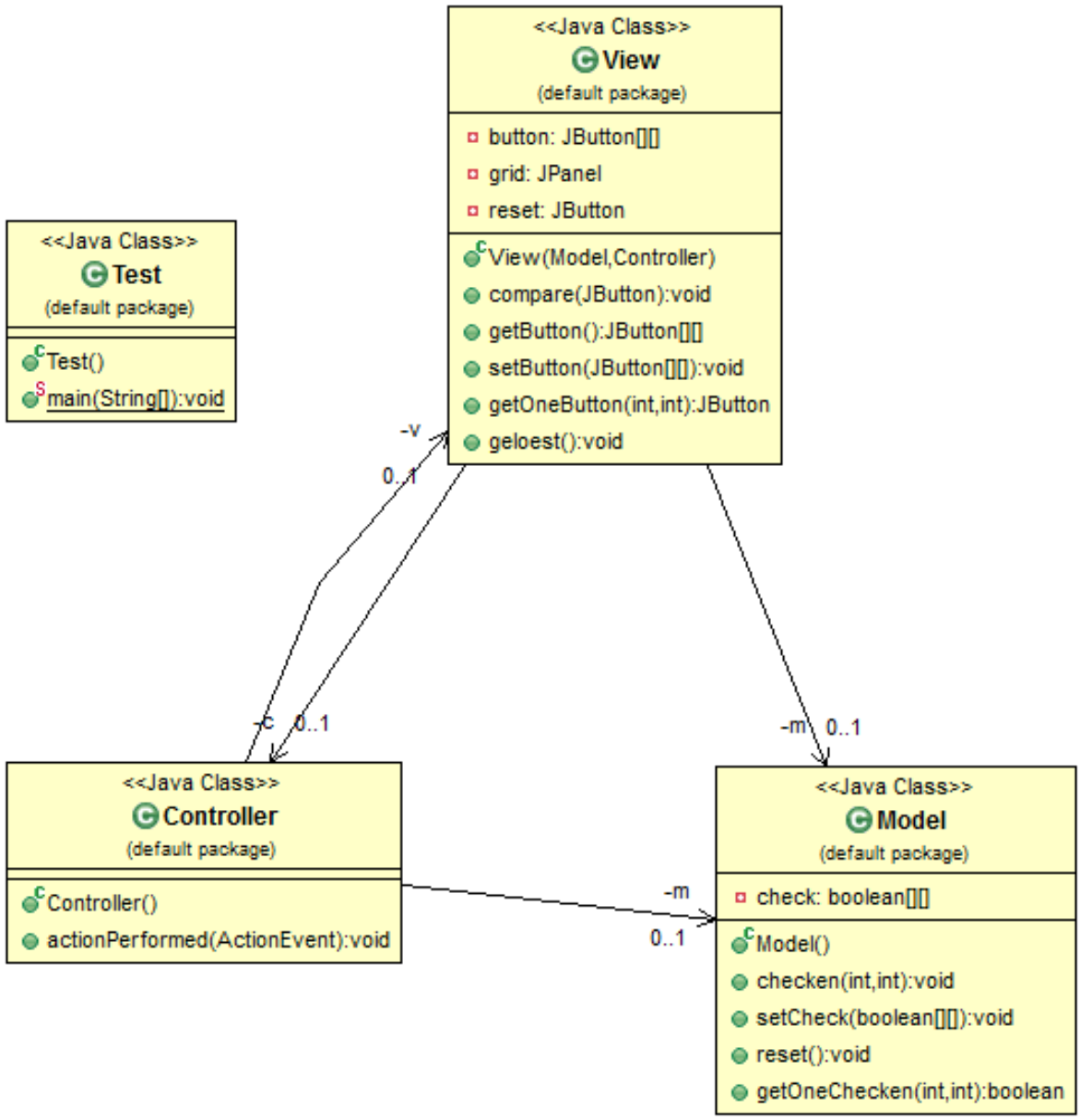
Jedes Team wird als Gesamtbild bewertet. Trotzdem hat jedes Teammitglied seine Teilnahme zu bestätigen (siehe GIT-Hub log).
Teams die keine gute zusammen Arbeit hatten werden anders benotet.

Abgabe

Jede Gruppe braucht ihre eigene Lösung. Gleiche Lösungen werden einen nicht gewertet oder mit einen hohen Punkte Abzug „bestraft“.
Verpflichtend für die Gruppenarbeit ist, ein Protokoll mit vorhandenen UML Diagramm, timetable und einer Test-Dokumentation.
Die Lösung soll auf e-learning als text Document mit einem Link zum GIT-Hub repository abgegeben werden.
Vor der Abgabe kann man die Abgabe so oft bearbeiten so oft man möchte.

	Langela	Freudenthaler	Appel	Tieber	Stunden
Model	0,5	0,5	4	0	
View	1	3	1	0	
Controller	1	0	0	0	
Protokoll	0,5	0	0	3	
Dokumentation	1	1	1	1	
Stunden per Person	4	4,5	6	4	18,5

UML


<<Java Class>>
View
(default package)
button: JButton[][]
grid: JPanel
reset: JButton
View(Model,Controller)
compare(JButton):void
getButton():JButton[][]
setButton(JButton[][]):void
getOneButton(int,int):JButton
geloest():void
<<Java Class>>
Test
(default package)
Test()
main(String[]):void
-v
0..1
-c
0..1
-m
0..1
<<Java Class>>
Controller
(default package)
Controller()
actionPerformed(ActionEvent):void
-m
0..1
<<Java Class>>
Model
(default package)
check: boolean[][]
Model()
checken(int,int):void
setCheck(boolean[][]):void
reset():void
getOneChecken(int,int):boolean

Leistung

Model

Das Model hat eine Boolean Variable die im Konstruktor auf false gesetzt wird:

```
public Model(){
            this.check = new boolean[7][7];
            for(int i = 0; i < 7; i++){
            for(int j = 0; j < 7;++j){
                  this.check[i][j] = false;
            }
            }
      }
```

Die Methode check nimmt zwei Werte und setzt den boolean check dementsprechend auf true oder false.

```
public void checken(int i, int j){
            if(check[i][j] == true){
                  check[i][j] = false;
                  System.out.println("wird false");
            }else{
                  check[i][j] = true;
                  System.out.println("wird true");
            }
      }
```

Die Methode setCheck setzt einen boolean Wert!

```
public void setCheck(boolean[][] check) {
            this.check = check;
      }
```

Die Methode getOneChecken returnt einen boolean Wert der durch die Parameter ausgewählt wird

```
public boolean getOneChecken(int i, int j){
            return this.check[i][j];
      }
```

Controller

Der Konstruktor der Klasse Controller erstellt ein neues View und Model Objekt.

```
public Controller(){
        this.m = new Model();
        this.v = new View(this.m, this);
    }
```

Die Methode actionPerformed erkennt Ereignisse, da sie ein ActionListener ist.

```
@Override
     public void actionPerformed(ActionEvent e) {
           // TODO Auto-generated method stub
        System.out.println("Klicken");
        this.v.compare((JButton)e.getSource());
      }
```

View

Klasse View Extended JFrame

Konstruktor der Klasse Initialisiert 49 Buttons (7 x 7) und setzt sie vorerst alle gelb.

```
this.button = new JButton[7][7];

        for(int i = 0; i < 7; i++){
            for(int j = 0; j < 7; j++){
                button[i][j] = new JButton();
                button[i][j].setBackground(Color.yellow);;
                button[i][j].addActionListener(c);
                this.add(button[i][j]);
            }
        }
```

Weiteres werden die äußeren Button invisible gesetzt. Damit nur noch die Inneren sichtbar sind.

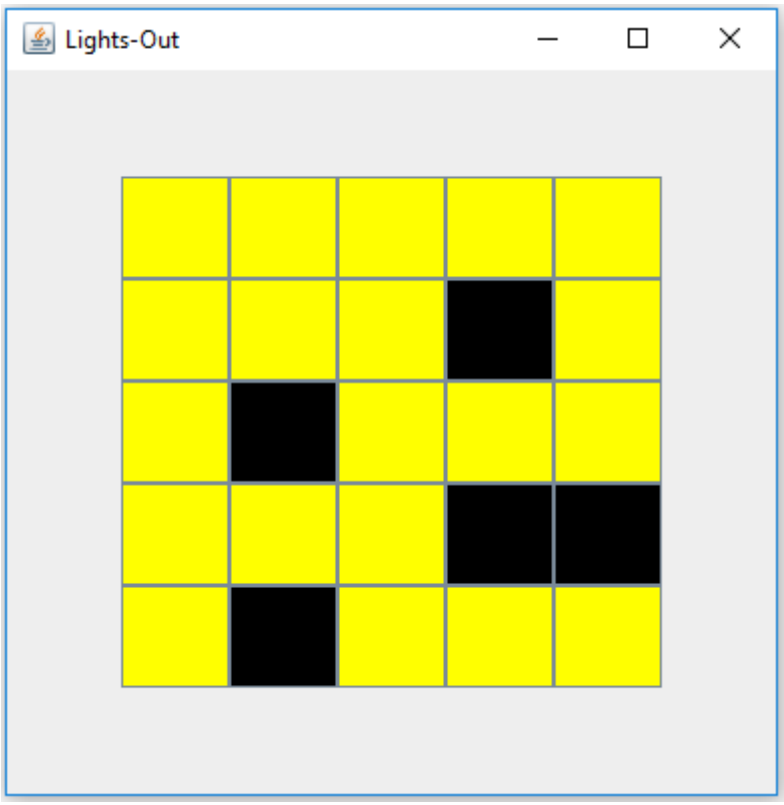
```
this.setLayout(new GridLayout(0,7));
        this.button[0][0].setVisible(false);
        this.button[0][1].setVisible(false);
        this.button[0][2].setVisible(false);
        this.button[0][3].setVisible(false);
        this.button[0][4].setVisible(false);
        this.button[0][5].setVisible(false);
        this.button[0][6].setVisible(false);

        this.button[1][0].setVisible(false);
        this.button[2][0].setVisible(false);
        this.button[3][0].setVisible(false);
        this.button[4][0].setVisible(false);
        this.button[5][0].setVisible(false);
        this.button[6][0].setVisible(false);
```

```
        this.button[1][6].setVisible(false);
        this.button[2][6].setVisible(false);
        this.button[3][6].setVisible(false);
        this.button[4][6].setVisible(false);
        this.button[5][6].setVisible(false);
        this.button[6][6].setVisible(false);

        this.button[6][1].setVisible(false);
        this.button[6][2].setVisible(false);
        this.button[6][3].setVisible(false);
        this.button[6][4].setVisible(false);
        this.button[6][5].setVisible(false);
```

Zuletzt werden im Konstruktor 4 – 8 Buttons schwarz gesetzt.

```
for(int i = 0; i <= (int)(Math.random()*(8 - 4)+ 4);i++){
           int x = (int)(Math.random()*(6 - 2)+ 2);
           int y = (int)(Math.random()*(6 - 2)+ 2);
           if(this.m.getOneChecken(x,y) == false){
                 this.button[x][y].setBackground(Color.black);;
                 this.m.checken(x, y);
                 System.out.println("Checken");
                 System.out.println();

           }
        }
```

Schlussendlich sieht das Spielfeld dann so aus:



Ist das Spiel gelöst wird dies in einer Meldung ausgegeben:

```
public void geloest(){
      if(this.button[1][1].getBackground() == Color.black &&
this.button[1][2].getBackground() == Color.black &&
this.button[1][3].getBackground() == Color.black &&
this.button[1][4].getBackground() == Color.black &&
this.button[1][5].getBackground() == Color.black &&
this.button[2][1].getBackground() == Color.black &&
this.button[2][2].getBackground() == Color.black &&
this.button[2][3].getBackground() == Color.black &&
this.button[2][4].getBackground() == Color.black &&
this.button[2][5].getBackground() == Color.black &&
this.button[3][1].getBackground() == Color.black &&
this.button[3][2].getBackground() == Color.black &&
this.button[3][3].getBackground() == Color.black &&
this.button[3][4].getBackground() == Color.black &&
this.button[3][5].getBackground() == Color.black &&
this.button[4][1].getBackground() == Color.black &&
this.button[4][2].getBackground() == Color.black &&
this.button[4][3].getBackground() == Color.black &&
this.button[4][4].getBackground() == Color.black &&
this.button[4][5].getBackground() == Color.black &&
this.button[5][1].getBackground() == Color.black &&
this.button[5][2].getBackground() == Color.black &&
this.button[5][3].getBackground() == Color.black &&
```

```
this.button[5][4].getBackground() == Color.black &&
this.button[5][5].getBackground() == Color.black)
      {
            System.out.println("geloest");
            JOptionPane.showMessageDialog(null, "Geloest am been!
            Krah Krah"); }
```