Grundlagen der Informatik 4 (GIT IV) Übung

Bearbeited von:

Name	Matrikelnummer	Nummer des Übungsblatts
Maximilian Bradstädter	1007486	9

Aufgabe 1:

- Abstraktion
  - Funktionen wir start, .setTitle etc. werden verwendet um komplexere sachverhalte (hier das backend von javafx) weg zu abstrahieren.
- Vererbung
  - Vererbung wird hier zum beispiel für ColorNeighborsButton verwendet, welcher eine Unterklasse von ColorNeighborsButton ist.
- Kapselung/Geheimnisprinzip
  - Hier wird Kapselung/Geheimnisprinzip genutzt, um z.B.: die implemetierung von der färbung der benachbarten tiles durch die methode processNeighbors zu verbärgen.
- Polymorphismus
  - processNeighbors von ColorNeighborsButton.

Aufgabe 2:

```
import java.util.ArrayList;


import javafx.application.Application;

import javafx.scene.Scene;

import javafx.scene.control.Button;

import javafx.scene.layout.GridPane;

import javafx.stage.Stage;
```

```
public class TileBasedApplication extends Application {

  private static final int GRID_SIZE = 20;
  private Button[][] gridButtons = new Button[GRID_SIZE][GRID_SIZE];

  abstract class ColorNeighborsButton extends Button {
    protected static final int TILE_SIZE = 30;

    public ColorNeighborsButton(String text) {
      super(text);
      this.setMinSize(TILE_SIZE, TILE_SIZE);
      this.setOnAction(e -> processNeighbors(GridPane.getRowIndex(this),
GridPane.getColumnIndex(this)));
    }

    abstract void processNeighbors(int i, int j);

    protected void process(int i, int j, String color) {
      for (int di = -1; di <= 1; di++) {
        for (int dj = -1; dj <= 1; dj++) {
          if (di == 0 && dj == 0)
            continue;

          int ni = i + di;
          int nj = j + dj;
          if (ni >= 0 && ni < GRID_SIZE && nj >= 0 && nj < GRID_SIZE) {
            Button neighbor = gridButtons[ni][nj];
            neighbor.setStyle("-fx-background-color: " + color + ";");
          }
        }
      }
    }
  }

  class RedButton extends ColorNeighborsButton {
    public RedButton() {
      super("r");
    }

    @Override
    void processNeighbors(int i, int j) {
      process(i, j, "red");
    }
  }

  class GreenButton extends ColorNeighborsButton {
    public GreenButton() {
      super("g");
    }

    @Override
    void processNeighbors(int i, int j) {
      process(i, j, "green");
    }
  }

  class RandomColorButton extends ColorNeighborsButton {
    public RandomColorButton() {
      super("P");
    }

    @Override
    void processNeighbors(int i, int j) {
      ArrayList<String> colors = new ArrayList<>();
      colors.add("blue");
      colors.add("yellow");
      colors.add("black");
      colors.add("orange");
      colors.add("brown");

      int randomNumber = (int) (Math.random() * colors.size());
      process(i, j, colors.get(randomNumber));
    }
  }

  @Override
  public void start(Stage primaryStage) {
```

```
GridPane grid = new GridPane();

for (int i = 0; i < GRID_SIZE; i++) {
  for (int j = 0; j < GRID_SIZE; j++) {
    // Generiert eine Zufallszahl zwischen 0 und 3
    int randomNumber = (int) (Math.random() * 3);

    ColorNeighborsButton button = null;

    switch (randomNumber) {
      case 0:
        button = new RedButton();
        break;
      case 1:
        button = new GreenButton();
        break;
      case 2:
        button = new RandomColorButton();
        break;
      default:
        throw new AssertionError();
    }

    gridButtons[i][j] = button;
    grid.add(button, j, i);
  }
```

```
    }

    Scene scene = new Scene(grid, 600, 600);
    primaryStage.setTitle("Tile-Based Application");
    primaryStage.setScene(scene);
    primaryStage.show();
  }

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