Programmaufteilung

Es ist darauf zu achten, dass zwischen Oberfläche und Programmlogik sauber getrennt wird. Dafür sollen auf oberster Ebene zwei Pakete "gui" und "logic" angelegt werden (s.a. **Tipps/Pakete**). Die Trennung von GUI und Logik ermöglicht ein Redesign des Erscheinungsbilds (welches eventuell von anderen Projektteilnehmern umgesetzt wird), ohne Änderungen in der Logik vornehmen zu müssen. So sollten z. B. andere Bilder, andere Komponenten (die vielleicht ein dreidimensional wirkendes Bild visualisieren) oder sogar andere Grafikbibliotheken (Swing oder etwas ganz Neues) eingesetzt werden können, ohne etwas an Objekten der Spiellogik zu ändern. Bilder sind also in einem Unterpaket von "gui" unterzubringen. Zum Erscheinungsbild gehört auch die Sprache, in der dem Benutzer etwas mitgeteilt wird. Textuelle Ausgaben sind also unbedingt im Grafikpaket unterzubringen.

Im FXML-Controller sollen nur Routinen zu finden sein, die etwas mit der direkten Oberflächensteuerung zu tun haben. Alle anderen Routinen sind in entsprechende andere Klassen auszulagern. Daher ist in diesem Projekt vorgeschrieben, GUIConnector, JavaFXGUI und FakeGUI zu implementieren.

Als erster Ansatz ist es sinnvoll, eine Klasse **JavaFXGUI** zu erstellen, die im Konstruktor alle Komponenten erhält, die sie aktualisieren können soll (z.B. diverse Bilder für Karten, Label oder auch ganze Panes). Der FXMLDocumentController erzeugt also eine Instanz von JavaFXGUI und gibt sie an die Spiellogik. Die Methoden der JavaFXGUI ermöglichen der Logik, die Ausgabe GUI zu ändern (z.B. Methode showCards(Player player1) zeigt die Karten eines Spielers an, indem die entsprechenden ImageViews aktualisiert werden). Die Klasse mit der Spiellogik erhält im Konstruktor die JavaFXGUI, so dass deren Methoden ausgeführt werden können.

Zum Testen ist es sinnvoll, keine grafische Oberfläche aufbauen zu müssen. Dafür bietet es sich an, ein Interface **GUIConnector** mit allen für die JavaFXGUI notwendigen Methoden zu gestalten und dieses von der Klasse **JavaFXGUI** implementieren zu lassen. Außerdem soll das Interface von einer Klasse **FakeGUI** implementiert werden, die in allen publizierten Methoden nichts ausführt. In der Spiellogik muss jetzt statt der konkreten JavaFXGUI an allen Stellen das Interface GUIConnector genannt werden. So kann die Spiellogik, die eine FakeGUI übergeben bekommt, alle Anzeigemethoden ausführen, ohne dass grafische Komponenten vorhanden sein müssen.

Der FXMLDocumentController erzeugt also eine Instanz von JavaFXGUI, um sie bei der Instanziierung der Logik zu übergeben, die Testumgebung erzeugt eine Instanz von FakeGUI, mit der die Logik erzeugt und getestet werden kann. Die Spiellogik erwartet lediglich eine Klasse, die den GUIConnector implementiert.

```
class GameLogic {
  GUIConnector gui;
  int score;

  public GameLogic(GUIConnector gui) {
    this.gui = gui;
  }

  public void setScore(int score) {
    this.score = score;
    gui.showScore(score);
  }
}
```

```
interface GUIConnector {
//enthält kein JavaFX!
 void showScore(int score);
}
```

```
class FakeGUI implements GUIConnector {
  void showScore(int score) {}
}
```

```
class JavaFXGUI implements GUIConnector {
  Label this.lblScore;
  JavaFXGUI(Label lblScore) {
    this.lblScore = lblScore;
  }
  void showScore(int score) {
    lblScore.setText(String.valueOf(score));
  }
}
```

Der GUIConnector gehört in das Logik-Paket, die JavaFXGUI in das GUI-Paket und die FakeGUI in das Paket der Tests. Aus dem Logik-Paket darf ausschließlich das Interface GUIConnector genutzt werden.

Die Trennung von Logik und Oberfläche ist korrekt, wenn

- keine Klasse im Logik-Paket eine Klasse aus dem GUI-Paket importiert oder nutzt,
- im Logik-Paket keine Ausgabetexte erstellt werden, die bei einer Umstellung auf eine andere Sprache übersetzt werden müssten,
- nirgendwo Klassen aus "awt" oder "swing" benutzt oder importiert werden.

Logik

Legt die Logik universell an, so dass Spiele mit weniger Spielern/Spielsteinen/Karten/Feldern gespielt werden können. Der Standardkonstruktor sollte hierfür einen Konstruktor mit Parametern aufrufen und die Parameter aus privaten Konstanten entnehmen. So könnt Ihr überschaubare Spiele testen. Die GUI darf trotzdem davon ausgehen, dass die Konstanten in der Logik für sie passend gesetzt sind, muss also nicht variabler sein, als die Spielregeln verlangen.

GUI

- Lasst die GUI so aussehen, wie Ihr es von einem Standardprogramm erwartet. Ermöglicht also ein Verändern der Formulargröße, verwendet also z.B.
  - bei sich ausschließenden Auswahlmöglichkeiten RadioButtons,
  - bei an-/auszuschaltenden Optionen Checkboxen,
  - bei Werten, die innerhalb eines Bereichs einzustellen sind, Schieberegler (Slider).
- Nicht erwartungskonform sind also beispielsweise Obermenüpunkte ohne Unterpunkte, eine ComboBox als Untermenüpunkt, Buttons als Ein-/Ausschalter (hiermit sollte immer irgendeine Aktion ausgelöst werden).

Tests

- Nutzt keine automatische Testgenerierung, da hiermit stumpf jede Methode einen Test generiert bekommt, statt dass die relevanten Methoden bzw. Situationen sinnvoll getestet werden.
- Getter und Setter brauchen nicht getestet werden, auch wenn bei Settern ein Test für nicht erlaubte Werte sinnvoll wäre.
- Testet stattdessen Spielsituationen: Generiert eine Spielsituation (es kann nur für diesen Zweck ein package-private Konstruktor geschrieben werden, der eine konkrete Spielsituation mitten im Spiel aufnimmt), führt einen Zug aus und prüft dann, ob die Spielsituation mit Eurer Erwartung übereinstimmt (ist der Spieler wie erwartet am Zug, sieht das Spielbrett wie erwartet aus). Für einen "normalen" Zug und für jeden Zug, der zu einer Sonderregelung führt, muss mindestens ein Test existieren.
- Entsteht die Testklasse in einem gleichnamigen Paket wie die zu testende Klasse, kann in ihr auf alle im selben Paket sichtbaren Elemente zugegriffen werden.

Code, sofern er schon implementiert wird:

- Versucht die Verwendung von **Klassenvariablen** so weit wie möglich einzuschränken. Setzt keine Klassenvariablen ein, die Ihr aus anderen Zuständen (z.B. Arraylänge) ermitteln könnt. Und versucht, wenn Ihr Klassenvariablen deklariert habt, so wenig wie möglich aus anderen Klassen auf diese Variablen zuzugreifen (auch nicht über Getter), sondern sie stattdessen an die entsprechenden Prozeduren und Funktionen als Parameter zu übergeben.
- Schränkt jede **Sichtbarkeit** so weit wie möglich ein (Attribute immer private). Getter werden nur public angeboten, wenn unbedingt notwendig. Setter werden nur public angeboten, wenn das hiermit gesetzte Attribut nach Erstellung des Objekts verändert werden können muss.
- Setzt keine **magic numbers** ein, sondern verwendet Konstanten!
- Erstellt für die Spiellogik einen **Konstruktor**, der eine gesamte Spielsituation in für den Menschen gut lesbarer Form aufnehmen kann. So kann beispielsweise bei einem Aufruf von
  ```
  new TicTacToeGame("_ x _" +
                    "o _ x" +
                    "o _ _")
  ```
  sofort die Spielsituation erfasst werden und bereits getestet werden, ob ein Spieler gewonnen hat. Der übergebene String muss lediglich auf das Spielfeld übersetzt werden.
- Erstellt **mehrere Konstruktoren**, die sich gegenseitig aufrufen. Gibt es für das TicTacToe-Spiel noch einen Konstruktor, der außer dem Spielfeldstring den aktuellen Spieler annimmt, so kann beispielsweise der optimale Zug ermittelt werden.
- Entfernt ungenutzten **automatisch generierten Code**.
- Setzt als **Variablentyp** einen möglichst allgemeinen ein (z.B. GUIConnector statt JavaFXGUI oder auch List statt ArrayList).

Kontakt FH Wedel

Staatlich anerkannte
Fachhochschule Wedel
Gemeinnützige Schul-
gesellschaft mbH
Feldstraße 143
D - 22880 Wedel
Tel.: +49 (0)4103 - 8048-0
Fax: +49 (0)4103 - 8048-39
E-Mail: sekretariat@fh-wedel.de