Tipps für die Langzeitklausur

AvHG, Inf, My

Don‘t panic!

- Top-Down entwickeln: Erst mal so tun als ob es eine Methode gäbe, die genau das tut, was ihr braucht. Etwa `wareBestellen()`. Und erst anschließend diese Methode konkret implementieren.
- Auslagern des Programme-Codes zur Erzeugung der Benutzeroberfläche in eine eigene Methode. Etwa `createGUI()`.
  Der Konstruktor eurer eigenen Klasse besteht dann nur noch aus dem Aufruf dieser Methode und eventuell einiger (weniger) weiterer Methoden, die direkt beim Programmstart ausgeführt werden sollen.
- Keine Anwendungslogik in `createGUI()`. In den Eventhandlern nur Methodenaufrufe platzieren. Die eigentliche Funktionalität/Anwendungslogik gehört dann in diese Methoden.
- Sinnvolle („sprechende“) Namen für Variablen und Methoden wählen! Auch für die Komponenten der Benutzeroberfläche, die ihr mit dem WindowBuilder erstellt.
- Probleme nur einmal lösen! Bei etwas größeren Programmen kann es durchaus passieren, dass ihr ähnlichen Probleme mehrmals begegnet. Beispielsweise das Anzeigen von Ergebnissen einer Datenbankabfrage in einer `JList`-Komponente. Dazu habt ihr vielleicht eine Methode `ergebnisseAnzeigen()` programmiert. Später sollt ihr dann eine Methode zum Löschen eines Datensatzes programmieren. Natürlich muss anschließend die Liste aktualisiert werden. Da wäre es natürlich klug, die bereits vorhandene Methode `ergebnisseAnzeigen()` einfach erneut aufzurufen, anstatt diese erneut zu programmieren.
- Objekte und Variablen, die nicht nur in einer bestimmten Methode benutzt werden sollen, müssen global deklariert werden. Speziell mit den Komponenten der Benutzeroberfläche kann es euch passieren, dass diese vom WindowBuilder lokal deklariert werden. Mit `Refactor → Convert Local Variable to Field` kann Eclipse die Umwandlung in ein globales Attribut der Klasse für euch übernehmen.
- `JList` Objekte werden erzeugt, indem man zunächst ein `DefaultListModel`-Objekt erzeugt und dieses anschließend an den `JList`-Konstruktor übergibt.
  Sowohl `JList` als auch `DefaultListModel` sollten parameterisiert werden:

  ```
  DefaultListModel<String> termine = new DefaultListModel<String>();
  Jlist<String> listTermine = new Jlist<String>(termine);
  ```

  Das gleiche gilt für `JComboBox` und `DefaultComboBoxModel`.
- Nicht vor der großen Aufgabe erschrecken. Auch nicht vor großen Teilaufgaben. Immer Schritt für Schritt!
- Nicht an einzelnen Teilaufgaben verzweifeln und dort hängen bleiben. Dann lieber dort abbrechen und mit der nächsten Teilaufgabe fortfahren. Evtl. lässt sich statt der (schwierigen) richtigen Lösung auch eine (leichte) Ersatzlösung implementieren, mit der ihr dann erst mal weiter arbeiten könnt.

  Beispiel: Ihr sollt eine Zahl aus einem längeren String extrahieren (mit Hilfe von String-Methoden). Aber das will euch nicht gelingen. Dann könntet ihr hilfsweise programmieren:

  ```
  int zahl = 13;  //  Hier habe ich abgekürzt, weil es mir zu schwer war …
  ```

  Wenn ihr später noch Zeit habt, könnt ihr es ja noch mal probieren zu einer „richtigen“ Lösung zu kommen.
- Alle SQL Befehle zunächst in einem String abspeichern. Diesen dann zunächst auf der Console mit `System.out.println()` ausgeben lassen und erst anschließend mit `executeQuery()` bzw. `executeUpdate()` an den Datenbankserver zur Ausführung übergeben.

  Beispiel:

  ```
  String sql = "SELECT * FROM tier WHERE name = '" + tfNname.getText();
  sql += "' AND tierart = 'Katze';
  System.out.println(sql);
  ResultSet rs = stmt.executeQuery(sql);
  ```