Abnahme Zwischen- und Hauptprojekt

Prüfung Zwischenprojekt

Bewertung des Praktikums erfolgt durch eine Endabnahme mit einer Gesamtdauer von ca. 90 Minuten.

Die Abnahme des Zwischenprojekts verläuft ganz genauso wie beim Hauptprojekt, mit der Ausnahme das für das Zwischenprojekt keine Note vergeben wird.

Stattdessen wird bei der Abnahme des Zwischenprojekts die individuelle Zulassung der Teilnehmer zum Hauptprojekt geprüft.

Die folgenden Folien beziehen sich auf die Abnahme des Hauptprojekts und gelten damit ganz analog auch für Abnahme des Zwischenprojekts.

Abnahme Hauptprojekt

Die Endabnahme hat eine Gesamtdauer von ca. 90 Minuten.

Präsentation der Gruppenarbeit (max. 25 Minuten)

Jeder Teilnehmer präsentiert in ca. 5 Minuten einen Teil des Projekts; Abwechslung bei Präsentation nach Eurer Wahl.

Vorführung der Software (max. 10 Minuten)

Die Vorführung kann mit der Präsentation vermischt sein.

Befragung

Befragung der einzelnen Teilnehmer zu Details des Projekts, zur Bestimmung der **individuellen** Note.

Prüfungsprotokoll

Offline gespeicherte Videoaufnahme; wird wie eine Klausuren archiviert, Verwendung nur für den Fall möglicher Einsprüche.

Präsentation

Ziel der Projektpräsentation ist, einen **Überblick über die geleistete Arbeit** der Gruppe zu geben. Uns interessieren:

- Wie ist das Programm aufgebaut?
  - Was sind die wichtigsten Klassen und ihre Aufgaben?
  - Wie werden die Daten im Modell gespeichert?
  - Wie funktioniert die Anzeige?
- Viele Teile des Projekts können auf verschiedene Weise gelöst werden. Welche Entscheidungen wurden getroffen?
- Wurden Erweiterungen (evtl. eigene) implementiert?
- Gibt es bekannte Mängel?
- **Wer hat was gemacht?** Wie wurde kommuniziert?

Präsentation

Präsentation soll Zuhörer überzeugen, dass gute Software entwickelt wurde!

Stellt eigene Ideen heraus und erzählt uns nicht, was wir schon kennen (z.B. stumpf die gesamte Spezifikation wiederholen).

Präsentation gibt Möglichkeit, auf besonders interessante oder gelungene Teile der eigenen Arbeit hinzuweisen, z.B.:

- Erklärung eines implementierten Algorithmus
- Einfache Lösung für ein zunächst schwierig erscheinendes Problem
- Elegante Problemlösungen
- **Schöner, eigener Java-Code**

Präsentation

In der Präsentation muss in kurzer Zeit die Arbeit von zwei Monaten erklärt werden.

Empfehlung:

- Die Gesamtheit der Arbeit Überblick-artig umreißen.
  Gegenbeispiel: Endlose Folien zur View-Beschreibung, welche sagen das eine `GridPane` in einer `VBox` in einer `AnchorPane` in einer ...Zzzz...

- Einen interessanten Punkt herausgreifen und im Detail erklären.
  Beispiel: Realisierung unseres flüssigen Zooms zwischen den Ansichten mit `IrgendeineSuperDuperPane` ...

Bewertungskriterien

- Gesamteindruck der entwickelten Software, d.h. Funktionalität, Korrektheit, Vollständigkeit
- Erweiterungen (Anzahl + Schwierigkeitsgrad)
- Umsetzung der Plenumsinhalte
  Beispiele: MVC, Datenstrukturen, Fehlerbehandlung, Vermeidung von Anti-Patterns, Dokumentation, ...
- Lesbarkeit des Codes
- Präsentation
- Antworten in der Befragung

Verbesserungsmöglichkeiten Struktur

- Model-View-Controller (bzw. MVP) umgesetzt?
- Single-Responsibility Principle beachtet?
- Ist der öffentliche Teil der Klassen (`public`) adäquat dokumentiert?
- Ist nur `public`, was auch `public` sein muss?
- Oft gibt es Fehler, die durch Herausgabe von veränderlichen privaten Daten einer Klasse zustande kommen.
  Beispiel:

```
private Map<K, V> map;
public Map<K, V> getMap() {
  return map;
}
```

  Achtung: Ein Aufrufer kann die Map ändern!

Verbesserungsmöglichkeiten Struktur

- Model-View-Controller (bzw. MVP) umgesetzt?
- Single-Responsibility Principle beachtet?
- Ist der öffentliche Teil der Klassen (`public`) adäquat dokumentiert?
- Ist nur `public`, was auch `public` sein muss?
- Oft gibt es Fehler, die durch Herausgabe von veränderlichen privaten Daten einer Klasse zustande kommen.
  Beispiel:

```
private Map<K, V> map;
public Map<K, V> getMap() {
  return Collections.unmodifieableMap(map);
}
```

  Verhindern durch **unveränderliche Objekte**
  oder durch **defensives Kopieren**. (Veränderliche Inhalte?)

Verbesserungsmöglichkeiten Programmcode

- Ist der Code für Menschen leicht lesbar?
- Wurden gute sprechende Namen für Klassen, Methoden usw. verwendet?
- Wurden feste Werte als Konstanten deklariert?
- Werden Strings verwendet, wo Enums angebracht wären?
- Gibt es viel Code, der mit Copy-and-Paste kopiert wurde?
- Gibt es unnötige Redundanz? (Wiederholung ähnlicher Code-Blöcke?)
- Kann man den Code noch vereinfachen?
- Wurde der richtige Typ gewählt?
  Beispiel: Funktioniert Methode `void m(ArrayList<E> l)` wirklich nur für `ArrayList<E>` oder auch für `List<E>`?

Verbesserungsmöglichkeiten Fehlerbehandlung

- Was passiert, wenn der Benutzer alle möglichen unsinnigen Eingaben macht?
- Was passiert, wenn z.B. Level-Dateien nur teilweise gefunden werden?
- Werden Ausnahmen adäquat behandelt?
  D.h. also gerade keine catch-alls wie `catch Exeption e`
- Werden Runtime-Exceptions wie `NullPointerException` abgefangen anstatt die Fehler im Code zu beheben?

Abgabe Zwischen-/Hauptprojekt

Abgabe erfolgt jeweils zum Stichtag über das GIT-Repository.

- Gruppenname muss enthalten sein
- Gruppenbetreuer und Veranstalter (S. Barth, S. Jost) müssen Zugriff auf Repository haben
- Vollständiger Quellcode muss enthalten sein
- **Nur HP:** Präsentation als PDF muss enthalten sein
- **Nur HP:** Lizenz-Datei muss enthalten sein (Dürfen wir Screenshots und/oder Code von Euch auf veröffentlichen?)