Projekt 2

Kontext

Im letzten Semesterdrittel setzen Sie als Gruppenarbeit ein eigenes Projekt um. Die Projektidee dazu wird im "Workshop Projektidee" erarbeitet und evaluiert. Anschliessend wurden in der Projektskizze die Anforderungen und Umfang des Projektes genauer spezifiziert.

Einbettung in die Lernziele von PM2

Die Umsetzung des Projektes trägt zur Erreichung von folgenden Lernzielen von PM2 bei:

- Die Studierenden sind in der Lage ein Softwareprojekt mit vorgegebener Spezifikation im Team fachlich und methodisch korrekt umzusetzen, die Abläufe einzuhalten, sowie Werkzeuge konsequent einzusetzen.
- Sie arbeiten in einem Team aktiv und zielführend zusammen und übernehmen dabei Verantwortung für die Erarbeitung des gemeinsamen Projektes wie auch für den Lernfortschritt aller Teammitglieder.

Auftrag

Setzen Sie das in ihrer Projektidee beschriebene Projekt im vorgegebenen Rahmen um.

Anforderungen

- Die Anwendung muss *mindestens den in der Projektskizze beschriebenen Hauptablauf plus dazu notwendige Nebenabläufe* umsetzen. Die Auswahl der umzusetzenden Abläufe ist mit dem Betreuer abzusprechen.
- Es muss eine Benutzerinteraktion über eine **JavaFX-Benutzerschnittstelle** stattfinden. Das Model-View-Presenter Pattern soll konsequent umgesetzt werden.
- Es können Sprachfeatures bis und mit der aktuellen Java LTS-Version (Long Term Support) verwenden (aktuell Java 21).
- Verwenden Sie weitgehend JDK-interne Komponenten. Externe Libraries sollen auf ein Minimum beschränkt werden. Einfache Libraries wie z.B. für Testing, Ein-/Ausgabe in Dateien/Konsole, Codierung, sowie für grafische Benutzeroberflächen (JavaFX) dürfen eingesetzt werden. Komplexe Frameworks wie z.B. Spring, Hibernate, Game-Engines, Android, etc. dürfen nur mit ausdrücklicher Erlaubnis des Betreuers / der Betreuerin eingesetzt werden. Diese wird in der Regel nur erteilt, wenn alle Teammitglieder bereits mindestens während eines Projektes mit dem Framework gearbeitet haben und somit die Einarbeitung entfällt.
- Als **Architekturdokumentation** muss ein Klassendiagramm erstellt werden. Achten Sie auf einen sinnvollen Detaillierungsgrad und eine korrekte UML-Syntax gemäss Anleitung Klassendiagramme (Link in der PM2 Kursinformation in Moodle). Erläutern Sie zudem stichhaltig und nachvollziehbar, warum Sie diese Architektur gewählt haben. Der Umfang der Erläuterung sollte zirka 0.5 - 1 A4-Seite umfassen. Die Dokumentation muss Teil des Repository sein. Sie kann entweder direkt in der README-Datei enthalten sein oder muss von dieser verlinkt werden.
- Erstellen Sie ein **Testingkonzept**, in welchem Sie identifizieren, welche die zentralen zu testenden Bereiche der Anwendung sind. In der Regel sollte dies die Klassen sein, welche

Domänenlogik (Verarbeitungslogik) enthalten. Überlegen und dokumentieren Sie, welche **Testcases** Sie dazu programmieren wollen.

- Setzen Sie die Testcases entsprechend dem Testingkonzept um. Die Testobjekte müssen isoliert getestet werden. Verwenden Sie dazu **Mocktesting**.
- **Build-Tooling** wird korrekt verwendet. Die Anwendung muss mit dem Befehl `'gradlew run'` gestartet und die Tests mit `'gradlew test'` ausgeführt werden können.
- Die Projektstruktur muss sinnvoll sein und Dateien am richtigen Ort liegen.
- Das **Branching-Model** soll sinnvoll sein und konsequent angewendet werden. Die Spezifikation des Branching-Models oder ein Link darauf muss in der README-Datei enthalten sein.
- Das Projekt wird über **Issues** und **GitHub-Project** verwaltet (Mindestspalten: 'Todo' bzw. 'Backlog', 'Doing' und 'Done'; weitere wie z.B. 'Review' sind zulässig) und GitHub-Issues für die Planung des Vorgehens und Dokumentation der Änderungen und Entscheidungen verwendet. Verwenden Sie, strukturierte Beschreibungen (siehe GitHub Issue Templates) und Labels zur Kategorisierung.
- **Pull-Requests** werden konsequent für Reviews und das Merging der Beiträge verwendet. Verlinken Sie in der README-Datei zwei Pull-Requests, in welchen Reviews diskutiert und Feedback integriert wurde.
- Ihr Projekt muss die im **CleanCode**-Handbuch definierten Regeln der Stufe L1 - L4 erfüllen.
- Fassen Sie ihren zeitlichen Aufwand wöchentlich zusammen. Wird nicht bewertet, kann aber für das Coaching und die Retrospektive nützlich sein.

Zwischenstandsitzung

In der im Semesterplan definierten Woche findet eine Zwischenstandsitzung statt, in welcher die Architekturdokumentation und das Testkonzept vorgestellt wird.

Speziell wird hier angeschaut:

- **Architektur**:
  Formale Korrektheit des Klassendiagramms. Ist es übersichtlich und hat einen sinnvollen Detaillierungsgrad? Sind genau die relevanten Klassen, Beziehungen und Attribute im Diagramm enthalten? Hilft es, die Programmierung zielgerichtet vorzunehmen?
- **Testingkonzept**:
  Überzeugt das Konzept, dass die relevanten Teile der Applikation getestet werden? Ist die Darstellung übersichtlich? Sind die definierten Äquivalenzklassen/Test-Cases sinnvoll, nachvollziehbar und decken alle essenziellen Fälle ab?

Die entsprechenden Dokumente müssen rechtzeitig vor dem Meeting (gemäss Vorgabe der jeweiligen Dozierenden) im Repository bereitstehen und im README verlinkt sein.

Abgabe

Die Abgabe erfolgt in der im Wochenplan definierten Woche, am Folgetag des Unterrichts um 23:59.

- Der erarbeitete Code muss bis zur Frist im Projektrepository auf GitHub hochgeladen sein.
- Die Issues sowie die Dokumentation ihrer Lösung (inkl. Klassendiagramm) sind im GitHub Repository vorhanden und auf der Readme-Seite verlinkt.
- Der letzte Commit im Haupt-Branch (main) vor diesem Zeitpunkt wird als Abgabestand verwendet.

Bewertung

In die Bewertung fliessen die folgenden Kriterien ein:

Allgemeine Anforderungen (all-or-nothing)

Voraussetzung für Punkteerteilung: Das Programm ist lauffähig. Ein nicht lauffähiges Programm erhält die Note 1.

Architektur (20%)

- Architekturdokumentation ist vorhanden, vollständig und konsistent.
- Das Klassendiagramm ist formal korrekt, übersichtlich und hat einen sinnvollen Detaillierungsgrad. → Zwischenstandsitzung.
- Die Klassenstruktur ist sinnvoll, die Koppelung ist niedrig, die Aufteilung der Klassen in Module ist sinnvoll. Vererbung ist sinnvoll eingesetzt. Pattern werden eingehalten.

Software (60%)

- Das Programm besitzt die geplante bzw. mit dem Betreuer / der Betreuerin abgesprochene Funktionalität.
- Sie halten die Vorgaben hinsichtlich einsetzbarer Konstrukte und Clean Code ein und Ihr Code ist sauber dokumentiert.
- Das Testingkonzept ist vorhanden, deckt die relevanten Teile der Anwendung ab und definiert sinnvolle Äquivalenzklassen/Testcases (inkl. Mock-Testing)
  → Zwischenstandsitzung.
- Die Test-Cases sind gemäss Testingkonzept umgesetzt und können erfolgreich ausgeführt werden.

Projekt (20%)

- Ihr Vorgehen war planmässig, Fortschritte und Änderungen wurden strukturiert beschlossen und dokumentiert. Beiträge (Features, Bugs, Verbesserungen) und Entscheidungen wurden über GitHub-Issues verwaltet und dokumentiert.
- Beiträge von Projektmitgliedern wurden einem Review unterzogen und gezielt integriert.
- Korrekte Verwendung von Build-Tooling. Der Build funktioniert IDE unabhängig.
- Die Projektstruktur ist sinnvoll und Dateien liegen am richtigen Ort.
- Das Branching-Modell wurde definiert und korrekt eingehalten.
- Alle Gruppenmitglieder haben gleichermassen Code beigetragen und auf GitHub eingecheckt.

