UEBUNG05: GRUPPENÜBUNG

HINWEISE ZU DEN ÜBUNGEN

- **Weitere Literatur siehe Ende des Dokuments**

  Nehmen Sie sich Zeit, um die Übungsangaben genau und sorgfältig durchzulesen, bevor Sie mit der Bearbeitung beginnen.

- **Schritt-für-Schritt-Ansatz:**

  Sehen Sie sich im **VORHINEIN** die gesamte Übungsanleitung durch.

- **Wichtiger Hinweis für die Abschlussübung:**

  Berücksichtigen Sie in dieser Übung alle Aspekte die Sie in den vorherigen Übungen angewendet haben.

ÜBUNGSANLEITUNG

---

i *Ziel der fünften Übung ist es**, alle gelernten Inhalte gemeinsam im Rahmen eines größeren Beispiels (Queue) umzusetzen.***

⚠ *Sie erstellen ein NEUES Repository und arbeiten gemeinsam in diesem Repository.*

KICKOFF

- Als Ausgangsbasis verwenden Sie das zur Verfügung gestellte Archiv. In diesem Archiv sind einige schlecht kommentierte und teilweise fehlerhafte Java Dateien als Ausgangsbasis vorhanden.
- Ein Teammitglied erstellt **ein neues Repository** auf GitHub und lädt die anderen Gruppenmitglieder als Kollaborator*innen ein.
  - Der Name des Repositories ist nicht vorgegeben, aber der Name sollte Bezug zur Aufgabenstellung haben.

WICHIG: Die folgenden Aufgaben sollen in unterschiedlichen Branches umgesetzt werden und diese Hinweise sind für alle Branches relevant.

- Erstellen Sie neue Commits sobald Sie einen kleinen sinnvollen Teil abgeschlossen haben bzw. die Ergebnisse mit Ihren Kolleg*innen teilen möchten.
- Alle Branches sollen auf den remote Server gepushed werden damit Sie gemeinsam daran arbeiten können.
- Bennen Sie die Branches wie in den Überschriften vorgegeben (z.B. **maven**).
- Nach Fertigstellung eines Branches erstellen Sie einen pull request - akzeptieren Sie diesen nicht selbst – dieser soll von einem anderen Gruppenmitglied geprüft und gemerged werden.
- Manche (Teile) dieser Aufgaben können bzw. sollen Sie
  - parallel umsetzen
  - oder gemeinsam bearbeiten.
- Best Practices beim Verfassen der Dokumentation/git commit Nachrichten berücksichtigen
- Taskliste aktualisieren: Mehr Details dazu siehe weiter unten
- Keine generierten Dateien in der Versionsverwaltung
- Maven Ordnerstruktur

Branch maven

- Erstellen Sie einen Branch maven. Ein Teammitglied erstellt ein neues Maven Projekt im Repository Ordner und integriert die Vorlagedateien (siehe Moodle).
  - Erstellen Sie alle Dateien im Package **at.campus02.bsd**
  - Achten Sie darauf, dass das Projektverzeichnis gleichzeitig das root Verzeichnis des Git Repositories ist (sprich: Der Projektordner ist kein Unterordner im Repository)
  - Erstellen Sie eine erste Version der *.gitignore* Datei damit nur relevante Dateien ins Repository gespeichert werden.

- In weiterer Folge kann es notwendig sein die .gitignore Datei zu erweitern.

- Bedenken Sie die Best Practices zum Thema git Kommentare!

---

i *Versuchen Sie sich fortlaufend zu koordinieren, wann wer welche git Kommandos im Bezug mit dem Remote Repository durchführt. Es erleichtert Ihnen die Zusammenarbeit.*

Branch readme

- Erstellen Sie einen Branch readme und erstellen Sie darin eine Markdown README Datei.
  - Notieren Sie den Projektnamen und die Namen aller Gruppenmitglieder.
  - Befüllen Sie die Datei mit den Inhalten die üblicherweise in solch einer Datei sein sollten. (siehe vorherige Übungen)
  - Nachdem Sie die ganze Übungsanleitung gelesen haben koordinieren Sie sich und erstellen gemeinsam eine **Taskliste** in der Readme.(Markdown Flavor[1]).
    - Erstellen Sie eine Liste aller weiteren Aufgaben und teilen Sie sich die Aufgaben auf. Überlegen Sie sich konkrete Aufgaben auf Basis der Aufgabenstellung – das sollten mehr Tasks als die Anzahl der Branches sein.
    - Nach Erledigung der jeweiligen Aufgaben bearbeiten Sie auch die „Tasks“ im README.
      - „Hakerln“ Sie die Aufgabe ab (im jeweiligen Branch) um Erledigung zu signalisieren (Markdown Flavor[2]).

Branch maven_detail

- Erweitern Sie Ihr **pom.xml** um alle notwendigen Bibliotheken bzw. Konfigurationen um die gesamte Aufgabenstellung umzusetzen.
  - **Jedes Gruppenmitglied soll die eigenen Entwickler*innen Informationen hinzufügen.**

[1] https://help.github.com/en/github/managing-your-work-on-github/about-task-lists
[2] https://help.github.com/en/github/managing-your-work-on-github/about-task-lists

- Ein Teammitglied soll die Adresse des Git Repositiories hinzufügen.
- Fügen Sie die notwendigen Maven Abhängigkeiten hinzu.
- Testen Sie Ihre Konfiguration mit Ihrer Maven Installation über die Kommandozeile – eine große Anzahl an Warnungen oder Fehler sollen beim Ausführen nicht auftreten.

Falls sie maven über IntelliJ ausführen, müssen Sie beachten, dass nicht ihr installiertes Maven sondern eine integrierte IntelliJ Maven Version verwendet wird – diese kann sich bei Versionsunterschieden anders verhalten. Bewertet wird jedoch die Funktionsweise von maven über die Kommandozeile mit der aktuellen maven Version – Sie können auch in IntelliJ Ihre eigene Maven Version auswählen.

Branch bug_hunt

- Im bestehenden Code sind drei **Fehler** – finden Sie diese und korrigieren Sie diese Fehler.

Branch drink

- Erstellen Sie eine Klasse **Cocktail.** Diese soll das Interface **Drink** implementieren.
  - Ein Cocktail kann aus beliebig vielen Zutaten bestehen – Ihr Cocktail soll aus einer beliebigen Menge Liquid Objekten bestehen.
  - Der Konstruktor soll einen Namen (String) bzw. eine Liquid. Liste (List<…>) als Parameter erhalten.

Branch queue

- Erstellen Sie eine Klasse **DrinkQueue** die es ermöglicht alle Getränke, also Klassen die das Interface *Drink* implementieren, in einer Queue zu verwalten.
  - Diese Methode sollte ohne Fehler umgesetzt werden ;)
  - Teilen Sie die durch das Interface vorgegebenen Methoden so auf, dass jedes Gruppenmitglied ungefähr gleich viele Methoden erstellt.
    - Setzten Sie Ihre Teile jeweils in einem eigenen Branch um, indem jedes Gruppenmitglied einen eigenen Branch ala *queue_hinterseer* oder

queue_hofer erstellt. Nach Fertigstellung mergen Sie die eigene Umsetzung in den *queue* Branch zurück.

Branch javadoc

- Erstellen Sie **JavaDoc Kommentare** für die bereits zur Verfügung gestellte Klassen, als auch die neuen Klassen.
- Für eine Übersicht über Javadoc siehe folgende Quelle[3] bzw. folgendes Tutorial[4].
- Bei den Methoden sollen neben den allgemeinen Beschreibungen auch die Annotations **param**, **return** und ggf. **throws** verwendet werden.
- Vergessen Sie nicht die Attribute der Klassen.

Branch tests

- Erstellen Sie **Tests** für die **bestehenden als auch neuen Klassen.**
  - Sie müssen für alle Klassen eine **hundertprozentige** Testabdeckung erreichen.
  - Alle Testdateien sollen nach dem folgenden Namensschema benannt werden – **NameDerKlasseTest** - dazu ein Beispiel:
    - DrinkQueue.java zugehöriger Test: DrinkQueueTest.java
  - Bedenken Sie wo diese Tests im Maven Projekt gespeichert werden sollen.
  - Teilen Sie sich das Testen innerhalb der Gruppe auf.
  - Der Aufruf **mvn test** auf der Kommandozeile muss fehlerfrei ausführbar sein.

Branch double_queue (nur für 3er Gruppen)

- Im Fall einer 3er Gruppe sind folgende weitere Aufgaben umzusetzen:
  - Erstellen Sie eine Queue Version, die es ermöglicht Double Werte in einer Queue zu verwalten – Name der Klasse: **DoubleQueue**.
  - Für diese Implementation müssen ebenso Kommentare und Tests in den anderen Branches erstellt werden.

[3] https://www.oracle.com/technical-resources/articles/java/javadoc-tool.html
[4] https://www.baeldung.com/javadoc

Branch maven_site

- Erstellen Sie eine **maven site** Dokumentation:
  - Integrieren und adaptieren Sie allgemeine Projektinformationen, Teststatistiken (jacoco) und auch die generierte API Dokumentation.
  - Jedes Teammitglied soll eine zusätzliche Markdown Datei hinzufügen, die auf der Hauptseite der maven site Dokumentation verlinkt wird:
    - Benennung: **member1.md, member2.md, member3.md** (je nach Anzahl der Personen)
    - Nennen Sie dort Ihren Namen und beschreiben Sie welche Teile von Ihnen umgesetzt wurden.
  - Der Aufruf **mvn site** von der Kommandozeile muss fehlerfrei ausführbar sein.
- Tipps zur Umsetzung:
  - Erstellen Sie die für Maven Site notwendige **Ordnerstruktur** – siehe Abbildung:

```
+- src/
   +- site/
      +- apt/
      |  +- index.apt
      |
      +- fml/
      |  +- general.fml
      |  +- faq.fml
      |
      +- markdown/
      |  +- markup.md
      |
      +- xdoc/
      |  +- other.xml
      |
      +- xhtml/
      |  +- xhtml-too.xhtml
      |
      +- site.xml
```

  - Da Sie nur Markdown Inhalte hinzufügen werden, ist der **markdown** Unterordner ausreichend.
  - Erstellen Sie eine **site.xml** um die Struktur der Maven Site Dokumentation zu konfigurieren.
    - Eine Ausgangsbasis wird Ihnen ebenso auf Moodle zur Verfügung gestellt.
  - Versuchen Sie **mvn site** auszuführen:

- Vergessen Sie nicht zuvor (bzw. nach jedem **clean**) zuerst **mvn test** auszuführen um die Testergebnisse in die Dokumentation zu integrieren.
- Die Ergebnisse finden Sie unter *target/site/index.html*. Sind alle Inhalte wie erwartet vorhanden?
- Sind die Projekt und Entwickler*innen Informationen in der Dokumentation ersichtlich?
- Sind die member Seiten links im Menü ersichtlich?
- Ist die **Javadoc API**-Hilfe Ihrer Klasse eingebunden?
- Ist die JUnit Testauswertung über jacoco ersichtlich?

ABSCHLUSS DER ÜBUNG

- ☐ Kontrollieren Sie Ihr Repository auf github.com.
  - Alle erwünschen Inhalte vorhanden?
  - Alle unerwünschen Inhalte nicht vorhanden?
- ☐ Speichern Sie Ihren **Repository Link** in der fünften Abgabe. Eine Abgabe pro Gruppe ist ausreichend.

LITERATUR UND HILFREICHE LINKS

- *Divio*
  The documentation system
  https://documentation.divio.com/
  last visited on 17.06.2024
- *Oracle*
  How to Write Doc Comments for the Javadoc Tool
  https://www.oracle.com/technical-resources/articles/java/javadoc-tool.html
  last visited on 17.06.2024
- *Stackoverflow*
  Best practices for writing code comments
  https://stackoverflow.blog/2021/07/05/best-practices-for-writing-code-comments/
  last visited on 17.06.2024

(bzw. Literatur der vorhergehenden Übungen)