Übung 4: Advanced Java, Test-Driven Design (TDD) und LLM-gestütztes Entwickeln

Ziel: In dieser Übung setzen Sie Ihre im vorherigen Übungsblatt entwickelte Implementierungsstrategie um. Sie beginnen, Ihr Domain-Driven-Design-Modell in Code zu überführen und wenden dabei Test-Driven Design (TDD) an. Der Fokus liegt auf der sauberen Implementierung Ihrer Domänenlogik unter Verwendung fortgeschrittener Java-Techniken, der Sicherstellung der Testbarkeit und dem kritisch-reflektierten Einsatz von LLMs.
Relevante Lerneinheiten: AJV + TST(LE8) aus den Bachelorunterlagen
Bearbeitungsdauer: eine Woche.

1. **Review Ihrer Implementierungsstrategie:** Überprüfen Sie die Implementierungsstrategie aus dem letzten Übungsblatt. Nutzen Sie eine LLM, um (a) Ihre Implementierungsstrategie zu analysieren und potenzielle Schwachstellen zu identifizieren; (b) Vorschläge für eine bessere Strukturierung Ihrer Domain-Events zu erhalten; (c) Die Entwicklungsreihenfolge zu optimieren.
   Vorgehen: Präsentieren Sie dem LLM Ihre bisherige Strategie und Bounded Contexts; lassen Sie sich mindestens 3 konkrete Verbesserungsvorschläge geben; bewerten Sie kritisch, welche Vorschläge sinnvoll sind und welche nicht; Passen Sie Ihre Strategie entsprechend an
   Ziel/Output: Die wichtigsten Domain-Events (mind. 2) Ihres Projekts sowie ihre sinnvoll angepasste Entwicklungsstrategie werden nun festgehalten. Welche LLM-Vorschläge haben Sie übernommen/abgelehnt und warum? (ca. 5-6 Sätze)

2. **Testfälle mit LLM generieren und validieren (TDD Schritt 1):** Schreiben Sie Tests für den ersten zentralen Teil Ihrer Domänenlogik, der auf den, im letzten Übungsblatt definierten Bounded Contexts basiert (z.B. das Erstellen oder Verwalten einer Entität). Nutzen Sie dabei ein LLM zur Testgenerierung.
   Vorgehen:
   1. LLM-Prompt erstellen: Beschreiben Sie dem LLM präzise Ihre Domänenlogik und die zu testende Funktionalität (z.B. Validierungsmethoden für Benutzereingaben mit Regular Expressions)
   2. Test-Cases generieren lassen: Lassen Sie das LLM verschiedene Test-Cases vorschlagen:
      - Happy-Path-Tests
      - Edge-Cases (Grenzfälle)
      - Negative Tests (ungültige Eingaben)
   3. Kritische Bewertung: Analysieren Sie die vorgeschlagenen Tests: Sind alle relevanten Fälle abgedeckt? Gibt es unsinnige oder redundante Tests? Welche Tests fehlen?
   4. Regex-Validierung: Lassen Sie sich vom LLM Regular Expressions für Ihre Validierungslogik generieren (z.B. für Namen, E-Mails, Geburtsdaten, Vitaldaten). Prüfen Sie diese auf Korrektheit!
   5. Implementierung: Schreiben Sie die finalen JUnit-Tests

   Ziel/Output: Vollständige JUnit-Testsuite für Ihre Validierungslogik. Dokumentation (ca. 6-7 Sätze): Welche LLM-generierten Tests waren gut/schlecht? Welche haben Sie angepasst oder ergänzt und warum?

3. **Implementierung der Domänenlogik (TDD Schritt 2) mit LLM-Pair-Programming:** Implementieren Sie die Domänenlogik, die Ihre Tests erfüllt. Nutzen Sie das LLM als "Pair-Programming-Partner".
   Vorgehen:
   1. Code-Generierung: Lassen Sie das LLM eine erste Implementierung generieren, die Ihre Tests besteht
   2. Iterative Verbesserung: Bitten Sie das LLM, fortgeschrittene Java-Techniken einzusetzen (Streams, Generics, Lambdas, Optional, Record-Types). Lassen Sie verschiedene Implementierungsvarianten vorschlagen. Fordern Sie Erklärungen für verwendete Patterns
   3. Kritische Qualitätsprüfung: Verstehen Sie den generierten Code vollständig? Entspricht er den DDD-Prinzipien? Ist er wartbar und lesbar? Erfüllt er alle Tests?
   4. Code-Review mit LLM**:** Lassen Sie das LLM Ihren finalen Code reviewen und nach möglichen Bugs oder Verbesserungen suchen

   Ziel/Output: Funktionierende, getestete Implementierung Ihrer Domänenlogik. Dokumentation (ca. 10-12 Sätze): Beschreiben Sie den Pair-Programming-Prozess mit dem LLM. Welche Vorschläge waren hilfreich? Wo mussten Sie korrigieren? Welche Java-Techniken hat das LLM gut/schlecht eingesetzt?

4. **Tests-Erweiterung und Refaktorisierung (TDD Schritt 3):** Erweitern Sie Ihre Tests, um auch Randfälle und Fehlerbedingungen abzudecken.
   Vorgehen:
   1. Test-Erweiterung: Lassen Sie das LLM zusätzliche Edge-Cases und Fehlerbedingungen identifizieren
   2. Refactoring-Vorschläge: Bitten Sie das LLM um Vorschläge für sog. „Code-Smells“ in Ihrer Implementierung, Verbesserung der Code-Lesbarkeit, Performance-Optimierungen sowie bessere Verwendung von Java-Features
   3. Systematisches Refactoring: Implementieren Sie die sinnvollen Vorschläge schrittweise. Stellen Sie sicher, dass alle Tests weiterhin bestehen. Lassen Sie bei Unsicherheiten das LLM die Änderungen erklären.

   Ziel/Output: Erweiterte Testsuite mit Edge-Cases; Refaktorisierter, optimierter Code. Dokumentation (ca. 5-6 Sätze): Welche Refactoring-Vorschläge haben Sie umgesetzt? Welche waren nicht sinnvoll und warum?

5. **Modularität und Testbarkeit via CI/CD sicherstellen:**
   (a) Überprüfen Sie, ob Ihr Code modular aufgebaut ist und sich gut testen lässt. Trennen Sie die verschiedenen Bounded Contexts und sorgen Sie dafür, dass Abhängigkeiten klar definiert sind.
   (b) Setzen Sie sich nun mit dem **zweiten Übungsblatt zum Thema CI/CD** erneut auseinander und dokumentieren Sie Ihr Vorgehen: Wie werden die zuvor implementierten Tests in der CI-Pipeline automatisch ausgeführt?
   Ziel/Output: Eine modulare Architektur, die den Prinzipien von DDD entspricht und eine funktionierende CI-Pipeline. Eine kurze Erklärung (8-10 Sätze), wie Sie Modularität und Testbarkeit in Ihrem Code umgesetzt haben und wie Sie Ihre CI-Pipeline (inkl. Tests) aufgesetzt haben.

6. **Kritische Reflektion zu TDD, DDD und LLM-gestützte Entwicklung:** Reflektieren Sie über Ihre Erfahrungen:
   - Wie hat sich TDD auf Ihre Entwicklung ausgewirkt?
   - Welche Vorteile und Herausforderungen ergaben sich durch DDD?
   - Wie hat der Einsatz von LLMs Ihren Entwicklungsprozess verändert?
   - Was sind Grenzen und Risiken beim LLM-Einsatz in der Softwareentwicklung?
   - Wie stellen Sie sicher, dass Sie den Code verstehen und nicht nur "Copy-Paste" betreiben?
   - Welche Aufgaben eignen sich besonders gut/schlecht für LLM-Unterstützung?

   Ziel/Output: Ein besseres Verständnis der Herangehensweise sowie eine Reflexion zu DDD, TDD und LLM-gestützter Entwicklung. Kritische Bewertung der Chancen und Risiken (ca. 10-15 Sätze).

Hinweise:
- TDD-Prinzip: Tests werden immer VOR der Implementierung geschrieben. (Testframework: JUnit o.ä.)
- DDD-Basis: Die Implementierung baut auf Ihrer DDD-Strategie aus dem vorherigen Übungsblatt auf.
- Backend-Fokus: Konzentration auf Domain, Repositories, Services, Exceptions. Frontend-Entwicklung wird ab der nächsten Übung relevant sein.
- Code-Qualität: Auch LLM-generierter Code muss Clean-Code-Prinzipien folgen
- LLM als Werkzeug: LLMs sind Hilfsmittel, kein Ersatz für Ihr Verständnis. Sie müssen jeden generierten Code verstehen und verantworten können. Hinterfragen Sie LLM-Outputs systematisch. LLMs können falsche oder suboptimale Lösungen generieren. Bei jeder Aufgabe muss dokumentiert werden, wie und wo LLMs eingesetzt wurden.

Abgabe (optional): via Git Repository bis zum **10.11.25, 23:55 Uhr** bereitstellen und via Moodle Kommentar einreichen.