Babeș-Bolyai Universität, Cluj Napoca
Fakultät für Mathematik und Informatik
Objekt-Orientierte Programmierung, 2023

Labor 5

Diskussion: Kalenderwoche 18, **Abgabe:** Kalenderwoche 19, **Wert:** 10/50

Teilnahme am Labor wird nur für diejenigen registriert, die etwas vorzeigen!

Die Laboraufgabe ist in Teams von 2-3 Personen zu bearbeiten.

In den folgenden Laboraufgaben wird die Anwendung zu erweitern sein!

Die Anwendung soll einem Elektroscooterverleih zur Verwaltung des Fuhrparks dienen.
Jedes Elektroauto wird durch die folgenden Charakteristika beschrieben:
Identifier (3 Buchstaben), Modell, Inbetriebnahmedatum, Kilometer, letzter Standort (Adresse als Text), aktueller Zustand (geparkt, reserviert, in Verwendung, in Wartung, außer Betrieb).

Funktionale Anforderungen

- F1. Das Programm soll von 2 unterschiedlichen Benutzern verwendet werden: Manager und Kunde. Zu Beginn des Programms wählt der Benutzer seine Rolle aus.
- F2. Ein Manager kann Elektroscooter hinzufügen, löschen oder bearbeiten können.
- F3. Ein Kunde oder Manager möchte auch nach Elektroscooter via Standort suchen können.
- F4. Ein Kunde oder Manager möchte die Fahrzeugliste nach dem Alter oder Kilometer filtern können, mit einem vom Benutzer angegebenen Wert.
- F5. Ein Manager möchte alle Elektroscooter nach Alter aufsteigend sortiert sehen.
- F6. Ein Kunde soll einen Elektroscooter aus der Liste reservieren oder verwenden können.

Weitere Anforderungen

- W1. Die Interaktion des Benutzers mit dem Programm passiert via **Konsole**. Dem Benutzer werden dort das Menu und Ergebnissen angezeigt, und dort auch die Befehle des Benutzers eingelesen (User Interface, UI).
- W2. Das Programm soll bei Start Beispieldaten von 10 oder mehr Produkten beinhalten. Die Fuhrparkdaten müssen **nicht persistent** gespeichert werden.
- W3. Die Anwendung soll in einer Schicht-Architektur (nach dem Beispieldiagramm in Labor 4) angelegt sein, und entsprechend **modularisiert** werden (mit namespaces für domain, repository, controller, ui);
- W4. Die Referenzen auf repository und controller müssen als **Smart-Pointer** implementiert sein.

Qualitäts-Anforderungen

- Q1. Ausnahmen (z.b. ungültige Eingaben) sollen sorgfältig behandelt werden.
- Q2. Code sollte durchwegs in Englisch geschrieben sein und **selbsterklärende und zweckgemäße Namen** für Klassen, Methoden und Variablen verwenden.
- Q3. Um sicherzustellen dass repository und controller funktionieren, ohne Verwendung der Benutzeroberfläche, sind Unit-Tests erforderlich;

Abgabekriterien

- K1. Die Lösung kann aus mehreren Dateien bestehen. Diese sind als ZIP abzugeben **`L5_TeamX.zip (X ist die Nummer)`** (kein RAR, 7z, o.ä., das Archiv beinhaltet nur Quellcode und keine Verzeichnisse)
- K2. Jedes Teammitglied soll das Ergebnis einreichen und den Code erklären können.
- K3. Die Datei **`Main.cpp`** enthält `main()`
- K4. Das Programm muss mit dem GnuCompiler auf folgende Weise kompilierbar sein:
  ```
  g++ -std=c++20 -o prog *.cpp
  ```

Empfehlungen

- R1. Vereinbart im Team zuallererst die die Schnittstellen der Bereiche domain, ui, controller, repository; das ermöglicht danach parallele Arbeit an der Implementierung verschiedener Teile.