Beleg Programmierung 3 SS 20

Entwicklung einer mehrschichtigen und getesteten Anwendung

Allgemeine Anforderungen

- Java8 (language level)
- Trennung zwischen Test- und Produktiv-Code
- JUnit5 als Testframework
- Mockito als Mock/Spy-framework
- main-Methoden nur im default package
- Abgabe als zip-Datei, welche ein lauffähiges IntelliJ-IDEA-Projekt enthält
- 100% Testabdeckung (additiv) außerhalb des default packages

Testanforderungen

- nicht leer
- nicht tautologisch
- deterministisch
- keine Systemanforderungen
- betriebssystemunabhängig
- immer nur eine Methode unter Test und nur eine Zusicherung, oder Abweichung ist mit Kommentar begründet
- Zusicherung richtig befüllt

Geschäftslogik

Erstellen Sie eine Geschäftslogik zur Verwaltung von Frachtgut mit begrenzter Lagerkapazität. Die Typen der Fracht (`Cargo`) sind bereits als Interfaces definiert. Neben dem Lager für die Fracht ist eine Verwaltung von Kunden (`Customer`) zu realisieren. Kunden sind dabei eindeutig über ihren Namen identifiziert.

Die Anforderung leiten sich aus dem Befehlssatz für das CLI her, wobei die Persistierung nicht zur GL gehört.

Beim Einlagern von Frachtstücken ist zu prüfen: dass die Fracht zu einem bereits existierenden Kunden gehört und die Gesamtlagerkapazität nicht überschritten wird; außerdem ist eine Lagerposition und ein Einlagerungsdatum zu vergeben.

CLI

Implementieren Sie eine Benutzeroberfläche. Die Kommunikation zwischen Oberfläche und Geschäftslogik soll dabei über events erfolgen.

Weiterhin sollen nach dem Beobachterentwurfsmuster 2 Beobachter realisiert werden: der Erste soll eine Meldung produzieren wenn nur noch ein Platz im Lager frei ist, der Zweite über Änderungen an den vorhandenen Gefahrenstoffen informieren. Beachten Sie dass diese erweiterte Funktionalität nicht zur Geschäftslogik gehört.

Das UI soll als zustandsbasiertes (Einfüge-, Anzeige-, Lösch- und Änderungs-Modus, …) command-line interface (CLI) realisiert werden.

Stellen Sie sicher, dass Bedienfehler in der Eingabe keine unkontrollierten Zustände in der Applikation erzeugen.

Beim Starten der Anwendung sollen die Argumente ausgelesen werden. Ist eine Zahl angegeben ist dies die Lagerkapazität. Ist `TCP` oder `UDP` angegeben ist die Applikation als Client für das entsprechende Protokoll zu starten. Dabei kann davon ausgegangen werden, dass der jeweilige Server bereits läuft.

Befehlssatz

- `:c` Wechsel in den Einfügemodus
- `:d` Wechsel in den Löschmodus
- `:r` Wechsel in den Anzeigemodus
- `:u` Wechsel in den Änderungsmodus
- `:p` Wechsel in den Persistenzmodus
- `:config` Wechsel in den Konfigurationsmodus

Einfügemodus:

- `[Kundenname]` fügt einen Kunden ein
- `[Frachttyp] [Kundenname] [Wert] [Einlagerungsdauer in Sekunden] [kommaseparierte Gefahrenstoffe, einzelnes Komma für keine] [[zerbrechlich (y/n)] [unter Druck (y/n)] [fest (y/n)]]` fügt eine Fracht ein; Beispiele:
  - `UnitisedCargo Beispielkunde 2000 86400 , n`

    - `MixedCargoLiquidBulkAndUnitised Beispielkunde 4000.50 86400 radioactive n y`

Anzeigemodus:

- `customer` Anzeige der Kunden mit der Anzahl eingelagerter Frachtstücke
- `cargo [[Frachttyp]]` Anzeige der Frachtstücke ,ggf. gefiltert nach Typ, mit Lagerposition, Einlagerungsdatum und Datum der letzten Inspektion
- `hazard [enthalten(i)/nicht enthalten(e)]` Anzeige der vorhandenen bzw. nicht vorhandenen Gefahrenstoffe

Löschmodus:

- `[Kundenname]` löscht den Kunden
- `[Lagerposition]` löscht das Frachtstück

Änderungsmodus:

- `[Lagerposition]` setzt das Inspektionsdatum auf den aktuellen Zeitpunkt

Persistenzmodus:

- `saveJOS` speichert mittels JOS
- `loadJOS` lädt mittels JOS
- `saveJBP` speichert mittels JBP
- `loadJBP` lädt mittels JBP
- `save [Lagerposition]` speichert eine einzelne Instanz eine Datei für alle Instanzen, falls die Datei nicht existiert werden alle Instanzen gespeichert
- `load [Lagerposition]` lädt eine einzelne Instanz aus der Datei

Konfigurationsmodus:

- `add [Klassenname]` registriert einen benannten Beobachter bzw. listener
- `remove [Klassenname]` de-registriert einen benannten Beobachter bzw. listener

Simulation

Stellen Sie sicher, dass die Geschäftslogik thread-sicher ist.

Simulieren Sie die Verwendung der Geschäftslogik. Erstellen Sie dafür einen Einlagerungsthread der kontinuierlich versucht eine zufällige `Cargo`-Instanz einzufügen.

Erstellen Sie einen Auslagerungsthread der dafür sorgt, dass bei einem fehlgeschlagenen Versuch eine Fracht einzufügen, die Fracht deren Einlagerung am weitesten zurückliegt in ein anderes Lager umgelagert wird. Dieser thread soll nur auf Anforderung tätig werden (waiting). Mit dem Warten soll jetzt auch verhindert werden, das der Einlagerungsthread weitere fruchtlose Versuche unternimmt.

Erstellen Sie eine Simulation mit zwei Einlagerungsthreads, drei Lagern und drei Auslagerungsthreads. Hier sollen die Einlagerungsthreads jedes Mal zufällig auswählen welches der drei Lager versucht wird zu befüllen. Auch die Auslagerungsthreads sollen das Ziellager zufällig wählen. Benutzen Sie auch hier wait/notify um unnötige Rechenzeit zu verhindern.

Stellen Sie auf der Konsole die Aktionen der threads und die Zustandsänderungen an den Lagern (diese über Beobachter) dar. Verwenden Sie nicht `Thread.sleep` o.ä. Eine Terminierung der Simulationen ist nicht gefordert.

GUI

Realisieren Sie eine skalierbare graphische Oberfläche mit JavaFX für die Lagerverwaltung. Sie soll den gleichen Funktionsumfang wie das CLI haben. Die Auflistung der Kunden und Frachtstücke soll immer sichtbar sein und nach Benutzeraktionen automatisch aktualisiert werden.

Die Auflistung der Frachtstücke soll sortierbar nach Lagerposition, Einlagerungsdauer und Kunde sein.

Ermöglichen Sie die Änderung der Lagerposition mittels drag&drop.

I/O

Realisieren Sie die Funktionalität den Zustand der Geschäftslogik zu laden und zu speichern.

Die Persistierung soll wahlweise mit JOS oder JBP erfolgen.

Implementieren Sie weiterhin die Möglichkeit einzelne Frachtstücke wahlfrei (random access auf Dateisystemebene) zu Laden und zu speichern.

net

Realisieren Sie die Verwendung von Oberflächen und Geschäftslogik auf verschiedenen Rechnern. Die Verbindung soll wahlweise über TCP oder UDP erfolgen.

Der Server wird mit 2 Argumenten gestartet: Protokoll und Lagerkapazität.

Ermöglichen Sie die Verwendung mehrerer Clients, die sich auf einen gemeinsamen Server verbinden.

Die Berücksichtigung von Skalierbarkeit, Sicherheit und Transaktionskontrollen ist nicht gefordert.

Zusätzliche Anforderungen

Werden zu Beginn des PZRs spezifiziert.

Dokumentation

- Architekturdiagramm mit allen Schichten und der Einordung aller packages
- ggf. Begründungen
- Quellennachweise (s.u.)

Quellen

Zulässige Quellen sind suchmaschinen-indizierte Internetseiten. Werden mehr als drei zusammenhängende Anweisungen übernommen ist die Quelle in den Kommentaren anzugeben. Ausgeschlossen sind Quellen, die auch als Beleg oder Übungsaufgabe abgegeben werden oder wurden. Zulässig sind außerdem die über moodle bereitgestellten Materialien, diese können für die Übungsaufgaben und Beleg ohne Quellenangabe verwendet werden.

Bewertungsschema

Wird zu Beginn des PZRs spezifiziert.