Beleg Programmierung 3 SS 23 PZR1

Entwicklung einer mehrschichtigen und getesteten Anwendung

Allgemeine Anforderungen

- Trennung zwischen Test- und Produktiv-Code
- JUnit5 als Testframework
- Mockito als Mock/Spy-framework
- keine weiteren Bibliotheken außer JavaFX
- geforderte main-Methoden nur im module `belegProg3`, ansonsten keine weitere Klassen im default package
- Abgabe als zip-Datei, welche ein lauffähiges IntelliJ-IDEA-Projekt im root enthält
- keine Umlaute, Sonderzeichen, etc. in Datei- und Pfadnamen
- mindestens sechs modules, eines zu jeder Basisfunktionalität außer I/O und `belegProg3`

Anforderungen an Testqualität

- eine falsifizierbare Aussage zu einem Methodenaufruf
  - nicht leer (noch zu implementierende Test werden mit `fail` befüllt)
  - enthalten genau eine Zusicherung (ggf. Abweichung kommentieren)
  - Ergebnis ist vom Produktiv-Code abhängig
- ein Ablaufpfad
  - keine Schleifen, Verzweigungen
  - Abzweigungen terminieren mit `fail`
- überall schnell ausführbar
  - betriebssystemunabhängig
  - kein Zugriff auf Dateisysteme, Netzwerk, o.Ä.
  - deterministisch, d.h. keine Zufallsgeneratoren, keine threads
  - schnell, d.h. kein `sleep`, keine Lasttests mit hunderten von Einträgen
- gut lesbar
  - Zusicherung richtig befüllt
  - Duplikate sind zulässig
  - sparsame und angemessene Verwendung von `@BeforeEach`
  - überschaubares setup

- sprechender Testname

Geschäftslogik

Erstellen Sie eine Geschäftslogik zur Verwaltung eines Lagers mit begrenzter Kapazität für Frachtstücke. Die möglichen Inhalte sind bereits als Interfaces im module `contract` im Projekt `belegProg3` definiert. Außerdem sollen auch die Kund*innen bzw. Besitzer*innen verwaltet werden.

Die Geschäftslogik muss folgende Funktionalität realisieren:

- Anlegen von Kund*innen; dabei muss sichergestellt sein, dass kein Name mehr als einmal vorkommt
- Einfügen von Frachtstücken:
  - unterstützt werden alle Typen die sowohl von `Cargo` als auch `Storable` ableiten
  - es ist zu prüfen, dass die Gesamtkapazität nicht überschritten wird
  - es ist zu prüfen, dass das Frachtstück zu einer bereits existierenden Kund*in gehört, ansonsten wird es nicht eingefügt
  - beim Einfügen wird ein Platz vergeben (`storageLocation`); zu keinem Zeitpunkt können mehrere Frachtstücke innerhalb des Lagers den gleichen Platz haben
  - es wird ein Einfügedatum vergeben
- Abruf aller Kund*innen mit der Anzahl der ihrer Frachtstücke
- Abruf eingelagerter Frachtstücke, wird ein Typ (Klassenname) angegeben werden nur Frachtstücke von diesem Typ aufgelistet
- Abruf aller vorhandenen bzw. nicht vorhandenen Gefahrenstoffe (hazards) im Lager
- Setzen des Datums der letzten Überprüfung (`lastInspectionDate`)
- Löschen einer Kund*in
- Entfernen eines Frachtstücks

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 90% der Kapazität

überschritten werden, 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. Jeder Befehl wird mit einem Zeilenwechsel abgeschlossen. Eine Menüführung, insbesondere für Befehle mit mehreren Token, ist nicht gewünscht.

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

Beim Starten der Anwendung sollen die Argumente[1] ausgelesen werden. Ist eine Zahl angegeben ist dies die Kapazität. 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 und an ihm die Kapazität gesetzt wurde.

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

Einfügemodus:

- `[K-Name]` fügt eine Kund*in ein
- `[Fracht-Typ] [K-Name] [Wert] [kommaseparierte Gefahrenstoffe, einzelnes Komma für keine] [[fragile(true/false)]] [[pressurized(true/false)]] [[grainSize]]` fügt ein Frachtstück ein, die Reihenfolge von pressurized, Korngröße und fragile entspricht der Reihenfolge im Typnamen; Beispiele:
  - `LiquidAndDryBulkCargo Alice 4004,50 flammable,toxic true 10`
  - `UnitisedCargo Bob 10000 , false`

[1] https://www.jetbrains.com/help/idea/2023.1/run-debug-configuration.html

Anzeigemodus:

- `customers` Anzeige der Kund*innen mit der Anzahl der Frachtstücke
- `cargos [[Typ]]` Anzeige der Frachtstücke - ggf. gefiltert nach Typ - mit Platz, Inspektionsdatum und Einlagerungsdauer. Diese Darstellung muss nur so aktuell wie der letzte Abruf sein.
- `hazards [enthalten(i)/nicht enthalten(e)]` Anzeige der vorhandenen bzw. nicht vorhandenen Gefahrenstoffe

Löschmodus:

- `[K-Name]` löscht die Kund*in
- `[Platz]` entfernt das Frachtstück

Änderungsmodus:

- `[Platz]` setzt das Datum der Inspektion

Persistenzmodus:

- `saveJOS` speichert mittels JOS
- `loadJOS` lädt mittels JOS
- `saveJBP` speichert mittels JBP
- `loadJBP` lädt mittels JBP

alternatives CLI

Erstellen sie ein alternatives CLI mit eigener main-Methode in dem 2 Funktionalitäten (vorzugsweise Löschen von Kund*innen und Auflisten der Gefahrenstoffe) deaktiviert sind und nur ein Beobachter aktiv ist. Dieses CLI soll sich vom eigentlichen CLI nur durch die Konfiguration unterscheiden, nicht durch die Implementierung. D.h. der Unterschied besteht nur im setup in der main-Methode beim Einhängen der listener bzw. handler. Welche Funktionalitäten deaktiviert sind muss dokumentiert werden. Dieses CLI muss nicht über das Netzwerk funktionieren.

Simulation

Stellen Sie sicher, dass die Geschäftslogik thread-sicher ist. Erstellen Sie dafür Simulationen, die die Verwendung der Geschäftslogik im Produktivbetrieb testen. Die Abläufe in der Simulation sollen auf der Konsole dokumentiert werden. Jeder thread muss für jede ändernde Interaktion an der Geschäftslogik eine Ausgabe

produzieren. Jede Änderung an der Geschäftslogik muss eine Ausgabe produzieren, sinnvollerweise über Beobachter.

Zur Entwicklung darf `Thread.sleep` o.Ä. verwendet werden, in der Abgabe muss dies deaktiviert sein bzw. darf nur mit Null-Werten verwendet werden.

Den Simulationen ist die Kapazität per Kommandozeilenargument zu übergeben, wobei 0 ein zulässiger Wert ist.

Simulation 1

Erstellen Sie einen thread der kontinuierlich versucht einen zufällig erzeugtes Frachtstück einzufügen. Erstellen Sie einen weiteren thread der kontinuierlich die Liste der enthaltenen Frachtstücke abruft, daraus zufällig eines auswählt und löscht. Diese Simulation sollte nicht terminieren und nicht wait/notify (bzw. await/signal) verwenden.

Simulation 2

Erweitern Sie die Simulation 1 so, daß beim Start neben der Kapazität auch eine Anzahl *n* beim Start übergeben wird. In der Simulation sollen dann *n* einfügende und *n* löschende threads gestartet werden. Diese threads schreiben auf der Konsole welche Änderung sie an der Geschäftslogik vornehmen wollen. Die Änderungen an der Geschäftslogik werden durch einen Beobachter auf der Konsole ausgegeben.

Simulation 3

Erstellen Sie einen weiteren thread der kontinuierlich die Liste der enthaltenen Frachtstücke abruft, daraus zufällig einen auswählt und die Inspektion auslöst. Modifizieren Sie den löschenden thread so, dass er das Frachtstück mit dem ältesten Inspektionsdatum löscht. Sorgen Sie mit wait/notify (bzw. await/signal) dafür, das Einfügen und Löschen miteinander synchronisiert sind. D.h. wenn das Einfügen wegen Kapazitätsmangel nicht möglich ist werden die löschenden threads benachrichtigt und während diese arbeiten warten die einfügenden threads. Umgekehrt arbeiten auch die löschenden threads nicht während der Ausführung des Einfügens.

Beim Start der Simulation 3 werden drei Parameter übergeben, die Kapazität, die Anzahl der zu startenden threads (Einfügen, Löschen, Update) und ein Intervall in Millisekunden. Die Änderungen an der Geschäftslogik werden jetzt nicht mehr durch einen Beobachter ausgegeben, sondern dafür soll ein thread implementiert

werden, der in regelmäßigen Abständen (übergebenes Intervall) den Zustand der Geschäftslogik liest und ausgibt. Von diesem thread soll immer nur eine Instanz gestartet werden. Verwenden Sie dafür den `ExecutorService` wie in https://www.baeldung.com/java-timer-and-timertask beschrieben.

GUI

Realisieren Sie eine skalierbare graphische Oberfläche mit JavaFX für die Verwaltung. Sie soll den gleichen Funktionsumfang wie das CLI haben, abzüglich der Beobachter. Die Auflistung der Kund*innen und Frachtstücke soll immer sichtbar sein und nach Benutzeraktionen automatisch aktualisiert werden.

Die Auflistung der Frachtstücke soll sortierbar nach Platz, Kund*in, Inspektionsdatum und Einlagerungsdauer sein.

Ermöglichen Sie die Tausch von Lagerplätzen mittels drag&drop.

Das Einfügen der Frachtstücke sollte nicht sperrend sein, es sollte also möglich sein, während der Einfügens andere Funktionen oder ein weiteres Einfügen auszulösen. D.h. die Ausführung des Einfügens sollte nebenläufig erfolgen.

I/O

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

Die Anwender*innen können wählen ob die Persistierung mit JOS oder JBP erfolgt.

net

Realisieren Sie die Verwendung von CLI und Geschäftslogik in verschiedenen Prozessen. Die Verbindung soll in Abhängigkeit vom übergebenen Kommandozeilenargument über TCP oder UDP erfolgen. Der Server wird mit 2 Argumenten gestartet: Protokoll und Kapazität. Ermöglichen Sie die Verwendung mehrerer Clients, die sich auf einen gemeinsamen Server verbinden für TCP oder UDP.
Die Berücksichtigung von Skalierbarkeit, Sicherheit und Transaktionskontrollen ist nicht gefordert. Auch die Beobachter müssen im Netzwerkmodus nicht unterstützt werden.

zusätzliche Anforderungen

Realisieren sie ein optionales, mehrsprachiges Logsystem.

Wird beim Starten der CLI-Applikation „EN“ oder „DE“ als Kommandozeilenparameter übergeben wird das Log verwendet. Ist kein Kürzel angegeben wird es nicht verwendet. Im Netzwerkmodus muss das Log nicht unterstützt werden.

In dem Log wird jede Benutzerinteraktion, die die Geschäftslogik erreicht, in einer Zeile dokumentiert. Außerdem werden alle Änderungen am Zustand der Geschäftslogik ebenfalls in einer Zeile dokumentiert. Die Logeinträge zu Änderungen an der Geschäftslogik können unspezifisch gehalten werden.

Der Dateiname für das Log ist „log.txt“. Beachten Sie, dass das Log eine singuläre Ressource ist und geschützt werden muss. Auch gehört das Log nicht zur Geschäftslogik und die bestehende Implementierung sollte unabhängig von der Log-Implementierung sein.

Implementieren Sie das Log vollständig für DE und prototypisch mit vier übersetzten Log-Einträgen für EN. Berücksichtigen Sie, dass theoretisch beliebige Sprachen unterstützt werden sollen. Encoding und Schreibrichtung müssen aber nicht berücksichtigt werden.

Dokumentation

- befüllte Checklist.md im root des Projektes
- Architekturdiagramm mit allen Schichten (modules) und deren Einordung in die Schichtenarchitektur, mit kurzer Beschreibung der Schichten
- 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.

Flüchtige Quellen, wie Sprachmodelle, sind per screen shot zu dokumentieren.

Bewertungsschema

siehe Checklist.draft.md

Testabdeckung Geschäftslogik

100% Testabdeckung (additiv)

Testabdeckung Rest

100% Testabdeckung (additiv) für:

- Einfügen von Kund*innen über das CLI
- Anzeigen von Kund*innen über das CLI
- ein Beobachter bzw. dessen alternative Implementierung
- deterministische Funktionalität der Simulationen
- Speichern via JOS oder JBP
- Laden via JOS oder JBP

Kapselung

prüft anhand von Testfällen die Kapselung der Implementierung, z.B.:

1. erzeuge Lager mit zwei Frachtstücken
2. hole Liste der Frachtstücke
3. lösche diese Liste (`clear()`)
4. hole Liste der Frachtstücke erneut
5. prüfe Inhalt dieser Liste (sollte nicht leer sein)

keine Ablauffehler

prüft anhand von use cases die Korrektheit der Implementierung, z.B.:

```
:c
Alice
LiquidBulkCargo Alice 750 toxic false
DryBulkCargo Alice 1002,50 , 10
:r
cargos DryBulkCargo
hazards i
customers
```

Architekturdiagramm

Erstellen Sie eine Graphik in der alle modules gemäß der Schichtenarchitektur angeordnet sind. Die kurze Beschreibung der Schichten kann auch in einer weiteren Textdatei erfolgen.

geforderte main-Methoden

- CLI
- alternatives CLI
- je eine für jede Simulation
- GUI
- Server