

Beleg Programmierung 3 SS 24 PZR1

Entwicklung einer mehrschichtigen und getesteten Anwendung

Allgemeine Anforderungen

- Trennung zwischen Test- und Produktiv-Code
- keine weiteren Bibliotheken außer JUnit5, Mockito und JavaFX (und deren Abhängigkeiten)
- 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

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`
 - sprechender Testname
 - richtige Testklassenzuordnung inkl. package

Geschäftslogik

Erstellen Sie eine Geschäftslogik zur Verwaltung von Mediadateien mit begrenzter Kapazität. Die Typen der Mediadateien (`MediaContent`) sind bereits als Interfaces definiert im module `contract` im Projekt `belegProg3` definiert. Neben der Verwaltung für die Mediadateien ist eine Verwaltung von Produzent*innen (`Uploader`) zu realisieren. Die Geschäftslogik ist nur für die Verwaltung zuständig, nicht das tatsächliche Hochladen oder Speichern von Dateien.

Die Geschäftslogik muss folgende Funktionalität realisieren:

- Anlegen von Produzenten; dabei muss sichergestellt sein, dass kein Name mehr als einmal vorkommt
- Hochladen von Mediadateien:
 - unterstützt werden alle Typen die sowohl von `Uploadable` als auch `MediaContent` ableiten
 - es ist zu prüfen, dass die Media-Datei zu einer bereits existierenden Produzent*in gehört
 - es ist zu prüfen, dass die Gesamtkapazität nicht überschritten wird, dafür ist die Dateigröße in `size` definiert
 - beim Hochladen wird eine Abrufadresse vergeben (`address`); zu keinem Zeitpunkt können mehrere Mediadateien innerhalb der Verwaltung die gleiche Abrufadresse haben
 - beim Hochladen wird ein Upload-Datum vergeben
- Abruf aller Produzent*innen mit der Anzahl der ihrer Mediadateien
- Abruf vorhandener Mediadateien; wird ein Typ angegeben werden nur Mediadateien von diesem Typ aufgelistet
- Abruf aller vergebenen bzw. nicht vergebenen Tags in der Verwaltung
- Hochsetzen der Abrufe (`accessCount`)
- Löschen einer bestimmten Produzent*in
- Löschen einer bestimmten Mediadatei

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 Tags 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 Argumente¹ 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:

- `[P-Name]` fügt eine Produzent*in ein
- `[Media-Typ] [P-Name] [kommaseparierte Tags, einzelnes Komma für keine] [Größe] [Abrufkosten] [[Optionale Parameter]]` Fügt eine Mediadatei ein, die optionalen Parameter sind abhängig vom Typ, ihre Reihenfolge wird durch den Typ-Namen bestimmt. Bei keiner Angabe sind default-Werte zu vergeben.
Beispiele:

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

- o `AudioVideo RedLetterMedia Review 8700 3,60 44100 1080`
- o `Audio EdBangerRecords , 1200 2,35 44100`

Anzeigemodus:

- `uploader` Anzeige der Produzent*innen mit der Anzahl der Mediadateien
- `content [[Typ]]` Anzeige der Mediadateien - ggf. gefiltert nach Typ - mit Abrufadresse, Verfügbarkeit und Anzahl der Abrufe. Die Verfügbarkeit ergibt sich aus der Differenz zwischen dem aktuellen Datum und dem Upload-Datum. Diese Darstellung muss nur so aktuell wie der letzte Abruf sein.
- `tag [enthalten(i)/nicht enthalten(e)]` Anzeige der vorhandenen bzw. nicht vorhandenen Tags

Löschmodus:

- `[P-Name]` löscht die Produzent*in
- `[Abrufadresse]` löscht die Mediadatei

Änderungsmodus:

- `[Abrufadresse]` erhöht den Zähler für die Abrufe um eins

Persistenzmodus:

- `save [JOS|JBP]` speichert den Zustand der Geschäftslogik mittels der angegebenen Technologie
- `load [JOS|JBP]` lädt den Zustand der Geschäftslogik mittels der angegebenen Technologie

alternatives CLI

Erstellen sie ein alternatives CLI mit eigener main-Methode in dem 2 Funktionalitäten (vorzugsweise Löschen von Produzent*innen und Auflisten der Tags) 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, handler bzw. Beobachter. 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 eine zufällig erzeugte Mediadatei einzufügen. Erstellen Sie einen weiteren thread der kontinuierlich die enthaltenen Mediadateien abrufen, daraus zufällig eine 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 Mediadateien abrufen, daraus zufällig eine auswählt und den Zugriff auslöst. Modifizieren Sie den löschenden thread so, dass er die Mediadatei mit dem niedrigsten Zugriffs-Zähler 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 `ScheduledExecutorService` 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 und der Netzwerkfunktionalität. Die Auflistung der Produzent*innen und Mediadateien soll immer sichtbar sein und nach Benutzeraktionen automatisch aktualisiert werden.

Die Auflistung der Mediadateien soll sortierbar nach Abrufadresse, Produzent *in, Verfügbarkeit und Anzahl der Abrufe sein.

I/O

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

Die Beobachter gehören hierbei nicht zum Zustand der Geschäftslogik.

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 Logsystem.

Wird beim Starten der CLI-Applikation „log“ als Kommandozeilenparameter übergeben wird das Log verwendet. Ist der Parameter nicht angegeben wird es nicht verwendet. Im Netzwerkmodus muss das Log nicht unterstützt werden. Auch nach dem Laden der GL muss das Log nicht mehr funktionieren.

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.

Dokumentation

- befüllte Checklist.md im root des Projektes
- Architekturdiagramm mit allen Schichten (modules) und deren Einordnung in die Schichtenarchitektur
- 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 Produzent*innen über das CLI
- Anzeigen von Produzent*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

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

1. erzeuge Verwaltung mit zwei Mediadateien
2. hole Aufzählung der Mediadateien
3. lösche diese Aufzählung (`clear()`)
4. hole Aufzählung der Mediadateien erneut
5. prüfe Inhalt dieser Aufzählung (sollte nicht leer sein)

keine Ablauffehler

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

```
:c
Alice
Audio Alice Music 1200 2,05 44100
AudioVideo Alice Review,Music 7600 2,25
:r
content Audio
tag i
uploader
```

Architekturdiagramm

Erstellen Sie eine Graphik in der alle modules gemäß der Schichtenarchitektur angeordnet sind.

geforderte main-Methoden

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