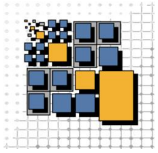

Otto-Friedrich-Universität Bamberg - Fakultät WIAI
Lehrstuhl für Praktische Informatik

Advanced Java Programming SoSe 2022
Assignment 3 - Testen mit JUnit (und Mockito)

Hinweise zum Assignment

- Die Abgabe erfolgt am 12.07.2022 **um 22:00 Uhr** Serverzeit.
- Es sind insgesamt 20 Punkte zu erreichen.
- Nutzen Sie zur Bearbeitung des Assignments die im VC verfügbare Projektstruktur.
- Nutzen Sie bei Fragen das Forum im VC oder das Tutorium.

Aufgabe 1

Aufgrund Ihrer Leistungen in den vorherigen Aufträgen ist nun ein kleinerer fränkischer Filmverleih namens NedFligs auf Sie aufmerksam geworden. NedFligs möchte in Zukunft stärker auf das Internet setzen und seine Software auch für das Web einsetzen. Dabei sind auch verschiedene Softwareerweiterungen geplant. Ein kleiner Teil der Software, das Preis- sowie Rechnungssystem, wurde von Martin Fowler geschrieben und von Werkstudenten gewartet. Martin Fowler hat seiner Software sogar ein ganzes Buchkapitel [Fowler, 1999, Kapitel 1] gewidmet. Dieses erläutert die grundlegende Struktur der Software, also sollten Sie es als Erstes lesen.

Ihre Aufgabe ist es, das vorhandene System zur Preis und Rechnungserstellung zu dokumentieren, vollständig zu testen und anschließend um weitere Funktionalität zu erweitern. Die Erweiterungen sollen natürlich auch getestet werden.

Fehlerfälle
Erweitern Sie alle nicht privaten Methoden und Konstruktoren um die Validierung von Eingabeparametern. Achten Sie dabei insbesondere auf die Behandlung von Grenzfällen wie `null`, leeren Strings, leeren Listen, negativen Werten und der Zahl 0. Diese können mit Exceptions behandelt werden, wobei dies nicht immer sinnvoll oder notwendig ist.

Dokumentation
Dokumentieren Sie die Klassen `Customer` und `Movie` mit Hilfe von JavaDoc. Schreiben Sie eine Dokumentation für die Klasse selbst und für alle Methoden, wobei triviale Getter und Setter nicht beachtet werden müssen. Achten Sie dabei auf entsprechende Vorbedingungen, Nachbedingungen, Beschreibung der Parameter, Rückgabewerte, mögliche Fehlerfälle und deren Ursache.

Testen
Das System muss sowohl Unit- als auch Integrationstests besitzen, die automatisch ausführbar sind und keine Interaktion eines Testers erfordern. Nutzen sie zum Testen JUnit.

Unit Tests
Schreiben Sie für jede nicht-private Methode mindestens einen Testcase, achten Sie dabei auf eine hohe Test coverage. Da die Methoden isoliert voneinander getestet werden sollen, sind (nach Möglichkeit) Mocks und/oder Stubs zu verwenden. Für die Implementierung von entsprechenden Mock/Stub Objekten können Sie entweder Mockito oder eigene, zu diesem Zweck erstellte, Java-Klassen verwenden. Für das Testen von konkreten Methoden in abstrakten Klassen empfiehlt sich die Erstellung eigener Klassen, die von der abstrakten Klasse erben. Ansonsten ist der Aufwand mit Mockito geringer.

Hinweise zu den Unit Tests:

- Zum Testen können sowohl Stubs, Mocks als auch Kombinationen von beiden Arten verwendet werden. Die Wahl steht Ihnen frei - treffen Sie eine sinnvolle Entscheidung!
- Listen müssen **NICHT** mittels eines Mocks/Stubs simuliert werden. Hier ist eine entsprechende Java Implementierung (`LinkedList`, `ArrayList`, etc.) zu verwenden.
- Alle Unit-Tests sollen im Verzeichnis `src/test/java` parallel zu `src/main/java` liegen, welches sich direkt im Projektverzeichnis befindet.
- Zum Testen von größeren `String`-Ausgaben bietet sich die Verwendung von Dateien an, die innerhalb des Verzeichnisses für Tests liegen. Der Inhalt der Dateien ist einzulesen und mit der Ausgabe des Programms zu vergleichen.

Integration Tests
Des Weiteren sollen Integrationstests für das Erstellen von Rechnungen in Text oder HTML geschrieben werden. Hier dürfen keine Mocks/Stubs verwendet werden, da die Integration aller bisher in Isolation getesteter Klassen und Methoden überprüft wird.

Hinweise zu den Integration Tests:

- Alle Integrationstests sollen im Verzeichnis `src/integrationtest/java` parallel zu `src/main/java` liegen, das direkt im Projektverzeichnis liegt.
- Zum Testen von größeren `String`-Ausgaben bietet sich die Verwendung von Dateien an, die innerhalb des Verzeichnisses für Tests liegen. Der Inhalt der Dateien ist einzulesen und mit der Ausgabe des Programms zu vergleichen.

Erweiterungen
Folgende Änderungen am System sind gefordert:

- Die Software stammt aus einer Zeit, in der noch keine Enumerations in Java möglich waren. Modernisieren und refactoren Sie den bestehenden Code bezüglich der "PriceCodes", indem Sie die in der Klasse `Movie` definierten Konstanten für die Preiskategorien in ein `enum` umwandeln.
- Eine neue Preiskategorie für *Staffeln* von Fernsehserien soll eingeführt werden. Eine Staffel einer TV Serie kostet 3.5 am ersten Tag, 3 am zweiten und pro weiteren Tag 1.5. Mit TV Serien kann ein Kunde einmalig 3 Frequent Renter Points sammeln.
- Die Filme und Serien sollen in unterschiedlichen Bildqualitäten ausleihbar sein, nämlich in HD oder 4k. Die Bildqualität soll in der Klasse `Movie` verwaltet werden. Ein Film in 4k soll 2 mehr kosten als ein Film in HD unabhängig von der Dauer der Ausleihe. Zusätzlich soll die Bildqualität auf der Rechnung hinter dem Filmtitel ausgewiesen werden.
- Einem Kunden soll ein Gutschein zugewiesen werden können. Ein Gutschein besteht aus einem Gutscheincode und einem Wert. Der Wert des Gutscheins wird gegen den Rechnungsbetrag gegengerechnet. Dadurch kann sich der zu zahlende Rechnungsbetrag auch auf 0 reduzieren, wenn die Gutscheinsumme größer oder gleich dem eigentlichen Rechnungsbetrag ist. Gleichzeitig bekommt der Kunde, der einen Gutschein einsetzt, zusätzlich drei *Frequent Renter Points*. Der Gutschein und dessen Auswirkungen müssen in der Rechnung ersichtlich sein (HTML und Text).

Getter/Setter und Konstruktoren
Triviale Konstruktoren, Getter und Setter müssen nicht getestet oder dokumentiert werden. Ausnahmen ergeben sich, wenn diese über ihre normalerweise triviale Logik hinausgehen (z.B. wenn übergebene Parameter geprüft werden oder Exceptions geworfen werden).

Generelle Anmerkungen zur Bearbeitung des Assignments

- As the system is developed and maintained by an international team the whole source code and the entire documentation should be written in English. Do not mix English and German.
- Versuchen Sie bei den geforderten Erweiterungen die Methodik des Test-Driven Developments einzusetzen.
- Abzugebendes Artefakt ist ein einzelnes Gradle-Projekt mit dem Namen `Assignment3`. Alle von Gradle benötigten Dateien sind bereits in der Vorlage enthalten und sollen auch beibehalten werden. Die bestehenden Inhalte der `build.gradle`-Datei dürfen nicht verändert werden, einzig Hinzufügungen sind erlaubt, aber für die Bearbeitung nicht notwendig.
- Nutzen Sie das für Ihre Gruppe erstellte Git-Repository bei der Bearbeitung der Aufgabe. Laden Sie dazu Ihr Projektverzeichnis (namens `Assignment3`) direkt in den root-Ordner Ihres Gruppenrepositories. Die Abgabe des Assignments erfolgt ausschließlich über das Gruppenrepository, d.h. der Stand, der am Ende des Bearbeitungszeitraums im Repository committed ist, wird bewertet.
- Verwenden Sie bitte das vorgebene Package `de.uniba.wiai.dsg.ajp.assignment3` für Ihr Projekt. Bilden Sie Unterpakete und gliedern Sie Ihr Programm sinnvoll.
- Achten Sie auf grundlegende Programmierregeln! Halten Sie sich an die Java Code Conventions[1]. Achten Sie auf ordnungsgemäße Benennung und Formatierung.
- `e.printStackTrace` oder `throw new RuntimeException(e)` sind keine adequaten Fehlerbehandlungen.

Literatur

[Fowler, 1999] Fowler, M. (1999). *Refactoring: Improving the Design of Existing Code*. Addison-Wesley Longman, Inc., Boston, MA, USA. https://katalog.ub.uni-bamberg.de/query/BV035479478.

[1] Siehe http://www.oracle.com/technetwork/articles/javase/codeconvtoc-136057.html