Praktikum Mock-Testing

Einleitung

Ziele dieses Praktikums sind:

- Anwenden von erweiterten Testverfahren (Mocks, Spys)
- Übung des Umgangs mit Mockito

Voraussetzungen

- Vorlesung Mock-Testing

Tooling

- Installiertes JDK 17+
- Gradle 7.6+
- Mockito ab Version 2.

Struktur

Ein Praktikum kann verschiedene Arten von Aufgaben enthalten, die wie folgt gekennzeichnet sind:

[TU] – Theoretische Übung

Dient der Repetition bzw. Vertiefung des Stoffes aus der Vorlesung und als Vorbereitung für die nachfolgenden Übungen.

[PU] – Praktische Übung

Übungsaufgaben zur praktischen Vertiefung von Teilaspekten des behandelten Themas.

[PA] – Pflichtaufgabe

Übergreifende Aufgabe zum Abschluss. Das Lösen dieser Aufgaben ist Pflicht. Sie muss bis zum definierten Zeitpunkt abgegeben werden, wird bewertet und ist Teil der Vornote.

Zeit und Bewertung

Für das Praktikum stehen die Wochen gemäss den Angaben in Moodle zur Verfügung.
Je nach Kenntnis- und Erfahrungsstufe benötigen Sie mehr oder weniger Zeit. Nutzen Sie die Gelegenheit den Stoff zu vertiefen, auszuprobieren, Fragen zu stellen und Lösungen zu diskutieren (Intensive-Track).
Falls Sie das Thema schon beherrschen, müssen Sie nur die Pflichtaufgaben lösen und bis zum angegebenen Zeitpunkt abgeben (Fast-Track).

Die Pflichtaufgaben werden mit 0 bis 2 Punkten bewertet (siehe *Leistungsnachweise* auf Moodle).

Sie können die Lösungen der Textaufgaben in diesem Praktikum direkt in den Java-Files als Kommentar hinzufügen - oder als Markdown-File neben den Java-Files.

Referenzen

- Praktikumsverzeichnis – Quellcode, Projektstruktur
- Mockito Homepage
- Mockito Dokumentation
- Konkrete Beispiele mit Mockito

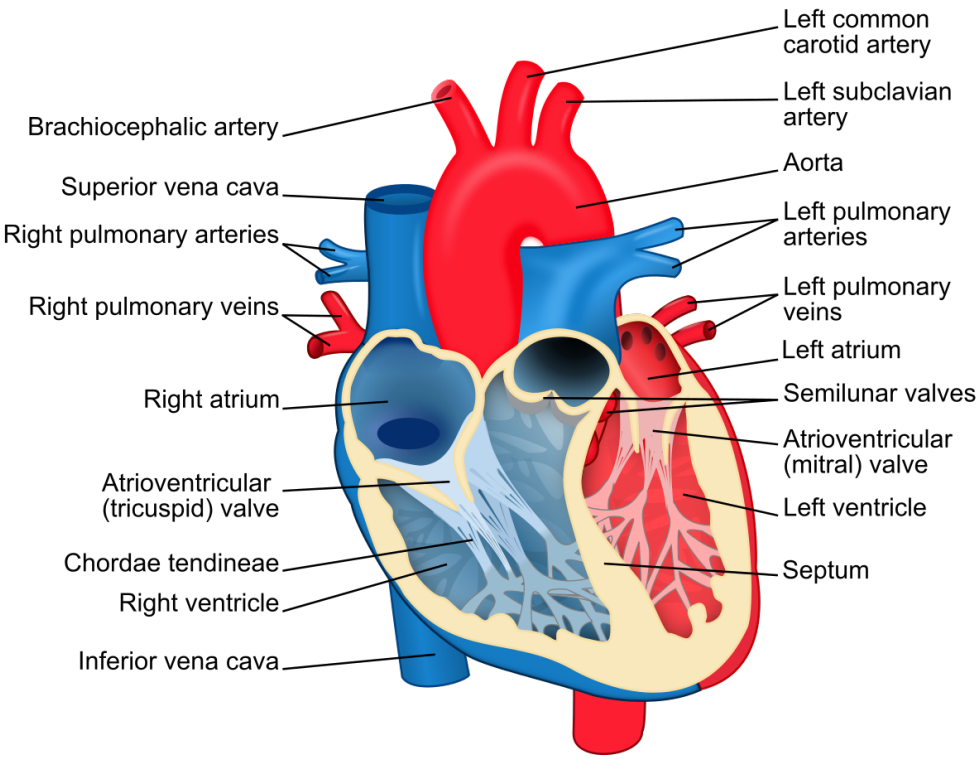
1. Herz

Das menschliche Herz ist ein lebenswichtiges und kompliziertes Organ. Nur, wenn seine einzelnen "Bauteile" richtig koordiniert werden, kann es eine ausreichende Blutversorgung unseres Körpers gewährleisten.

Da es beim Mocking um das Testen des Verhaltens eines Objekts geht, bietet sich das Herz als Beispiel an. Darum simulieren wir in dieser Aufgabe ein menschliches Herz. Eine kurze Erklärung der Funktionsweise des Herzens gibt es auf Planet Wissen .

Einige Fakten über das Herz, die bei dieser Aufgabe helfen können - im Projekt werden englische Begriffe verwendet, die Sie in Klammern finden:

- Das Herz (Heart) besteht aus zwei Hälften.
- Jede Herzhälfte (Half) hat einen Vorhof (Atrium) und eine Herzkammer (Ventricle). Vorhöfe sammeln Blut, Herzkammern pressen es in den Körper.
- Zwei Phasen wechseln sich jeweils ab.
  - Anspannungsphase: Systole
  - Entspannungsphase: Diastole
- Die Herzklappen sind eine Art Rücklaufventile, es gibt 2 verschiedene:
  - 2 Segelklappen (AtrioventricularValve): verhindern, dass Blut in der Systole in den Vorhof zurückfliesst.
  - 2 Taschenklappen (SemilunarValve): verhindern, dass in der Diastole Blut ins Herz zurückfliesst.



1.1. Ablauf des Herzschlags

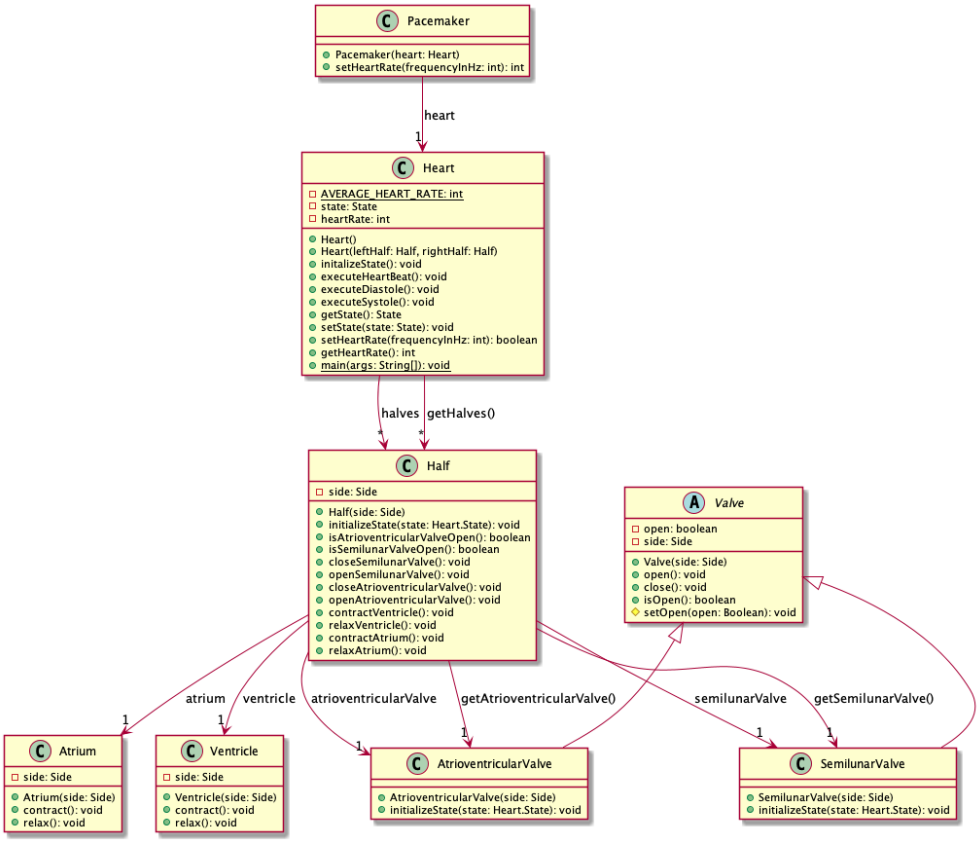
Diastole – Blut sammelt sich im Vorhof ("atrium")

- Segelklappen ("atrioventricular valves") öffnen
- Taschenklappen ("semilunar valves") schliessen
- Vorhöfe ("atria") entspannen
- Herzkammern ("ventricle") entspannen
- Warten auf Systole

Systole

- Segelklappen schliessen
- Taschenklappen öffnen
- Herzkammern kontrahieren
- Vorhöfe kontrahieren
- Warten auf Diastole

Es gibt die folgenden Klassen im Projekt:



Darüber hinaus gibt es eine Klasse `HeartTest`, die bereits mehrere Tests implementiert.

Stubs sind nicht sehr intelligent und daher kann man häufig nur oberflächlich mit Stubs testen. Sie sollen mit Hilfe von Mock-Objekten die Tests noch gründlicher gestalten.

Die Abhängigkeit für Mockito ist bereits für Sie in build.gradle erfasst worden. Studieren Sie kurz in diesem File, wie das gemacht wurde. Informationen dazu finden Sie auf der Mockito Homepage und auf MavenCentral.

Studieren bzw. überfliegen Sie bei dieser Gelegenheit auch gerade die Mockito Dokumentation.

2. Aufgaben

2.1. Einführung in Mockito [PU]

a. Studieren Sie die Testmethoden `HeartTest.testValveStatus()` und `HeartTest.testExecuteHeartBeatErrorBehavior()`. Dort wird **ein Teil** des Verhaltens des Herzens getestet.

b. Implementieren Sie die Methoden `Heart.executeDiastole()`, `Heart.executeSystole()` und `Heart.executeHeartBeat()` sodass die bestehenden Tests durchlaufen und das oben beschriebene Verhalten des Herzens modelliert wird.

c. Optional:
Ein echtes Herz hat eine Schlagfrequenz. Implementieren Sie, dass das Herz nach jeder Systole pausiert. Sie können z. B. den aktuellen Thread 1000 ms lang anhalten mit

```
try {
    Thread.currentThread().sleep(1000);
} catch (InterruptedException e) {
    System.out.println("Thread interrupted: " + e.getMessage());
}
```

2.2. Fragen zu Testing [TU]

a. Testing kann in zwei unterschiedliche Strategien aufgeteilt werden. Zum einen gibt es White-Box-Testing und zum zweiten Black-Box-Testing. Was für Java Libraries gibt es, um diese zwei Strategien zu testen? Wann wenden Sie welche Strategie an?

b. Das Erstellen von guten automatisierten Unit-Tests kann manchmal schwierig umzusetzen sein. Was ist der Hauptgrund dafür? Wie können Sie dieses Problem entschärfen?

c. Der Testfokus war bisher auf der Klasse `Heart`. Testen Sie jetzt die Klasse `Half`. Wo verwenden Sie Stubbing, wo Mocking?

2.3. Vorbedingungen, Exceptions und Callback-Funktionen [PA]

a. Testen Sie, dass beim Ausführen der Diastole und Systole alle Herzklappen in der richtigen Reihenfolge geöffnet und geschlossen werden. Implementieren Sie dafür die Methode `HeartTest.testValvesBehavior()` und benutzen Sie die von Mockito bereitgestellte Klasse `InOrder`. Benutzen Sie mindestens einmal die statische Methode `mock()` und einmal die Annotation `@Mock` von Mockito.
Testen Sie auch, dass keine Methode unnötig oft aufgerufen wird (mit der Methode `Inorder.verifyNoMoreInteractions()` bzw. `Mockito.verifyNoMoreInteractions(mock)`).

b. Die vorhandene Implementation des Testfalls `HeartTest.testExecuteHeartBeatErrorBehavior()` verwendet keine Mock-Objekte für die linke und rechte Herzhälfte. Implementieren Sie die Testmethode `HeartTest.testExecuteHeartBeatErrorBehaviorWithMocking()` analog zur bestehenden Variante, aber diesmal setzen Sie entsprechende Mock-Objekte für die Herzhälften ein.

Für das Stubbing werden Sie die `doThrow`- Methode von Mockito benötigen. Mit Stubbing ist die Konfiguration von Ihrem Mock-Objekt gemeint.

c. Die bestehende Implementierung hat einen ungenügenden Exception-Mechanismus. Beispielsweise führt die Herzhälfte (`Half`) ihre Methoden aus, auch wenn die notwendigen Vorbedingungen (d.h., die Klappenstellungen von `SemilunarValve` und `AtrioVentricularValve`) nicht gelten.
Beheben Sie dieses Problem, indem Sie in `Heart.executeDiastole()` und `Heart.executeSystole()` eine `InvalidValvePositionException` werfen, falls die notwendigen Vorbedingungen nicht in der korrekten Position (offen, geschlossen) sind.
Die Excepton `InvalidValvePositionException` müssen sie selber erstellen.
Erstellen Sie zwei Methoden `testDiastoleException()` und `testSystoleException()` und setzen Sie Mocking und Stubbing ein, um diesen Exception-Mechanismus zu testen.

d. Die Klasse Pacemaker ist eine Implementation für einen Herzschrittmacher. Es ist eine unvollständige Implementation, reduziert auf das Wesentliche für den Inhalt des Praktikums. Ihre Aufgabe ist es, die Methode `Pacemaker.setHeartRate` mit zwei vorgegeben Testfällen zu testen. Implementieren Sie Test-Methoden `PacemakerTest.testSetHeartRateRejectsFrequenciesOutOfRange` und `PacemakerTest.testSetHeartRateAppliesFrequenciesInsideRange`.
Die beiden Testfälle prüfen die Methode `Pacemaker.setHeartRate` darauf, ob sie korrekt reagiert, wenn das Herz die gewünschte Frequenz im einen Fall anwenden und im anderen Fall nicht anwenden kann. In diesem Szenario ist die Klasse `Pacemaker` die Class-Under-Test und die Klasse `Heart` eine Abhängigkeit für `Pacemaker`.
Da Pacemaker unabhängig von `Heart` getestet werden soll (bzw. der Test auch funktionieren soll, wenn `Heart` noch nicht implementiert ist), verwenden Sie für die Klasse `Heart` ein entsprechendes Mock-Objekt und stubben Sie das Verhalten gemäss der Spezifikation der Methode `Heart.setHeartRate()`. Eine Voraussetzung ist, dass Sie für beide Testfälle dasselbe Mock-Objekt mit demselben Stubbing verwenden können. Die Callback-Methode wird Ihnen dabei eine Hilfe sein.

Abschluss

Stellen Sie sicher, dass die Pflichtaufgaben mittels `gradle run` gestartet werden können, die Tests mit `gradle test` erfolgreich laufen und pushen Sie die Lösung vor der Deadline in Ihr Abgaberepository.