Hausaufgabenblatt 03

Aufgabe 3 – Tests

Testen – Wozu?

- Schaden ca. 84 Milliarden Dollar jährlich in Unternehmen wegen Softwarefehlern

Berühmte Beispiele:

- Röntgengerät verwendete bis zu 100fache Menge an Strahlung
- Ariana-5 Rakete explodierte kurz nach dem Start

Softwarefehler

Art des Fehlers	Beschreibung
Syntaxfehler	Code entspricht nicht Syntax der Sprache (wird vom Compiler angezeigt)
semantischer Fehler	z.B. Verwendung falscher Parameter bei Aufrufen, Klammern an der falschen Stelle
logischer Fehler	Programm lässt sich ausführen, Verhalten entspricht aber nicht der Spezifikation
Designfehler	Spezifikation ist falsch

JUnit

- Framework zum Testen von Java Programmen
- vordefinierte Testfunktionen, mit denen man Werte von Variablen und Rückgabewerte von Funktionen überprüfen kann

JUnit – Annotations

`@BeforeEach`:

- die folgende Funktion wird vor jedem Test ausgeführt (z.B. zum Initialisieren von Variablen und zum Erstellen von Objekten)

`@BeforeAll`:

- die folgende Funktion wird einmal vor den Tests ausgeführt

`@Test`:

- die folgende Funktion ist eine Testfunktion

JUnit – Asserts I

- überprüfen, ob etwas wahr ist:
  `assertTrue(<Wahrheitswert>, <Nachricht>);`
- überprüfen, ob etwas falsch ist:
  `assertFalse(<Wahrheitswert>, <Nachricht>);`
- überprüfen, ob etwas `null` ist:
  `assertNull(<Objekt>, <Nachricht>);`
- überprüfen, ob etwas nicht `null` ist:
  `assertNotNull(<Objekt>, <Nachricht>);`

JUnit – Asserts II

- überprüfen, ob zwei Werte gleich sind:
  `assertEquals(<Wert1>, <Wert2>, <Nachricht>);`
- überprüfen, ob zwei Werte nicht gleich sind:
  `assertNotEquals(<Wert1>, <Wert2>, <Nachricht>);`
- den Test fehlschlagen lassen:
  `fail( );`

Was sollte man testen?

„Ein Softwaretester kommt in eine Bar und…

… bestellt ein Bier.

… bestellt 0 Biere.

… bestellt 2147483647 Biere.

… bestellt -1 Biere.

… bestellt einen Hund.

… bestellt ein "Bier".“

Was sollte man testen?

- Initialisierung von Variablen und Objekten
- Beispielfälle
- Randfälle

JUnit – Testklasse in IDEA erstellen I

1. In der Klasse die getestet werden soll Rechtsklick auf den Klassennamen und *Show Context Actions* auswählen.
2. In dem Menü *Create Test* auswählen und bestätigen, dass die Testklasse im gleichen Verzeichnis erstellt werden soll.
3. *JUnit5* als *Testing library* auswählen.
4. Wenn die *JUnit5 library* nicht gefunden werden konnte, *Fix* klicken und anschließend mit *Ok* bestätigen.
5. Gegebenenfalls weitere Einstellungen vornehmen dann mit *Ok* bestätigen.

JUnit – Testklasse in IDEA erstellen II

6. Falls JUnit noch nicht korrekt zum Projekt hinzugefügt wurde, wird der Import in der ersten Zeile rot markiert. Um JUnit hinzuzufügen, *junit* anklicken, Alt+Enter auf der Tastatur drücken und anschließend *Add JUnit5.4 to classpath* auswählen.
7. Nun können Testmethoden mit den entsprechenden Annotations geschrieben werden.

Aufgabe

Testet nun mithilfe von JUnit die Funktionalität des Shops. Schreibt dazu eine neue Testklasse und verwendet die eben vorgestellten Funktionen von JUnit.

Aufgabe

Welche Aspekte der Klasse sollten getestet werden?

```
public class StringGenome {
	private String s = "";

	public void addNucleotide(char c) {
		if (c == 'A'  c == 'C'  c == 'G'  c == 'T')
			s = s + c;
		else
			throw new RuntimeException("Illegal nucleotide");
	}

	public char nucleotideAt(int i) {
		if (i < s.length())
			return s.charAt(i);
		else
			throw new RuntimeException("Genome out of bounds");
	}

	public int length() {
		return s.length();
	}

	public String toString() {
		return s;
	}

	@Override
	public boolean equals(Object obj) {
		StringGenome i = (StringGenome) obj;
		return i.s.equals(this.s);
	}

	@Override
	public int hashCode() {
		return this.s.hashCode();
	}
}
```

Lösung

- Initialisierung

```
public class StringGenome {
        private String s = "";

        public void addNucleotide(char c) {
                if (c == 'A'  c == 'C'  c == 'G'  c == 'T')
                        s = s + c;
                else
                        throw new RuntimeException("Illegal nucleotide");
        }

        public char nucleotideAt(int i) {
                if (i < s.length())
                        return s.charAt(i);
                else
                        throw new RuntimeException("Genome out of bounds");
        }

        public int length() {
                return s.length();
        }

        public String toString() {
                return s;
        }

        @Override
        public boolean equals(Object obj) {
                StringGenome i = (StringGenome) obj;
                return i.s.equals(this.s);
        }

        @Override
        public int hashCode() {
                return this.s.hashCode();
        }
}
```

Lösung

- Funktion für Parameter: A, C, G, T
- Funktion für andere Parameter

```
public class StringGenome {
        private String s = "";

        public void addNucleotide(char c) {
                if (c == 'A'  c == 'C'  c == 'G'  c == 'T')
                        s = s + c;
                else
                        throw new RuntimeException("Illegal nucleotide");
        }

        public char nucleotideAt(int i) {
                if (i < s.length())
                        return s.charAt(i);
                else
                        throw new RuntimeException("Genome out of bounds");
        }

        public int length() {
                return s.length();
        }

        public String toString() {
                return s;
        }

        @Override
        public boolean equals(Object obj) {
                StringGenome i = (StringGenome) obj;
                return i.s.equals(this.s);
        }

        @Override
        public int hashCode() {
                return this.s.hashCode();
        }
}
```

Lösung

- Funktion für Parameter:
  $$i \in [0, length[$$
- Funktion für Parameter:
  $$i \geq length$$
- Funktion für Parameter:
  $$i < 0$$
- Funktion auf leerem String

```
public class StringGenome {
	private String s = "";

	public void addNucleotide(char c) {
		if (c == 'A'  c == 'C'  c == 'G'  c == 'T')
			s = s + c;
		else
			throw new RuntimeException("Illegal nucleotide");
	}

	public char nucleotideAt(int i) {
		if (i < s.length())
			return s.charAt(i);
		else
			throw new RuntimeException("Genome out of bounds");
	}

	public int length() {
		return s.length();
	}

	public String toString() {
		return s;
	}

	@Override
	public boolean equals(Object obj) {
		StringGenome i = (StringGenome) obj;
		return i.s.equals(this.s);
	}

	@Override
	public int hashCode() {
		return this.s.hashCode();
	}
}
```

Lösung

- Funktion für Parameter: `null`
- Funktion für Parameter: kein `StringGenome` Objekt
- Funktion für Parameter: leere Strings
- Funktion für Parameter: gleiche Strings
- Funktion für Parameter: unterschiedliche Strings

```
public class StringGenome {
        private String s = "";

        public void addNucleotide(char c) {
                if (c == 'A'  c == 'C'  c == 'G'  c == 'T')
                        s = s + c;
                else
                        throw new RuntimeException("Illegal nucleotide");
        }

        public char nucleotideAt(int i) {
                if (i < s.length())
                        return s.charAt(i);
                else
                        throw new RuntimeException("Genome out of bounds");
        }

        public int length() {
                return s.length();
        }

        public String toString() {
                return s;
        }

        @Override
        public boolean equals(Object obj) {
                StringGenome i = (StringGenome) obj;
                return i.s.equals(this.s);
        }

        @Override
        public int hashCode() {
                return this.s.hashCode();
        }
}
```

Lösung

- Funktion für leeren String s
- Funktion für beliebigen String

```
public class StringGenome {
        private String s = "";

        public void addNucleotide(char c) {
                if (c == 'A'  c == 'C'  c == 'G'  c == 'T')
                        s = s + c;
                else
                        throw new RuntimeException("Illegal nucleotide");
        }

        public char nucleotideAt(int i) {
                if (i < s.length())
                        return s.charAt(i);
                else
                        throw new RuntimeException("Genome out of bounds");
        }

        public int length() {
                return s.length();
        }

        public String toString() {
                return s;
        }

        @Override
        public boolean equals(Object obj) {
                StringGenome i = (StringGenome) obj;
                return i.s.equals(this.s);
        }

        @Override
        public int hashCode() {
                return this.s.hashCode();
        }
}
```