# Objektorientierte Programmierung

## Übungsaufgaben vom 20.03.2015

## Thema 1: Enums

### 1. Kopfrechentrainer

Bearbeiten Sie den Konstruktor der TestRun-Klasse so, dass zwei Parameter übergeben werden müssen:

- die Anzahl der Zahlen, die während eines Testlaufs generiert werden
- die Rechenoperation, die getestet werden soll

Die möglichen Rechenoperationen sind Addition, Subtraktion und Multiplikation. Definieren Sie in einer extra Datei ein enum mit den Einträgen ADD, SUB und MUL. Verwenden Sie das enum als Parameter- und Attributtyp in der Klasse TestRun!

Passen Sie den Code so an, dass alle Rechenoperationen korrekt unterstützt werden.

## Thema 2: Collections

### 2. MyArrayList

Laden Sie die Dateien MyArrayList.java und MyArrayListTest.java herunter und integrieren Sie diese in ein eclipse-Projekt. Stellen Sie sicher, dass der JUnit-Test (Run As → JUnit Test Case) ausgeführt werden kann.

Implementieren Sie folgende Methoden der List-Schnittstelle in der Klasse MyArrayList:

- `Integer remove(int index)`
- `void add(int index, Integer element)`
- `boolean contains(Object o)`
- `int indexOf(Object o)`
- `int lastIndexOf(Object o)`

Durch Klick auf die Links gelangen Sie zur Java-Dokumentation der Methoden.

Implementieren Sie für jede Methode einen JUnit-Test in der Klasse MyArrayListTest.java!

# Thema 3: Implementierung von Objektbeziehungen

## 3. Personenbeziehungen

**Teil 1**

1. Definieren Sie ein enum „Geschlecht“ mit den Einträgen M und W.
2. Definieren Sie eine Klasse „Person“ mit den Attributen „vorname“, „nachname“, „alter“ und „geschlecht“. Ergänzen Sie für alle Attribute die Modifikatoren „private“.
3. Ergänzen Sie einen Konstruktor, der die Werte aller Attribute als Parameter entgegennimmt.
4. Ergänzen Sie Getter-Methoden für alle Attribute.

**Teil 2**

1. Ergänzen Sie in der Klasse „Person“ die privaten Attribute „vater“ und „mutter“ vom Typ „Person“.
2. Ergänzen Sie ein privates Attribut „kinder“ vom Typ „java.util.List<Person>“. Initialisieren Sie die Liste als „ new java.util.ArrayList<Person>()“.
3. Erweitern Sie den Konstruktor so, dass Vater und Mutter ebenfalls als Parameter übergeben werden müssen.
4. Ergänzen Sie eine Methode, mit der Kinder hinzugefügt werden können.
5. Ergänzen Sie eine Methode, mit der die Liste aller Kinder abgefragt werden kann.

**Teil 3**

1. Erstellen Sie eine neue Schnittstelle „IPerson“ und kopieren Sie dorthinein den Code der nachfolgenden Seite.
2. Implementieren Sie in der Klasse Person die Schnittstelle „IPerson“.
3. Erstellen Sie eine JUnit-Testklasse „PersonTest“ und implementieren Sie dort für jede Schnittstellenmethode ein oder mehrere Testmethoden!

**Hinweise:**

- Arbeiten Sie die Methoden von oben nach unten ab. Einige Methoden können Sie in den nachfolgenden Methoden wiederverwenden. Auf diese Weise lässt sich die Implementierung z.T. stark vereinfachen. So können Sie z.B. die Methode „getGeschwister“ gut gebrauchen, um die Methode „sindGeschwister“ zu implementieren.
- Sie können sich die Arbeit z.T. erleichtern, indem Sie geeignete Methoden der java.util.List-Schnittstelle verwenden. Nützliche Methoden sind z.B.:
    - **boolean** `contains(Object o)` → z.B. für “sindGeschwister”
    - **boolean** `addAll(Collection c)` → z.B. für “getCousinen…”

Informieren Sie sich in der Java-Dokumentation über diese Methoden und fragen Sie den Übungsleiter, wenn Sie etwas nicht verstehen!

- Zum Testen benötigen Sie ein paar „Urmütter“ bzw. „Urväter“. Für diese können Sie im Konstruktor für die Parameter „vater“ und „mutter“ den Wert null angeben. Die restlichen Personen erzeugen Sie stets mit der Methode „zeugeKind“.

```java
import java.util.List;

public interface IPerson {

	/*
	 * Nachbedingungen: Die erzeugte Person hat einen Vater und eine Mutter
	 *(abhängig vom Geschlecht ist die Person partner oder die Person this
	 * Vater bzw. Mutter), den Vornamen vorname, das Alter 0 und das
	 * Geschlecht g.
	 * Vater und Mutter enthalten die erzeugte Person in ihrer Liste kinder.
	 */
	Person zeugeKind(Person partner, String vorname, Geschlecht g);

	/*
	 * Gibt die Kinder vom Vater und die Kinder der Mutter von Person this zurück.
	 */
	List<Person> getGeschwister();

	/*
	 * Gibt true zurück, wenn Person p und die Person this Geschwister sind,
	 * ansonsten false.
	 */
	boolean sindGeschwister(Person p);

	/*
	 * Gibt true zurück, wenn die Anzahl der Geschwister von Person this 0 beträgt.
	 */
	boolean istEinzelkind();

	/*
	 * Gibt den Vater des Vaters und den Vater der Mutter von Person this zurück.
	 */
	List<Person> getGroßvaeter();

	/*
	 * Gibt die Schwestern des Vaters von Person this zurück.
	 */
	List<Person> getTantenVaeterlicherseits();

	/*
	 * Gibt alle Kinder der Geschwister des Vaters von Person this zurück.
	 */
	List<Person> getCousinenVaeterlicherseits();

}
```