Übung 0 – Grundlagen

Dieses Übungsblatt dient dazu, einige Inhalte aus dem vorangegangenen Semester zu wiederholen und so die Grundlagen für diese Lehrveranstaltung aufzufrischen. Sollten Ihnen die Inhalte nicht mehr präsent sein, oder wurden sie möglicherweise im früheren Semester garnicht gelehrt, so können Sie diese im Skript und/oder in Java ist auch eine Insel nachlesen.

Aufgabe 1 (Theorie)

Setzen Sie sich im Folgenden mit der Thematik Annotationen in Java auseinander. Suchen Sie sich dafür Informationen im Skript und in der Literatur und beantworten Sie folgende Fragen:

- 1. Was sind Annotationen in Java?
- 2. Warum werden Annotationen verwendet?
- 3. Was bewirkt die Annotation @Override?

Aufgabe 2

Modellieren Sie im Folgenden eine Art von Speicher sowie einen Kellerspeicher (Stack/-Stapel) und eine Warteschlange als konkrete Umsetzungen. Wir beschränken uns darauf, dass nur ganzzahlige Werte gespeichert werden sollen.

Hierzu definieren wir zunächst eine Schnittstelle Puffer, die generelle Operationen auf einem Speicher vorgeben soll. Die Schnittstelle soll die Methoden isEmpty, size und capacity bereitstellen. Alle drei Methoden haben eine leere Parameterliste. isEmpty soll einen Wahrheitswert zurückgeben, der besagt, ob der Speicher leer ist. size und capacity sollen jeweils ganzzahlige Werte zurückgeben. Die Methode size soll in ihrer konkreten Umsetzung dazu dienen die aktelle Größe des Speichers zu liefern (= Anzahl belegter Speicherplätze). capacity soll dazu dienen die maximale Größe des Speichers zu liefern. Weiterhin soll eine Methode insert vorgegeben werden, deren Funktion es sein soll, ein Element anzunehmen und in den Speicher einzufügen. Sollte der Speicher beim Aurfuf von insert voll sein, soll eine java.lang.IllegalStateException geworfen werden. Zuletzt soll eine Methode remove deklariert werden, die dazu dienen soll. ein Element aus dem Speicher zu entnehmen. Sollte der Speicher beim Aurfuf von remove leer sein, soll eine java.util.NoSuchElementException geworfen werden.

Die Schnittstelle Stapel soll die Schnittstelle Puffer erweitern. Zusätzlich soll diese Schnittstelle noch die Methode top deklarieren, die das aktuell oberste Element auf dem Stapel liefert, ohne es zu entfernen. Sollte der Stapel beim Aurfuf von top leer sein, soll eine java. Util. NoSuchElementException geworfen werden.

Die Schnittstelle Schlange soll die Schnittstelle Puffer erweitern. Zusätzlich soll diese Schnittstelle noch die Methode front deklarieren, die das aktuell vorderste Element in der Schlange liefert, ohne es zu entfernen. Sollte die Schlange beim Aurfuf von front leer sein, soll eine java. Util. NoSuchElementException geworfen werden.

Die konkreten Klassen StapelMitArray und SchlangeMitArray sollen die Schnittstellen Stapel bzw. Schlange implementieren. Intern sollen jeweils ganzzahlige Werte in einem Array gespeichert werden. Stellen Sie jeweils einen Konstruktor zur Verfügung, der einen ganzzahligen Wert maxGroesse annimmt und ein Array mit dieser Größe anlegt.

Hinweis: Überlegen Sie was Sie im Array intern machen müssen, damit bei der Schlange das aktuelle Element am Anfang des Arrays steht.

Gehen Sie wie folgt vor:

- Skizzieren Sie diese kleine Klassenhierarchie in einem UML-Diagramm. Überlegen Sie sich die Parameter und Rückgabewerte die Methoden haben müssen und welche Datentypen diese haben. Überlegen Sie auch welche Zugriffsrechte sinnvoll sind.
- Machen Sie sich die Funktionalität eines Stack und einer Schlange klar. Skizzieren die Funktionsweise der Methoden der Klasse.
- Schreiben Sie den Quellcode der Schnittstellen und Klassen

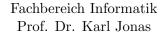
Hinweis: Dieses kleine Softwaredesign dient uns als Grundlage für spätere Übungen. Es soll eine möglichst einfache Hierarchie sein, die möglichst modular ist. Grundsätzlich könnte man das (und spätere Funktionalitäten) auch anders modellieren.

Aufgabe 3

Schreiben Sie eine abstrakte Klasse Person. Diese Klasse hat als private Eigenschaften einen Namen und Vornamen. In einem Konstruktor sollen zuerst der Name, dann der Vorname übergeben werden. über die Methoden getName und getVorname sollen diese Eigenschaften nach aussen kommuniziert werden können. In der Methode toString soll folgende Ausgabe gemacht werden: <Name>, <Vorname>. Also zuerst der Name gefolgt von einem Komma und einem Leerzeichen und dann dem Vornamen.

Schreiben Sie eine Klasse Student, die die Klasse Person erweitert. Als zusätzliche private Eigenschaft gibt es eine ganzzahlige Matrikelnummer. Es soll einen Konstruktor für Name, Vorname, Matrikelnummer geben. Ausserdem soll es eine Methode getMatrikel geben. Die toString Methode soll nach Name und Vorname (siehe oben) noch ein Leerzeichen und die Matrikelnummer ausgeben.

Schreiben Sie eine Klasse Boxer, die auch die Klasse Person erweitert. ähnlich zur Matrikelnummer in Student soll sie eine ganzzahlige Eigenschaft Gewicht (in kg) haben. Stellen Sie entsprechend einen Konstruktor und Getter zur Verfügung.





Implementieren Sie auch jeweils die equals-Methode für diese Klassen. Eine Person soll nach Name und Vorname verglichen werden. Studenten sollen zusätzlich nach Matrikelnummer verglichen werden, Boxer zusätzlich nach Gewicht. Überlegen Sie wie Sie geschickt in den Unterklassen den Vergleich der Oberklasse machen können.