

# Funktionale und objektorientierte Programmierkonzepte

## Übungsblatt 09



TECHNISCHE  
UNIVERSITÄT  
DARMSTADT

Prof. Karsten Weihe

Wintersemester 23/24

Themen:

Relevante Foliensätze:

Abgabe der Hausübung:

v1.0

<Themen>

<1>

XX.XX.202X bis 23:50 Uhr

Hausübung 09

<Übungstitel>

Gesamt: 27 Punkte

**Beachten Sie die Seite *Verbindliche Anforderungen für alle Abgaben im Moodle-Kurs*.**

Verstöße gegen verbindliche Anforderungen führen zu Punktabzügen und können die korrekte Bewertung Ihrer Abgabe beeinflussen. Sofern vorhanden, müssen die in der Vorlage mit TODO markierten crash-Aufrufe entfernt werden. Andernfalls wird die jeweilige Aufgabe nicht bewertet.

Die für diese Hausübung relevanten Verzeichnisse sind `src/main/java/h09` und ggf. `src/test/java/h09`.

**Verbindliche Anforderung:**

- Der Typ `X` dient als Platzhalter für einen *beliebigen* Typen und existiert in der Vorlage nicht. Sofern nicht anders angegeben, *muss* Ihre Umsetzung für *jeden* beliebigen Typen funktionieren.
- Bei der Instanziierung eines Typparameters *muss* der Typ möglichst genau angegeben werden.

## Einleitung

---

H1: <Aufgabentitel>?? Punkte

---

Im Package `h09.stack` finden Sie die Klasse `StackOfObject`. Ein Objekt der Klasse `StackOfObject` stellt einen Stack (deutsch: „Stapel“) – eine dynamische<sup>1</sup> Datenstruktur, welche mit einem Stapel vergleichbar ist – dar. Das Modifizieren eines Stack ist auf das *Ablegen* eines Objekts auf den Stapel und das *Entfernen* des obersten Objekts von dem Stapel beschränkt. Die Klasse `StackOfObject` implementiert für beide Operationen folgende Objektmethoden: Die rückgabelose Objektmethode `push` legt den aktuellen Parameter auf dem Stapel ab, die parameterlose Objektmethode `pop` entfernt das oberste Objekt von dem Stapel und liefert dieses Objekt.

---

H1.1:?? Punkte

---

Überführen Sie die Klasse `StackOfObjects` in eine generische Klasse. Die Klasse `StackOfObjects` wird zuerst um einen unbeschränkten Typparameter `T` ergänzt. Der formale Parameter der Methode `push` wird so ausgetauscht, dass als aktueller Parameter Objekte vom Typ `T` und Subtypen verwendet werden können. Der Rückgabetypp der Methode `pop` wird so ausgetauscht, dass die Methode Objekte vom Typ `T` und Subtypen liefern kann. Innerhalb der Methode `pop` wird zuletzt für das gelieferte Objekt ein geeigneter Cast durchgeführt.

**Unbewertete Verständnisfrage:**

Analysieren Sie die Funktionsweise der Klasse `StackOfObjects`. Die Klasse `StackOfObjects` hat ein Objektattribut `objects` vom statischen Typ „Array von `Object`“, welches die Objekte des Stack enthält. Warum kann der statische Typ dieses Attributs nicht durch „Array von `T`“ ausgetauscht werden?

---

H1.2:?? Punkte

---

Überführen Sie die Klassenmethode `of` der Klasse `StackOfObjects` in eine generische Methode. Die Methode `of` soll für einen aktuellen Parameter vom Typ „Array von `X`“ ein Objekt der Klasse `StackOfObjects` vom Typ `X` liefern. Passen Sie die innerhalb der Methode verwendeten Typen entsprechend an.

---

H1.3:?? Punkte

---

Erstellen Sie im Package `h09.stack` eine generische `public`-Klasse `StackOfRoomsWithSeats`, welche einen Typparameter `T` hat und direkt von der Klasse `StackOfObjects` abgeleitet ist. Der Typparameter `T` ist auf Typen beschränkt, die die Interfaces `Room` und `WithSeats` erweitern oder implementieren. Der Typparameter `T` der Basisklasse `StackOfObjects` wird mit `T` instanziiert.

---

<sup>1</sup>In diesem Kontext bedeutet *dynamisch*, dass neue Objekte hinzugefügt und bestehende Objekte entfernt werden können.

---

**H2: Operationen****?? Punkte**

---

**H2.1:****?? Punkte**

---

Überführen Sie die rückgabefreie Klassenmethode `filter` in eine generische Klassenmethode mit einem Typparameter `T`.

Instanzieren Sie die Typparameter der formalen Parameter so, dass (1) der erste aktuelle Parameter ein beliebiger Stack sein kann, aus welchem Objekte des Typs `T` gelesen werden können, (2) der zweite aktuelle Parameter ein beliebiger Stack sein kann, in welchen Objekte des Typs `T` geschrieben werden können und (3) der dritte aktuelle Parameter ein beliebiger Filter sein kann, welcher auf Objekte des Typs `T` angewendet werden kann. Passen Sie die innerhalb der Methode verwendeten Typen entsprechend an.

**H2.2:****?? Punkte**

---

Überführen Sie die rückgabefreie Klassenmethode `map` in eine generische Klassenmethode mit zwei Typparametern `O` und `I`.

Instanzieren Sie die Typparameter der formalen Parameter so, dass (1) der erste aktuelle Parameter ein beliebiger Stack sein kann, aus welchem Objekte des Typs `O` gelesen werden können, (2) der zweite aktuelle Parameter ein beliebiger Stack sein kann, in welchen Objekte des Typs `I` geschrieben werden können und (3) der dritte aktuelle Parameter eine beliebige Funktion sein kann, welche ein Objekt des Typs `O` auf ein Objekt des Typs `I` abbilden kann. Passen Sie die innerhalb der Methode verwendeten Typen entsprechend an.

---

**H3: Eigene Interfaces****?? Punkte**

---

**H3.1:****?? Punkte**

---

**H3.2:****?? Punkte**

---

---

H4: Funktionen?? Punkte

---

## H4.1: IsNullPredicate

?? Punkte

---

Aus Foliensatz 06 (Folien 33 bis 35) kennen Sie das generische Interface `Predicate`. In der Klasse `Functions` finden Sie eine Klassenkonstante `IS_NULL_PREDICATE` vom statischen Typ `Predicate`. Das von `IS_NULL_PREDICATE` dargestellte `Predicate` liefert für ein Objekt jeder beliebigen Klasse genau dann `true`, wenn der aktuelle Parameter dieses `Predicate` `null` ist.

Überführen Sie die den statischen, nicht-instanzierten Typ der Klassenkonstante `IS_NULL_PREDICATE` in einen instanziierten Typ.

---

H4.2: RemainingValueFunction?? Punkte

---

Ein Objekt einer Klasse, die das im Package `java.util.function` gegebene generische und funktionale Interface `Function` implementiert, dient der Darstellung einer Funktion. Das Interface `Function` hat zwei unbeschränkte Typparameter: Der erste Typparameter `T` wird mit dem Typ der *Eingabe* instanziiert, der zweite Typparameter `R` wird mit dem Typ der *Ausgabe* instanziiert.

In der Klasse `Functions` finden Sie die Klassenmethode `remainingValueFunction` mit einem formalen Parameter `divisor` vom Typ `double`. Die Methode `remainingValueFunction` liefert für den aktuellen Parameter – im Folgenden auch *Divisor* genannt – eine Funktion, die als Eingabe ein Objekt der Klasse `Number` bzw. ein Objekt einer beliebigen von Klasse `Number` abgeleiteten Klasse erhält und ein Objekt der Klasse `Float` liefert. Die gelieferte Funktion liefert den bei der Division der gegebenen Zahl durch den *Divisor* verbleibenden Rest.

Überführen Sie den nicht-instanzierten Rückgabetypp der Methode `remainingValueFunction` in einen instanziierten Typ. Sie können die *erste* `return`-Anweisung *einkommentieren* und die *zweite* `return`-Anweisung *auskommentieren*, um die Funktion zu testen.

**Anmerkung:**

Sie können beim Ausprobieren der Funktion auftretende Ungenauigkeiten des Rechenergebnisses ignorieren.

---

H5: Testen mittels JUnit?? Punkte

---

TODO Snippet, dass Tests in Verzeichnis `test` erstellt werden müssen.

---

H5.1: Test von filter?? Punkte

---

TODO

---

H5.2: Test von map?? Punkte

---

TODO