Funktionale und objektorientierte Programmierkonzepte Übungsblatt 09

Prof. Karsten Weihe

Wintersemester 23/24	v1.0.3
Themen:	Generics
Relevante Foliensätze:	05+06
Abgabe der Hausübung:	12.01.2023 bis 23:50 Uhr

Hausübung 09 — Gesamt: 32 Punkte

Generics

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:

- Die Typen `X` und `Y` dienen als Platzhalter für *beliebige* 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

Mit diesem Übungsblatt befassen Sie sich mit der Datenstruktur *Stack*[1] und dazugehörigen Funktionen, welche von Ihnen in generische Varianten zu überführen sind. Ein *Stack* (deutsch: „Stapel“) ist eine dynamische[2] Datenstruktur, welche mit einem Stapel vergleichbar ist. Das Modifizieren eines Stack ist auf (1) das Ablegen (*Push*-Operation) eines Objekts auf den Stapel und (2) das *Entfernen* mit gleichzeitigem *Lesen* (*Pop-Operation*) des obersten Objekts von dem Stapel beschränkt.

[1] Siehe auch https://de.wikipedia.org/wiki/Stapelspeicher.
[2] In diesem Kontext bedeutet *dynamisch*, dass neue Objekte hinzugefügt und bestehende Objekte entfernt werden können.

H1: Stacks of Objects

Im Package `h09.stack` finden Sie die Klasse `StackOfObjects`, wobei ein Objekt der Klasse `StackOfObjects` einen Stack darstellt.

Die Klasse `StackOfObjects` implementiert die in der Einleitung genannten Operationen Push und Pop mittels der gleichnamigen Objektmethoden: Die rückgabelose Objektmethode `push` legt das mittels dem aktualen Parameter gegebene Objekt auf dem Stapel ab, die parameterlose Objektmethode `pop` entfernt das oberste Objekt von dem Stapel und liefert dieses Objekt.

Die Klasse `StackOfObjects` implementiert mittels folgender Objektmethoden darüber hinaus folgende Operationen: Die parameterlose Methode `numberOfObjects` liefert die Anzahl n an Objekten auf dem Stack. Die Methode `get` hat einen formalen Parameter vom Typ `int` und liefert für den aktualen Parameter – den *Index* – das am gegebenen Index enthaltene Objekt. Das *unterste* Objekt befindet sich an Index 0, das *oberste* Objekt befindet sich an Index $n - 1$.

H1.1: Klasse `StackOfObjects` 7 Punkte

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

Überführen Sie die Klassenmethode `of` der Klasse `StackOfObjects` in eine generische Methode. Die Methode `of` soll für einen aktualen 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.

Hinweis:

Der formale Parameter der Methode `of` ist vom Typ `Object...` (*Variable Arguments* von `Object`). Innerhalb der Methode `of` können Sie den aktualen Parameter so benutzen, als wäre der formale Parameter vom Typ `Object[]`. Die Verwendung von *Variable Arguments* hat den Vorteil, dass die Methode mit beliebig vielen aktualen Parametern aufgerufen werden kann, welche innerhalb der Methode über den einzelnen aktualen Parameter vom Typ Array abgerufen werden können.

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?

H2: Stack Operations

H2.1: Methode `filter` — 5 Punkte

In der Klasse `StackOfObjectOperations` finden Sie die nicht-generische Klassenmethode `filter`. Ein Fallbeispiel zur filter-Operation in Racket finden Sie in Kapitel 04c, ab Folie 116 der FOP.

Überführen Sie die Methode in eine generische Methode. Instanziieren Sie die Typparameter der formalen Parameter so, dass der erste aktuelle Parameter ein *beliebiger* Stack sein kann, aus dem Objekte des Typs `X` gelesen werden können, und der zweite aktuelle Parameter ein *beliebiger* Filter sein kann, der auf Objekte des Typs `X` angewendet werden kann. Die Rückgabe der Methode soll einem *beliebigen* Stack zugewiesen werden können, der Objekte des Typs `Y` enthalten kann. Passen Sie die innerhalb der Methode verwendeten Typen entsprechend an.

> **Hinweis:**
>
> Die Rückgabe der Methode `filter` ist ein Stack vom Typ `Y`, um eine möglichst hohe Kompatibilität zu ermöglichen. Wäre die Rückgabe der Methode `filter` ein Stack vom Typ `X`, so könnte die Methode nur folgende Filter abbilden:
>
> - Eingabe: `StackOfObjects<Integer> input = ...;`
> - Ausgabe: `StackOfObjects<Integer> result = filter(input, x -> x > 0);`
>
> Es sollte jedoch auch folgende Filter möglich sein:
>
> - Eingabe: `StackOfObjects<Integer> input = ...;`
> - Ausgabetyp kann `Integer`, `Number`, ...oder `Object` sein.
> - Beispiel Ausgabe: `StackOfObjects<Number> result = ...;`

H2.2: Methode `map` — 6 Punkte

In der Klasse `StackOfObjectOperations` finden Sie die nicht-generische Klassenmethode `map`. Sie finden in Kapitel 04c, ab Folie 130 der FOP ein Fallbeispiel zur map-Operation in Racket.

Überführen Sie die Methode in eine generische Methode. Instanziieren Sie die Typparameter der formalen Parameter so, dass der erste aktuale Parameter ein *beliebiger* Stack sein kann, aus welchem Objekte des Typs `X` gelesen werden können und der zweite aktuale Parameter eine *beliebige* Funktion sein kann, welche ein Objekt des Typs `X` auf ein Objekt des Typs `Y` abbilden kann. Die Rückgabe soll einem *beliebigen* Stack zugewiesen werden können, welcher Objekte des Typs `Y` enthalten kann. Passen Sie die innerhalb der Methode verwendeten Typen entsprechend an.

H2.3: Interface `StackOfObjectsPredicate` — 2 Punkte

Im Package `h09.function` finden Sie ein Interface `StackOfObjectsPredicate`, welches das Interface `Predicate` erweitert.

Überführen Sie das Interface `StackOfObjectsPredicate` in ein generisches Interface und instanziieren Sie den Typen `Predicate`, welchen das Interface `StackOfObjectsPredicate` erweitert.

Eine Predicate, welches mittels eines Objekts der Klasse `StackOfObjectsPredicate` dargestellt wird, erhält als Eingabe ein Objekt der Klasse `StackOfObjects`. Ein solcher Stack of Objects enthält nur Objekte von dem Typ, mit

dem der Typparameter des Interface `StackOfObjectsPredicate` instanziiert wurde.

H3: Room Functions

Im Package `h09` finden Sie das Interface `WithSeats` und im Package `h09.room` das Interface `Room`.

Ein Objekt einer Klasse, welche das Interface `WithSeats` implementiert, stellt ein Subjekt dar, welches mit Plätzen ausgestattet ist. Das Interface `WithSeats` deklariert die parameterlose Objektmethode `numberOfSeats`, welche die Anzahl der Plätze des jeweiligen Subjekts liefert. Ein Objekt einer Klasse, welche das Interface `Room` implementiert, stellt einen Raum dar. Das Interface `Room` deklariert die parameterlose Objektmethode `name`, welche den Namen des jeweiligen Raums liefert.

Die `record`-Klassen `LectureHall` und `SeminarRoom` implementieren jeweils beide Interfaces. Ein Objekt der Klasse `LectureHall` stellt einen Hörsaal dar, ein Objekt der Klasse `SeminarRoom` stellt einen Seminarraum dar. Beide Klassen haben jeweils einen Konstruktor mit jeweils einem ersten formalen Parameter vom Typ `String` und jeweils einem zweiten formalen Parameter vom Typ `int`. Der erste aktuale Parameter des jeweiligen Konstruktors ist der Name, der zweite aktuale Parameter des jeweiligen Konstruktors ist die Anzahl der Plätze des jeweiligen Hörsaals bzw. Seminarraums.

H3.1: Predicate `IS_NULL_PREDICATE` — 1 Punkt

In der Klasse `RoomFunctions` finden Sie eine Klassenkonstante `IS_NULL_PREDICATE` vom statischen Typ `Predicate`. Das von `IS_NULL_PREDICATE` dargestellte Predicate liefert für ein Objekt einer beliebigen Klasse genau dann `true`, wenn der aktuale Parameter dieses Predicate `null` ist.

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

H3.2: Methode `isInArea` — 2 Punkte

In der Klasse `RoomFunctions` finden Sie die nicht-generische Klassenmethode `isInArea`.

Die Methode `isInArea` hat einen formalen Parameter vom Typ `char` und den nicht-instanziierten Rückgabetyp `Predicate`. Diese liefert für den aktualen Parameter – im Folgenden auch *Standort-Präfix* genannt – ein Predicate, welches genau dann `true` liefert, wenn das erste Symbol des Namens des gegebenen Raumes gleich dem Standort-Präfix ist. Sie können davon ausgehen, dass ein gegebener Name einen solchen Standort-Präfix besitzt.

Überführen Sie die Methode in eine generische Methode mit instanziiertem Rückgabetyp. Passen Sie die innerhalb der Methode verwendeten Typen entsprechend an.

> **Hinweis:**
>
> Die Klasse `String` hat eine Objektmethode `charAt` mit einem formalen Parameter vom Typ `int`. Diese liefert für einen aktualen Parameter `i` das Symbol an Position `i` in Form eines Wertes vom Typ `char`.

H3.3: Funktion `isInAreaAndHasMinimumNumberOfSeats` 3 Punkte

In der Klasse `RoomFunctions` finden Sie die nicht-generische Klassenmethode `isInAreaAndHasMinimumNumberOfSeats`. Diese Methode hat zwei formale Parameter: Der erste entspricht dem formalen Parameter der Methode `isInArea`, der ist vom Typ `char`. Die Methode `isInAreaAndHasMinimumNumberOfSeats` ruft mit dem ersten aktualen Parameter von der Methode `isInArea` ein Predicate ab und initialisiert mit dem zweiten aktualen Parameter – im Folgen auch *Mindestanzahl an Plätzen* genannt – ein Predicate, welches genau dann `true` liefert, wenn die Anzahl der Plätze des gegebenen Raumes größer oder gleich der Mindestanzahl an Plätzen ist. Die Methode `isInAreaAndHasMinimumNumberOfSeats` verknüpft beide Predicates logisch zu einem Predicate (mittels *AND*) und liefert dieses Predicate.

Überführen Sie die Methode `isInAreaAndHasMinimumNumberOfSeats` in eine generische Methode mit einem instanziierten Rückgabetyp.

H3.4: `toRoomTypeOrNull` 4 Punkte

In der Klasse `RoomFunctions` finden Sie die nicht-generische Klassenmethode `toRoomTypeOrNull`. Diese hat einen formalen Parameter vom Typ `Class` und den Rückgabetyp `Function`. Die Methode `toRoomTypeOrNull` liefert für aktualen Parameter – ein Objekt der Klasse `Class`, welches den *Zieltyp* darstellt – eine Funktion, welche als Eingabe ein Objekt vom Typ `Room` erhält. Wenn der Typ des eingegebenen Objekts vom Zieltyp ist, castet die Funktion das eingegebene Objekt zu einem Objekt des Zieltyps und liefert dieses Objekt. Andernfalls liefert die Funktion `null`.

Überführen Sie die Methode `toRoomTypeOrNull` in eine generische Methode mit einem instanziierten Typ des formalen Parameters und Rückgabetyp.

> **Anmerkung:**
>
> Ein Objekt der generischen Klasse `Class` stellt einen *beliebigen* Typen dar. Bei diesem Typen muss es sich *nicht* um eine Klasse handeln, auch wenn der Name „`Class`“ dies vermuten lässt. Der Typparameter der Klasse `Class` wird mit dem dargestellten Typen instanziiert. Für einen gegebenen Typ `Type` kann das dazugehörige Objekt der Klasse `Class` mittels `Type.class` abgerufen werden.

H4: Testen mittels JUnit

Zuletzt testen Sie – erstmals mittels JUnit – die Funktionalität der in der Klasse `StackOfObjectOperations` gegebenen Klassenmethoden `filter` und `map`.

In der Klasse `TUDa` finden Sie zwei Klassenmethoden: Die Methode `stackOfLectureHalls` liefert einen Stack von Hörsälen an der TU Darmstadt, die Methode `stackOfSeminarRooms` liefert einen Stack von Seminarräumen an der TU Darmstadt.

Instanziieren Sie zuerst die formalen Typparameter des Rückgabetyps der Methoden `stackOfLectureHalls` und `stackOfSeminarRooms` mit dem Typ, welcher jeweils in dem Stack enthalten ist.

Implementieren Sie dann folgende Tests in der jeweiligen Klasse im Modul Verzeichnis `src/test/java/h09`.

H4.1: *Die großen 5, definiert von Stadtmitte und Lichtwiese.* 1 Punkt

Prüfen Sie, ob die Filter-Funktion aus der Aufgabe H2.1 zum Filtern der größten fünf Hörsäle an der TU Darmstadt funktioniert, indem Sie die Methode `filter` mit dem von der Klassenmethode `TUDa.stackOfLectureHalls()` gelieferten Stack und einem geeigneten Predicate aufrufen. Prüfen Sie zuerst mittels der Methode `assertEquals`, ob der von der Methode `filter` gelieferte Stack 5 Elemente enthält. Prüfen Sie dann mittels der Klassenmethode `assertEquals` für *jeden* der fünf größten Hörsäle, ob sich der Hörsaal an der erwarteten Position im Stack befindet, indem Sie den Hörsaal an der erwarteten Position abrufen und den Namen dieses Hörsals mit dem erwarteten Namen vergleichen.

Implementieren Sie diesen Test in der Klasse `Tests` im Package `h09.function` in der Objektmethode `testFilter`.

Die fünf größten Hörsäle an der TU Darmstadt sind – sortiert nach Namen – L402/1, S101/A1, S105/122, S206/030 und S311/08, wobei der kleinste dieser fünf Hörsäle S311/08 mit 372 Plätzen ist.

> **Hinweis:**
>
> Die Sortierung der Hörsäle ist nach dem Namen und nicht nach der Anzahl der Plätze.
>
> Achten Sie ebenfalls darauf, dass die `filter`-Methode die Elemente aus der Eingabe entfernt.

H4.2: Test von map 1 Punkt

Prüfen Sie, ob die Map-Funktion aus der Aufgabe H2.2 zum Abbilden der Seminarräume auf die Anzahlen der Plätze funktioniert. Rufen Sie die Funktion `map` mit dem von der Klassenmethode `TUDa.stackOfSeminarRooms()` gelieferten Stack und einer geeigneten Funktion auf. Prüfen Sie zuerst mittels der Methode `assertEquals`, ob die Anzahl an Zahlen im Stack gleich der erwarteten Anzahl ist. Prüfen Sie dann mittels der Methode `assertEquals` für *jeden* Seminarraum, ob sich die Anzahl der Plätze des Seminarraums an der erwarteten Position im Stack befindet.

Implementieren Sie diesen Test in der Klasse `Tests` im Package `h09.function` in der Objektmethode `testMap`.

> **Hinweis:**
>
> Verwenden Sie einen *zweiten* von der Klassenmethode `TUDa.stackOfSeminarRooms()` gelieferten Stack, um den Vergleich durchzuführen.
>
> Achten Sie ebenfalls darauf, dass die `map`-Methode die Elemente aus der Eingabe entfernt.