UNIVERSITÄT BASEL

University of Basel

Dozent
Prof. Dr. Thomas Vetter
Departement
Mathematik und Informatik
Spiegelgasse 1
CH – 4051 Basel

Assistenten
Marcel Lüthi

Tutoren
Pascal Mafli
Loris Sauter
Linard Schwendener
Clemens Büchner
Florian Spiess
Jonathan Aellen
Lukas Stöckli

Erweiterte Grundlagen der Programmierung (45398-01)

Blatt 12 [8 Punkte]

Vorbesprechung	10. - 14. Dez
Abgabe	17. - 21. Dez

Wir empfehlen Ihnen, dass Sie im Buch "Sprechen Sie Java" das Kapitel 15 lesen, bevor Sie beginnen die Übungen zu lösen.

Aufgabe 1 - Generische Liste und Map [4 Punkte]

In dieser Aufgabe sollen generische Datenstrukturen implementiert werden. In einem Beispielprogramm, welches Sie von der Übungswebseite herunterladen können, werden diese Datenstrukturen verwendet. Es sollen Postleitzahlen mit dem jeweiligen Ortsnamen gespeichert werden. Für eine Postleitzahl soll der Ortsname zurück gegeben werden können.

Als erstes soll eine generische sortierte Liste geschrieben werden. Die Liste soll folgende Eigenschaften haben:

- Die Klasse *SortedList* soll generisch sein und Objekte speichern können, welche *Comparable* implementieren.
- Die Klasse soll eine öffentliche Methode *insert* und eine öffentliche Methode *find* besitzen.
- Die Klasse soll eine dynamisch generierte und sortierte Liste von Knoten zum Speichern der Objekte verwenden.
- Eine innere Klasse soll als Knotentyp verwendet werden.
- Die Klasse soll keine *Exceptions* werfen.
- Der Testcode soll laufen, wenn Sie den Funktionsaufruf und die Funktion *testGenericList* einkommentieren.

Schreiben Sie dann basierend auf Ihrer Liste eine generische Map. Die Map soll folgende Eigenschaften haben:

- Die Klasse *Map* soll generisch sein und jeweils Paare von einem *Key* und einem *Value* Objekt speichern können.
- Eine innere Klasse von *Map* soll einen *Key* und einen *Value* zu einem *Item* Objekt zusammenfassen.

- Die Klasse *Map* soll Ihre generische Liste verwenden, um Objekte vom Typ *Item* zu speichern.
- *Map* soll eine *insert* Methode haben und einen Key und einen Value entgegen nehmen.
- Eine weitere Methode *find* soll für einen Key den Value zurück geben oder eine Exception Werfen.
- Die geworfene Exception soll von der *RuntimeException* erben und *ElementNotFoundException* heissen.
- Der Testcode soll laufen, wenn Sie den Funktionsaufruf und die Funktion *testGenericMap* einkommentieren.

Aufgabe 2 - Lambdas und Streams [4 Punkte]

In dieser Aufgabe lernen Sie etwas kennen, was nicht in der Vorlesung behandelt wurde. Sie lernen das Paket *java.util.stream* kennen. Was Sie implementieren müssen beschränkt sich dabei auf Funktionsinterfaces und Lambdas welche in der Vorlesung behandelt wurden.

Streams sind ein Konzept aus der funktionalen Programmierung. Sie dienen dazu eine Reihe von Operationen auf einer Liste von Objekten hintereinander auszuführen. Dabei können Sie sich vorstellen, dass man mit einer initialen Liste von Objekten beginnt. Jede Operation auf dem Stream erstellt eine neue Liste anhand der vorherigen Liste und der Operation selbst. Dabei nimmt jede Operation einen Parameter von einem genau definierten Funktionsinterface entgegen. In der Aufgabe werden folgende Operationen verwendet:

- **filter** - Entscheidet für jedes Element ob es in die neue Liste übernommen wird oder nicht. Filter erwartet einen Parameter der das Funktionsinterface *Predicat*<*T*> erfüllt.
- **map** - Erstellt aus jedem Objekt ein neues Objekt. Map erwartet einen Parameter der das Funktionsinterface *Function*<*T,R*> erfüllt.
- **sorted** - Sortiert eine ganze Liste nach einer Regel. Sorted erwartet einen Parameter der das Funktionsinterface *Comparator*<*T*> erfüllt.
- **foreach** - Macht etwas für jedes Element ohne eine neue Liste zu erstellen. Foreach erwartet einen Parameter der das Funktionsinterface *Consumer*<*T*> erfüllt.

Laden Sie sich als erstes den gegebenen Quellcode von der Übungswebseite herunter. Sie müssen nur an den gekennzeichneten Stellen den Quellcode ändern. Gehen Sie wie folgt vor:

(a) Implementieren Sie die Methode *justPrintIt*. Die Methode soll ein Objekt zurück geben welches das Funktionsinterface *Consumer*<*T*> zurück geben. Wenn Sie die Funktion geschrieben haben, können Sie am Ende der *main*-Methode die erste und letzte auskommentierte Zeile einkommentieren. Führen Sie das Programm aus und beachten Sie die Konsolenausgabe.

(b) Weisen Sie nun der Variablen *filter* mit Hilfe einer anonymen Klasse des Types *Predicate*<*T*> einen Wert zu, so dass alle Postleitzahlen die kleiner sind als die

Variable *threshold* in der Liste bleiben. Kommentieren Sie die entsprechende Zeile am Ende der *main*-Methode ein.

(c) Definieren Sie nun eine Variable *comparator* vom Typ $Comparator<T>$, so dass die Liste absteigend nach Ortsnamen sortiert wird. Kommentieren Sie die entsprechende Zeile am Ende der *main*-Methode ein.

(d) Kommentieren Sie die letzte Zeile am Ende der *main*-Methode ein und schreiben Sie ein Lambda, welches für jedes *PLZOrt* Objekt ein neues Objekt mit dem selben Ort erstellt, die Postleitzahl jedoch um eins erhöht.