Übung 04: Generics und Lambdas

Abgabetermin: 30.3.2017, 8:15

Name: _____ **Matrikelnummer**: _____

Informatik: ☐ G1 (Marr) ☐ G2 (Prähofer) ☐ G3 (Prähofer) ☐ G4 (Löberbauer)

WIN: ☐ G1 (Khalil) ☐ G2 (Hummel) ☐ G3 (Khalil)

Aufgabe	Punkte	abzugeben schriftlich	abzugeben elektronisch	korr.	Punkte
Übung 4	24	Java-Programm, Ausgabe des Testläufs	Java-Programm	☐	

Übung 4: SortedSet

In dieser Übung soll eine generische Klasse `SortedSet<T>` für eine sortierte Menge von beliebigen Elementen implementiert werden. `SortedSet` soll `Iterable<T>` implementieren. Der Einfachheit halber, wird `SortedSet` mit einer einfach verketteten Liste implementiert.

Des Weiteren sind höhere Funktionen `map`, `filter` und `reduce` zu implementieren, die die funktionalen Interfaces aus dem Package `java.util.function` (siehe Anhang) verwenden.

Aufgabe 4.a) Implementierung von `add`, `remove` und `itertor` (12 Punkte)

Implementieren Sie eine Klasse `SortedSet<T>` im Package `set` mit der folgenden Schnittstelle:

```
public class SortedSet<T> implements Iterable<T> {
  ...

  /** Constructor with comparator for comparing elements. */
  public SortedSet(Comparator<T> comparator) { ... }

  /** Adds an element elem to the set so that elements are sorted.
   * @return true, if element added to set, false if it was already in the set
   */
  public boolean add(T elem) { ... }

  /** Removes an object elem from the set.
   * @return true if element was removed from set, false otherwise
   */
  public boolean remove(Object elem) { ... }

  /** Returns an iterator for iterating over elements in set.
   * @return an iterator for iterating over elements
   */
  public Iterator<T> iterator() { ... }
}
```

Die Sortierung der Elemente soll mittels einen `java.util.Comparator<T>` mit Methode `int compare(T a, T b)` erfolgen. Die Methoden `compare` vergleicht zwei Werte `a` und `b` und liefert ein int-Wert wie folgt: ist `a` gleich `b` ist das Ergebnis 0, ist `a` kleiner als `b` ist das Ergebnis negativ und ist `a` größer als `b` ist das Ergebnis positiv.

Verwenden Sie innere Klassen für die Knoten und für die Implementierung des Iterators.

Test:

Zum Testen des `SortedSet` implementieren Sie eine Klasse `Numbers` in der Package `set.apps`, mit einer `main`-Methode, die erst ein `SortedSet<Integer>` erstellt und dann die folgenden Zahlen in der folgenden Reihenfolge zum SortedSet hinzufügt: 42, 1, 12, 200, 12, 12, 30. Zum Vergleichen der Elemente verwenden sie einen `Comparator<Interger>`, welche die Differenz der zwei Zahlen `a` und `b` zurückgibt (`a - b`).

Nach dem Hinzufügen der Zahlen, geben Sie alle Elemente des Sets aus. Verwenden Sie die `forEach`-Methode. Die Test-Ausgabe sollte wie folgt aussehen:

```
Elements
1
12
30
42
200
```

Anschließend entfernen Sie die drei Elemente 1, 200, 30 in der angegebenen Reihenfolge und geben Sie wieder das Set aus. Die Testausgabe zeigt die restlichen Elemente:

```
Remaining Elements
12
42
```

Aufgabe 4.b) Implementierung von `filter`, `map` und `reduce` (12 Punkte)

Implementieren Sie nun folgende generischen höheren Funktionen:

```
public SortedSet<T> filter(Predicate<T> test)
```

Die Methode `filter` gibt ein neues `SortedSet` zurück, das nur die Elemente enthält, die das gegebenen Prädikat erfüllen.

```
public <U> SortedSet<U> map(Function<T, U> fn, Comparator<U> comparator)
```

Die Methode `map` bildet die Elemente des Sets über die Funktion `fn` ab, d.h., die Funktion `fn` liefert für jedes Element des ursprünglichen Sets vom Typ `T` einen Wert von Typ `U` und diese Elemente werden in einen neuen Set eingefügt. Die Sortierung im neuen Set wird durch den im zweiten Parameter übergebenen `Comparator` bestimmt wird. Beachten Sie, dass das Resultat eine `SortedSet` mit Typ `U` ist und `U` ein neuer Typparameter ist.

```
public <U> U reduce(U initial, BiFunction<U, T, U> reducer)
```

Die Methode `reduce` berechnet aus den Elementen des Sets einen einzelnen Wert vom Typ `U`. Basierend auf dem Initialwert (`initial`) wird durch eine Funktion `reducer` jedes Element des Sets mit dem bereits berechneten Teilergebnis kombiniert. Das heißt, das erste Teilergebnis ist gleich dem Wert `initial`

```
U result = initial
```

Das erste Element `e1` wird mit diesem ersten Teilergebnis kombiniert

```
result = reducer.apply(result, e1)
```

das zweite Element `e2` mit diesem Teilergebnis

```
result = reducer.apply(result, e2)
```

usw.

Testen der Implementierung

Zum Testen der Implementierung schreiben Sie eine neue `Students` mit `main`-Methode im `set.apps` Package. Diese Anwendung soll eine Klasse `Student` mit einem Feld für Namen und Alter verwenden.

Erstellen Sie dann ein neues `SortedSet<Student>`. Studenten sollen nach dem Namen sortiert werden. Fügen Sie mindestens 4 Studenten dem Set hinzu, mindestens 2 davon sollen älter als 20 sein. Geben Sie alle Studenten mit Namen und Alter aus.

Verwenden Sie nun die `filter`-Methode, um alle Studenten, die älter als 20 sind, zu ermitteln und geben Sie diese aus.

Verwenden Sie die `reduce`-Methode, um das Gesamtalter der Studenten zu ermitteln (= Summe der einzelnen Altersangaben der Studenten) und geben Sie das Ergebnis aus.

Verwenden Sie dann die `reduce`-Methode, um das Alter des ältesten Studenten zu ermitteln und geben Sie das Ergebnis aus.

Zum Abschluss verwenden Sie die `map`-Methode, um vom originalen Set der Studenten ein Set der Namen zu erstellen. Sortieren Sie die Namen alphabetisch.

Die Test-Ausgabe soll in etwas wie folgt aussehen:

```
Students:
A 22
B 18
C 19
D 25

Old Students:
A 22
D 25

Sum student age: 84

Max Student age: 25

Names:
A
B
C
D
```

Anhang: Wichtige Interfaces

Aus Package java.util.function

```
@FunctionalInterface
public interface Function<T,R> {
       R apply(T t);
       ...
}

@FunctionalInterface
public interface Predicate<T> {
    boolean test(T t);
    ...
}

@FunctionalInterface
public interface Consumer<T> {
       void   accept(T t);
       ...
}

@FunctionalInterface
public interface Supplier<T> {
       T get();
}

@FunctionalInterface
public interface BiFunction<T,U,R> {
       R apply(T t, U u);
       ...
}
```

Aus Package java.util

```
@FunctionalInterface
public interface Comparator<T> {
       int compare(T o1, T o2);
       ...
}

public interface Iterable<T> {
    Iterator<T> iterator();

    default void forEach(Consumer<? super T> action) {
        ...
        for (T t : this) {
            action.accept(t);
        }
    }
    ...
}
```