Übungen zu Objektorientierte Programmierung 2

frieder.loch@ost.ch

Woche 1: Generische Programmierung

Aufgabe 1: Generische Paare

In dieser Aufgabe sollen Sie einen generischen Datentyp `Pair` implementieren. Ein Pair repräsentiert eine Einheit von zwei Elementen (`first` und `second`) mit möglicherweise verschiedenen Typen, z.B. Paar von Strasse mit Hausnummer, Studierender mit Note, usw.

Das Interface von Pair (ohne Generics) sieht wie folgt aus:

```
interface Pair {
  Object getFirst();
  Object getSecond();
}
```

Passen Sie zunächst das Interface so an, dass es Generics verwendet.

Erstellen Sie anschliessend zwei konkrete Implementationen mit unterschiedlicher Gleichheitssemantik (`equals`). Die Elemente `first` und `second` können grundsätzlich unterschiedliche Typen haben, dürfen jedoch nicht null sein.

- `UnorderedPair`: Paare sind gleich, wenn Sie die gleichen Elemente haben, egal ob diese in `first` oder `second` gespeichert sind. `first` und `second` haben hier den gleichen Typ.
- `OrderedPair`: Paare sind gleich, wenn die beiden `first`-Elemente sowie die beiden `second`-Elemente gleich sind. `first` und `second` können verschiedene Typen haben.

Aufgabe 2: Reverse Map

In der Java-API fehlt eine nützliche Map-Datenstruktur, nämlich die Reverse Map: Hier werden Paare A und B in die Map eingetragen, wobei eine eindeutige Abbildung zwischen A und B in beide Richtungen entsteht (eine sog. Bijektion). Man kann dann sowohl A als auch B als Schlüssel für die Suche verwenden, um das Gegenstück des Paars zu finden.

Anwendungsbeispiel:

```
ReverseMap<Integer, String> map = new ReverseMap<Integer, String>();
map.put(123456, "Hans Meier");
map.put(333999, "Clara Müller");
String name = map.getRight(123456);          // returns "Hans Meier"
int number = map.getLeft("Clara Müller");    // returns 333999
```

Implementieren Sie die Klasse `ReverseMap<L, R>` unter Verwendung von Generics. Die Klasse verfügt über folgende Methoden:

Methode	Beschreibung
`void put(L left, R right)`	Hinzufügen des Paars (left, right). Beachten Sie, dass sowohl left als auch right noch nicht enthalten sein dürfen.

Methode	Beschreibung
`R getRight(L left) L getLeft(R right)`	Finde die Paar-Zuordnung für das gegebene linke bzw. rechte Paar-Element.
`Collection<L> leftValues()`	Collection aller linker bzw. rechter Paar-Elemente
`Collection<R> rightValues()`	
`int size()`	Anzahl der eingetragenen Paare
`void clear()`	Löscht alle Einträge

Benutzen Sie den Testcode der Vorlage.

Aufgabe 3: Merge and Sort

Implementieren Sie eine generische Methode namens mergeAndSort, die zwei Collections des gleichen Typs zusammenführt und sortiert. Die Methode soll Teil einer Utility-Klasse namens CollectionUtils sein. Für den Sortiervorgang soll die Methode Comparable::compareTo genutzt werden. Das Ergebnis soll eine sortierte Liste desselben Typs sein.

Anwendungsbeispiel:

```
List<Character> list1 = List.of('a', 'c', 'd');
List<Character> list2 = List.of('b', 'z', 'f');
var combinedList =  CollectionUtils.mergeAndSort(list2, list1); // returns [a, b, c, d, f, z]
```

Teste deine Implementation mit der Test Klasse CollectionUtilsTest.