Übungsblatt 06

3 Aufgaben, 20 Punkte

Objektorientierte Modellierung und Programmierung (inf031) Sommersemester 2020
Carl von Ossietzky Universität Oldenburg, Fakultät II, Department für Informatik

Dr. C. Schönberg

Ausgabe: 2020-05-26 14:00 **Abgabe:** 2020-06-02 12:00

Aufgabe 1: *Multimengen* **(1 + 2 + 5 Punkte)**

Für eine Grundmenge A heißt die Abbildung $M : A \to \mathbb{N}$ eine *Multimenge* über A. Im Gegensatz zu einer gewöhnlichen Teilmenge $T \subseteq A$ erlaubt eine Multimenge nicht nur die Darstellung, ob ein Element $a \in A$ in der Menge liegt oder nicht, sondern es wird zusätzlich die Vielfachheit eines Elements repräsentiert.

Beispiel: Sei A die Menge aller Kleinbuchstaben. Im mengentheoretischen Sinne sind die Teilmengen $\{a, a, b, c\}$ und $\{a, b, c\}$ gleich, während sie als Multimengen verschieden sind:
$\{(a, 2), (b, 1), (c, 1), (d, 0), (e, 0), \ldots\} \neq \{(a, 1), (b, 1), (c, 1), (d, 0), (e, 0), \ldots\}$.

Der Einfachheit halber notiert man bei Multimengen oft nur die Elemente mit Vielfachheit > 0, also etwa $\{(a, 2), (b, 1), (c, 1)\}$.

a) Notieren Sie die in dem Wort REGENWETTER enthaltenen Zeichen als Menge und als Multimenge.

b) Definieren Sie eine *generische* Java-*Schnittstelle* `MultiSet`, die das aus der Vorlesung bekannte `Iterable`-Interface erweitert. Beim Iterieren sollen die Elemente der Multimenge zurückgegeben werden, nicht deren Anzahl (also z.B. “a” statt “2”).
Außerdem soll die Schnittstelle eine Methode `void add(T element)` zum Hinzufügen von Elementen bereitstellen und eine Methode `int count(T element)`, welche zu einem Element der Menge seine Anzahl zurückgibt.

c) Schreiben Sie eine Java-Klasse `HashMapMultiSet`, die unter Verwendung der aus der Vorlesung bekannten Klasse `HashMap` das Interface `MultiSet` implementiert.

Aufgabe 2: *Generics und Exceptions* **(2 Punkte)**

In Java ist es nicht erlaubt, Exceptions generisch zu parametrisieren:

```
1 class GenericException<T> extends Exception { }
```

lässt sich nicht kompilieren und führt zu der Fehlermeldung "*The generic class `GenericException<T>` may not subclass `java.lang.Throwable`*".

Warum ist das so?

Hinweis: Betrachten Sie dazu folgende zwei Beispiele und überlegen Sie dabei, wie Java zur Laufzeit (!) mit generischen Typen umgeht:

```
1 try {
2   // ...
3 } catch (GenericException<Integer> e) {
4   // ...
5 } catch (GenericException<Double> e) {
6   // ...
7 }
```

```
1 try {
2   throw new GenericException<Number>();
3 } catch (GenericException<Integer> e) {
4   // ...
5 }
```

Aufgabe 3: *JDK* **(10 Punkte)**

In dieser Aufgabe sollen Sie den Umgang mit dem JDK üben.
Gegeben sei folgendes Java-Interface und eine Test-Klasse:

```
1  import java.util.Collection;
2
3  public interface Company {
4
5    void addEmployee(int id, String name) throws DuplicateIdException;
6    String getEmployeeName(int id);
7    void addProject(int id, String name) throws DuplicateIdException;
8    String getProjectName(int id);
9    void assignEmployeeToProject(int employeeId, int projectId) throws
         UnknownIdException;
10   void removeEmployeeFromProject(int employeeId, int projectId) throws
         UnknownIdException;
11   Collection<Integer> getEmployees();
12   Collection<Integer> getProjectsForEmployee(int employeeId) throws
         UnknownIdException;
13
14 }
```

```
1  public class StarkTest {
2    public static void main(String[] args) {
3      Company stark = new StarkEnterprises();
4      try {
5        stark.addEmployee(0, ”Tony”);
6        stark.addEmployee(1, ”Pepper”);
7        stark.addEmployee(2, ”Jarvis”);
8        stark.addProject(0, ”Suit”);
9        stark.addProject(1, ”Jarvis”);
10       stark.addProject(2, ”Jarvis”); // same name, different id
11       stark.addProject(3, ”Finances”);
12       stark.assignEmployeeToProject(0, 0);
13       stark.assignEmployeeToProject(0, 1);
14       stark.assignEmployeeToProject(1, 3);
15       stark.assignEmployeeToProject(2, 0);
16       stark.assignEmployeeToProject(2, 2);
17       System.out.println(stark);
18     } catch (InvalidIdException e) {
19       System.out.println(”Invalid ID: ” + e.getId());
20     }
21   }
22 }
```

Implementieren Sie eine Java-Klasse `StarkEnterprises`, welche bei der Ausführung der Test-Klasse folgende Ausgabe erzeugt:

```
Jarvis[2]: Jarvis[2] Suit[0]
Pepper[1]: Finances[3]
Tony[0]: Jarvis[1] Suit[0]
```

Sie dürfen dazu drei Exceptions und zwei Comparator-Klassen implementieren, dürfen sonst aber *ausschließlich* die aus der Vorlesung bekannten Klassen und Interfaces des JDK verwenden. Insbesondere dürfen Sie *keine* Klassen `Person` oder `Project` anlegen.
Denken Sie daran, die `toString`-Methode zu überschreiben, um die Ausgabe der Mitarbeiter (alphabetisch sortiert) mit ihren Projekten (ebenfalls alphabetisch sortiert) zu ermöglichen. Geben Sie bei den Mit-

arbeitern und Projekten auch immer die ID in [] mit an, damit Einträge mit gleichem Namen unterschieden werden können.

Die Methoden des Interfaces sollen folgendes Verhalten haben:

- `addEmployee`: Fügt einen Mitarbeiter (m/w/d) mit eindeutiger ID hinzu. Wirft eine Exception, wenn die ID bereits verwendet wird.
- `getEmployeeName`: Gibt den Namen eines Mitarbeiters zurück, oder **`null`** falls die ID unbekannt ist.
- `addProject`: Fügt ein Projekt mit eindeutiger ID hinzu. Wirft eine Exception, wenn die ID bereits verwendet wird.
- `getProjectName`: Gibt den Namen eines Projektes zurück, oder **`null`** falls die ID unbekannt ist.
- `assignEmployeeToProject`: Fügt einen Mitarbeiter einem Projekt hinzu. Ein Mitarbeiter kann an mehreren Projekten beteiligt sein, aber an jedem Projekt höchstens einmal (doppelte Zuweisungen werden ignoriert). Wirft eine Exception, wenn eine der IDs unbekannt ist.
- `removeEmployeeFromProject`: Entfernt einen Mitarbeiter von einem Projekt. Wirft eine Exception, wenn eine der IDs unbekannt ist. Hat keinen Effekt, wenn der Mitarbeiter nicht dem Projekt zugewiesen war.
- `getEmployees`: Gibt eine geordnete Sammlung aller Mitarbeiter-IDs zurück. Die Sammlung ist nach den Namen der Mitarbeiter sortiert.
- `getProjectsForEmployee`: Gibt eine geordnete Sammlung aller Projekt-IDs eines Mitarbeiters zurück. Die Sammlung ist nach den Namen der Projekte sortiert.

Hinweis 1: Einer `Comparator`-Implementierung können Sie (z.B. im Konstruktor) zusätzliche Informationen übergeben, die für einen Vergleich nötig sind.
Hinweis 2: Die Klasse `String` implementiert das `Comparable<String>`-Interface.
Hinweis 3: Beachten Sie beim Sortieren, dass nur die IDs, nicht aber die Namen eindeutig sind.