Philipps-Universität Marburg
Fachbereich Mathematik und Informatik
Arbeitsgruppe Programmiersprachen und -werkzeuge

Wintersemester 2019/2020
22. Januar 2020

Prof. Dr. Christoph Bockisch
Stefan Schulz

Übungen zur Vorlesung
Objektorientierte Programmierung

Abgabe: 29. Januar 2020,
bis **spätestens** 12:00 Uhr
über ILIAS

Übung 10

Aufgabe U-10.1: Comparable (2 + 3 + 2 = 7 Punkte)

In dieser Aufgabe sollen Sie Ihre Kartensammlung für das fiktive Sammelkartenspiel *Abracadabra: The Union* sortieren. Jede der Karten hat einen Namen und ist einer von fünf verschiedenen Farben (Weiß, Blau, Schwarz, Rot, Grün) zugeordnet. Zusätzlich ist das Erscheinungsjahr aufgedruckt. Ihre Aufgabe ist es nun, die Karten zuerst nach Erscheinungsjahr, Farbe und schlussendlich nach Namen zu sortieren.

a) Erstellen Sie die Klasse `Card`. Diese soll über einen Namen, eine Farbe, sowie das Erscheinungsjahr verfügen. Modellieren Sie die verschiedenen Farben als finale, statische Felder des Typs `String` in `Card`[1]. Implementieren Sie außerdem entsprechende Getter-Methoden.

b) Implementieren Sie nun das Interface `Comparable` in `Card` mit der folgenden Ordnung:

- Die Karten werden zuerst aufsteigend nach ihrem Erscheinungsjahr sortiert.
- Bei gleichem Erscheinungsjahr sollen die Karten nach ihrer Farbe sortiert werden:
  Weiß > Blau > Schwarz > Rot > Grün
- Bei gleichem Erscheinungsjahr und gleicher Farbe sollen die Karten lexikographisch nach ihrem Namen sortiert werden.

Hinweis: Die Klasse `String` implementiert ebenfalls die `Comparable`-Schnittstelle.

c) Testen Sie Ihre Implementierung in einem JUnit-Test, indem Sie ein Array von mindestens 10 Karten sortieren[2], wovon mindestens fünf einzigartig (Name und Farbe) sein sollen. Beachten Sie, dass Ihre Sammlung auch mehrere Kopien einer Karte enthalten kann.

[1] Somit können Sie Instanzen von `Card` beispielsweise wie folgt erzeugen: `new Card(<name>, <erscheinungsjahr>, Card.WHITE)`. Vorausgesetzt das statische Feld für die Farbe Weiß heißt `WHITE`.

[2] Sie können das Array mit der Anweisung `Arrays.sort(<Arrayname>)` sortieren.

Aufgabe U-10.2: Lambdas **(2 + 2 = 4 Punkte)**

Erstellen Sie die Klasse `Lambdas` mit einer `main`-Methode. Erzeugen Sie in dieser Methode eine `String`-Liste mit mindestens 10 Strings unterschiedlicher Länge und maximal einem Eintrag, der der Anfang eines anderen ist (z.B. „a“und „ab“).

a) Machen Sie sich mit dem Interface `java.util.Comparator<T>` vertraut[3]. Nutzen Sie die `sort`-Methode in `List`, um die Liste zu sortieren:
   - Aufsteigend nach der Länge der Elemente.
   - Absteigend nach dem Anfangsbuchstaben der Elemente. Dabei soll Groß- und Kleinschreibung ignoriert werden.

   Geben Sie die sortierten Listen jeweils auf der Konsole aus.

b) Machen Sie sich mit dem Interface `java.util.function.Consumer<T>` vertraut[4]. Nutzen Sie die `foreach`-Methode[5] von `List`, um für jedes Element der Liste den umgekehrten String auf der Konsole auszugeben. Beispiel: {"ABC", "DEF"} → Ausgabe: "CBA", "FED"

Übergeben Sie die benötigten `Comparator`- und `Consumer`-Argumente ausschließlich als Lambda-Ausdrücke.

Aufgabe U-10.3: Map **(1 + 1 + 5 + 2 = 9 Punkte)**

In dieser Aufgabe sollen Sie die Datenstruktur `ListMap` implementieren. Die Idee dieser Datenstruktur ist es, Elemente als Paar von Schlüsseln und Werten zu speichern. Das heißt also, dass jedem Schlüssel durch die Map ein Wert zugeordnet werden kann. Dabei darf jeder Schlüssel nur einmal, Werte aber beliebig oft vorkommen.

Beispiel:

Matrikelnummer	Name
36353741	John Doe
36912359	Jane Doe
37000001	Alice Liddell
37000002	Mallory McMallard
37000003	Bob Ross
38372558	John Doe

Eine `ListMap<K, V>` mit Schlüsseln vom Typ `K` und Werten vom Typ `V` verfügt über die folgenden Methoden:

- `void put(K key, V value)`, fügt ein Schlüssel-Wert Paar in die `Map` ein. Sollte für den übergebenen Schlüssel schon ein Wert hinterlegt sein, wird dieser ersetzt.
- `V get(K key)`, gibt den zum übergebenen Schlüssel abgelegten Wert zurück. Sollte der Schlüssel nicht in der `Map` vorhanden sein, wird stattdessen `null` zurückgegeben.
- `void remove(K key)` entfernt den Eintrag mit dem Schlüssel `key`.

a) Erstellen Sie die generische Klasse `MapEntry<K, V>`, die ein Schlüssel-Wert-Paar beliebiger Datentypen speichern kann. Speichern Sie Schlüssel und Wert in privaten Feldern und legen Sie Getter- und Setter-Methoden für beide an. Überschreiben Sie die `toString()`-Methode so, dass das Paar leserlich auf der Konsole ausgegeben werden kann.

[3] Siehe `https://docs.oracle.com/javase/10/docs/api/java/util/Comparator.html`
[4] Siehe `https://docs.oracle.com/javase/10/docs/api/java/util/function/Consumer.html`
[5] Siehe `https://docs.oracle.com/javase/10/docs/api/java/lang/Iterable.html#forEach`.

b) Erstellen Sie die generische Klasse `ListMap<K, V>`. Eine `ListMap` verwaltet die Einträge als Liste `java.util.List` von Schlüssel-Wert-Paaren (`MapEntry<K, V>`). Erstellen Sie ein entsprechendes Feld in der Klasse `ListMap` und initialisieren Sie es geeignet.

c) Implementieren Sie nun die oben beschriebenen Methoden `put`, `get` und `remove`.

d) Testen Sie Ihre Implementierung mithilfe von JUnit-Tests:

- `put`: Fügen Sie mindestens vier Schlüssel-Wert-Paare zur `Map` hinzu, zwei der Paare sollen den gleichen Schlüssel haben, die anderen zwei den gleichen Wert.
- `get`: Fragen Sie die Werte zu jeweils mindestens einem Schlüssel der in der Liste vorhanden ist und einem Schlüssel der nicht in der Liste vorhanden ist ab.
- `remove`: Entfernen Sie mindestens ein vorhandenes Element aus der Liste. Rufen Sie `remove` außerdem für einen Schlüssel auf, der nicht in der Map enthalten ist.