Programmierkurs Java

Dr.-Ing. Dietrich Boles

Aufgaben zu UE 23 – Generics

(Stand 16.04.2015)

Aufgabe 1:

In einem Programm soll ein abstrakter Datentyp *Menge* verwendet werden. Dieser soll es ermöglichen, mit Mengen im mathematischen Sinne umzugehen. Eine Menge soll dabei eine beliebig große Anzahl von Werten aufnehmen können, jeden Wert aber maximal einmal.

Schreiben sie eine generische Klasse *Menge*, welche diesen ADT implementiert.

Auf dem Datentypen Menge sollen folgende Funktionen möglich sein:

- Erzeugen einer neuen leeren Menge.
- Überprüfen auf Gleichheit zweier Mengen
- Hinzufügen eines Wertes zu einer Menge.
- Entfernen eines Wertes aus einer Menge.
- Überprüfung, ob ein bestimmter Wert in der Menge enthalten ist.
- Schnittmengenbildung zweier Mengen.
- Vereinigung zweier Mengen.
- Differenzbildung zweier Mengen.

Schreiben Sie weiterhin ein kleines Testprogramm für die Klasse *Menge*.

Aufgabe 2:

Implementieren Sie selbst eine generische Klasse *ArrayList*.

Aufgabe 3:

Implementieren Sie selbst eine generische Klasse *Stack*.

Aufgabe 4:

Teilaufgabe (a): Implementieren Sie eine Klasse `Datum`, die Jahre, Monate und Tage verwalten kann. Lassen Sie die Klasse `Datum` das Interface `java.lang.Comparable` implementieren. Schreiben Sie ein kleines

Testprogramm, bei dem Objekte Ihrer Klasse Datum erzeugt und in einer `java.util.ArrayList` abgespeichert werden.

Teilaufgabe (b): Nutzen Sie dann die Methode `Collections.sort` zum Sortieren der ArrayList.

Teilaufgabe (c): Definieren Sie selbst folgende Klasse zum Sortieren und nutzen Sie sie zum Sortieren Ihrer ArrayList (implementieren Sie bspw. den Bubblesort-Algorithmus):

```
class Sorter<T extends Comparable<T>> {
     public void sort(ArrayList<T> list) {
     }
}
```

Vermeiden Sie durchgängig falls möglich die Verwendung von raw-types!

Aufgabe 5:

Teilaufgabe (a):

Implementieren Sie eine ADT-Klasse `Intervall`. Diese soll den Umgang mit mathematischen Intervallen ermöglichen. Ein Intervall im Sinne dieser Aufgabe besteht dabei aus einer Untergrenze (`int`) und einer Obergrenze (`int`). Es beinhaltet damit implizit alle `int`-Zahlen zwischen der Untergrenze inklusive und der Obergrenze inklusive. Ist die Untergrenze größer als die Obergrenze ist das Intervall leer.

Die Klasse `Intervall` soll folgende Methoden bereitstellen:

- Einen Default-Konstruktor (erzeugt ein leeres Intervall)
- Einen Konstruktor, dem die Untergrenze und die Obergrenze des Intervalls als Parameter übergeben werden
- Einen Copy-Konstruktor
- Eine Methode `clone` zum Klonieren eines Intervalls (Überschreiben der Methode `clone` der Klasse `Object`)
- Eine Methode `equals`, mit der zwei Intervalle auf Gleichheit überprüft werden können (Überschreiben der Methode `equals` der Klasse `Object`). Zwei Intervalle sind dabei gleich, wenn sie beide leer sind oder dieselben Zahlen beinhalten.
- Eine Methode `toString`, die eine String-Repräsentation eines Intervalls liefert (Überschreiben der Methode `toString` der Klasse `Object`). Die String-Repräsentation soll dabei folgende Gestalt haben: „`[]`“ für ein leeres Intervall und „`[u, o]`“ für nicht leere Intervalle, wobei *u* die Unter- und *o* die Obergrenze darstellen.

- Eine boolesche Methode `enthaelt`, die überprüft, ob das aufgerufene Intervall ein als Parameter übergebenen Intervall komplett enthält. Dabei gilt: Ein leeres Intervall ist in jedem Intervall enthalten. Ein leeres Intervall enthält nur leere Intervalle. Für nicht leere Intervalle *a* und *b* enthält *a b* genau wenn, wenn alle Zahlen von *b* auch in *a* enthalten sind.
- Eine statische Methode `schnittmenge`, die zwei Intervalle als Parameter übergeben bekommt und die Schnittmenge dieser Intervalle als neues Intervall-Objekt liefert. Dabei gilt: Die Schnittmenge eines leeren Intervalls mit einem anderen Intervall ist ein leeres Intervall. Die Schnittmenge zweier nicht leerer Intervalle ist ein Intervall, das alle Zahlen umfasst, die in beiden Intervallen enthalten sind.
- Getter-Methoden für alle Attribute.

Teilaufgabe (b):

Sie kennen die for-each-Schleife von Java. Wir haben sie benutzt, um durch ein Array zu iterieren:

```
int[] werte = {33, 2, 678};
int summe = 0;
for (int wert : werte) {
      summe += wert;
}
```

Die for-each-Schleife kann aber nicht nur für Arrays sondern für alle Objekte von Klassen genutzt werden, die das Interface `java.lang.Iterable` implementieren.

Die Klasse `java.util.ArrayList` implementiert beispielsweise das Interface `Iterable`, so dass durch folgende Funktion die Summe der `int`-Werte der als Parameter übergebenen ArrayList berechnet und zurückgegeben wird.

```
int getSumme(java.util.ArrayList<Integer> werte) {
      int summe = 0;
      for (int wert : werte) {
            summe += wert;
      }
      return summe;
}
```

Diese for-each-Schleife wird dabei durch den Compiler in folgendes Konstrukt überführt:

```
int getSumme(java.util.ArrayList<Integer> werte) {
      int summe = 0;
      java.util.Iterator<Integer> iter = werte.iterator();
      while (iter.hasNext()) {
            summe += iter.next();
      }
      return summe;
}
```

Aufgabe: Erweitern Sie die Klasse `Intervall` aus Teilaufgabe(a) derart, dass sie auf folgende Art und Weise in Zusammenhang mit der for-each-Schleife benutzt werden kann:

```
Intervall intervall = new Intervall(-3, 5);
for (int i : intervall) {
      System.out.println(i);
}
```

Die for-each-Schleife soll dabei alle Werte des Intervalls von der Untergrenze bis zur Obergrenze aufsteigend durchlaufen, in dem Beispiel also die Zahlen von -3 bis 5 auf den Bildschirm ausgeben.

Schauen Sie sich die Interfaces `java.lang.Iterable` und `java.util.Iterator` bitte in der Java-API-Dokumentation an. Die `Iterator`-Methode `remove` können Sie außer Acht lassen.

Aufgabe 6:

Ein Wörterbuch im Sinne der folgenden Aufgabe ist dadurch charakterisiert, dass es mehrere Begriffe enthält, denen jeweils ein oder mehrere Übersetzungen zugeordnet sind.

Aufgabe (a): Implementieren Sie eine Klasse `Woerterbuch` mit folgenden Methoden:

1. Eine Methode `einfuegen` mit zwei `String`-Parametern, nämlich einem Begriff und einer (u. U. weiteren) Übersetzung.
2. Eine Methode `getUebersetzungen`, die einen String als Parameter übergeben bekommt und als String-Array die im Wörterbuch gespeicherten Übersetzungen dieses Begriffs liefert.

Nutzen Sie zur Realisierung der internen Datenstruktur die Klasse `java.util.HashMap` und `java.util.ArrayList` Vermeiden Sie raw-types!

Aufgabe (b): Schreiben Sie mit Hilfe der Klasse `Woerterbuch` ein kleines Programm, in dem ein Benutzer zunächst eine Menge an Begriffen und Übersetzungen eingeben kann und er sich anschließend zu einzelnen einzugebenden Begriffen die gespeicherten Übersetzungen ausgeben lassen kann.

Aufgabe 7:

In nahezu jedem Lehrbuch finden Sie im hinteren Teil so genannte Indexe bzw. Sachwortverzeichnisse. Hier werden wichtige Begriffe, die im Buch vorkommen, sowie die jeweiligen Seitenzahlen, auf denen die Begriffe vorkommen, aufgelistet.

Implementieren Sie eine ADT-Klasse *Index*, die einen solchen Index repräsentiert. Die Klasse soll folgende Methoden zur Verfügung stellen:

- Einen Default-Konstruktor, der einen leeren Index erstellt.
- Eine Methode *hinzufuegen*, die einen Begriff (String) sowie eine Seitenzahl (int-Wert) als Parameter übergeben bekommt und die den Begriff mit der Seitenzahl im Index vermerkt. Dabei gilt: Begriffe dürfen nicht mehrfach im Index vorkommen und zu jedem Begriff darf eine Seitenzahl höchstens einmal im Index vorkommen.

- Eine Methode *toString*, die eine String-Repräsentation des aktuellen Index liefert (Überschreiben der von der Klasse *Object* geerbten Methode *toString*). Die String-Repräsentation soll dabei folgendermaßen aufgebaut sein: Für jeden im Index vermerkten Begriff soll der String jeweils eine Zeile der folgenden Form enthalten: <begriff>: <seitenzahl 1> <seitenzahl 2> … <seitenzahl n> (also bspw.: Hamster: 2 45 123)

Hinweis:

Eine sortierte Ausgabe ist nicht notwendig! Wählen Sie geeignete JDK-Klassen! Vermeiden Sie raw-types!

Testen:
Ob Ihre Klasse (zumindest teilweise) korrekt ist, können Sie testen, indem Sie das folgende Testprogramm ausführen:

```
public class IndexTest {

     public static void main(String[] args) {
           Index index = new Index();

          index.hinzufuegen("Objekt", 1);
           index.hinzufuegen("Objekt", 15);
           index.hinzufuegen("Objekt", 3);
           index.hinzufuegen("Objekt", 15);
           index.hinzufuegen("Objekt", 17);

           index.hinzufuegen("Hamster", 45);
           index.hinzufuegen("Hamster", 2);
           index.hinzufuegen("Hamster", 199);
           index.hinzufuegen("Hamster", 45);

           index.hinzufuegen("Java", 3);
           index.hinzufuegen("Java", 2);
           index.hinzufuegen("Java", 3);
           index.hinzufuegen("Java", 3);

           System.out.println(index);
     }

}
```

Es sollte folgende Ausgabe (unsortiert) erscheinen:

```
Objekt: 17 1 3 15
Hamster: 2 199 45
Java: 2 3
```

Aufgabe 8:

Implementieren Sie die statische generische Methode `shuffle` („Mischen“) der folgenden Klasse `Shuffler`. Der Aufruf der Methode soll die übergebene Collection zufallsmäßig durchmischen.

```
public class Shuffler {
     public static <T> void shuffle(Collection<T> coll)
}
```

Aufgabe 9:

Gegeben seien die folgenden Interfaces bzw. Klassen:

```
interface Predicate<T> {
     public boolean call(T obj);
}

interface Projection<T, RT> {
     public RT call(T obj);
}

public class QueryArrayList<T> extends java.util.ArrayList<T> {

     QueryArrayList<T> where(Predicate<T> pre) {
           QueryArrayList<T> newList = new QueryArrayList<T>();
           for (T obj : this) {
                 if (pre.call(obj)) {
                       newList.add(obj);
                 }
           }
           return newList;
     }

     <R> QueryArrayList<R> project(Projection<T, R> sel) {
           QueryArrayList<R> newList = new QueryArrayList<R>();
           for (T obj : this) {
                 newList.add(sel.call(obj));
           }
           return newList;
     }
}
```

`Predicate` und `Projection` sind zwei Interfaces, die das Überprüfen von Eigenschaften von Objekten bzw. das Überführen von Objekten in eine andere Form erlauben. Die Klasse `QueryArrayList` erweitert die Klasse `ArrayList` um zwei Methoden:

- Über die Methode `where` werden alle aktuell gespeicherten Objekte auf eine bestimmte übergebene Eigenschaft überprüft. Alle entsprechend gültigen Objekte werden dann in einer neuen Liste zurückgeliefert.
- Über die Methode `project` werden alle aktuell gespeicherten Objekte gemäß einer übergebenen Projektionsregel in eine andere Form überführt und dann in einer neuen Liste zurückgeliefert.

Diese gegebenen Interfaces bzw. Klassen sollen Sie nun folgendermaßen einsetzen:

Gegeben sei die folgende Klasse Person:

```
class Person {
     public String name; // Nachname
     public String firstname; // Vorname
     public boolean isMale; // maennlich?

     public Person(String name, String firstname, boolean isMale) {
           this.name = name;
           this.firstname = firstname;
           this.isMale = isMale;
     }
}
```

Schreiben Sie ein Programm, in dem zunächst `Person`-Objekte durch Benutzereingaben über die Console erzeugt und in einer `QueryArrayList` abgespeichert werden. Anschließend sollen durch Einsatz der Methoden `where` und `project` die Nachnamen der männlichen Personen in der Liste ermittelt und anschließend auf die Console ausgegeben werden.

Beispiel:

Im Folgenden wird ein möglicher Programmablauf skizziert (Benutzereingaben in <>):

```
Personenerzeugung:
Weitere Person (j/n)?<j>
Name: <Mueller>
Vorname: <Otto>
maennlich (true/false)?<true>
Weitere Person (j/n)?<j>
Name: <Meier>
Vorname: <Karin>
maennlich (true/false)?<false>
Weitere Person (j/n)?<j>
Name: <Schulze>
Vorname: <Karl>
maennlich (true/false)?<true>
Weitere Person (j/n)?<n>

Nachnamen:
Mueller
Schulze
```

Aufgabe 10:

Ausgangslage sind die folgenden Klassen:

```
abstract class Spieler {}

class FussballSpieler {
  FussballSpieler(String name)
  String geName()
  void schiessTor()
  int anzahlGeschosseneTore()
}

class BasketballSpieler {
  BasketballSpieler(String name)
  String getName()
  void wirfKorb()
  int erzieltePunkte()
}


class Mannschaft {
  void aufnehmen(Spieler spieler)
  void rausschmeissen(Spieler spieler)
  void auswechseln(Spieler alt, Spieler neu)
}
```

Teilaufgaben:

1. Implementieren Sie die obigen Klassen.

2. Ersetzen Sie die Klasse `Mannschaft` durch eine generische Klasse.
3. Erlauben Sie nur die Aufnahme von Spieler-Objekten in eine Mannschaft.
4. Definieren Sie eine generische Klasse `Liga` zur Aufnahme von Mannschaften eines bestimmten Typs.
5. Definieren Sie eine generische Klasse `WildeLiga`, in der beliebige Spieler spielen dürfen.
6. Definieren Sie eine Klasse `SaisonEnde` mit einer generischen Methode, die implementiert, dass die letzte Mannschaft einer Liga in eine andere Liga absteigt und die erste Mannschaft dieser Liga in die andere Liga aufsteigt.

Aufgabe 11:

In dieser Aufgabe geht es um die Ausgabe von Sporttabellen der folgenden Form

Verein	Spiele	S	U	N	Torverh.	Diff.	Pkt.
FC Bayern München	23	19	3	1	63:8	+55	60
Borussia Dortmund	23	12	7	4	51:27	+24	43
Bayer 04 Leverkusen	23	12	6	5	43:30	+13	42
Eintracht Frankfurt	23	11	5	7	38:34	+4	38
SC Freiburg	23	9	8	6	29:22	+7	35
Hamburger SV	23	10	4	9	28:32	-4	34
Hannover 96	23	10	3	10	46:42	+4	33
1. FSV Mainz 05	23	9	6	8	31:28	+3	33
FC Schalke 04	23	9	6	8	37:38	-1	33
Borussia M'Gladbach	23	7	10	6	32:34	-2	31
VfB Stuttgart	23	8	5	10	25:40	-15	29
SV Werder Bremen	23	8	4	11	39:47	-8	28
Fortuna Düsseldorf	23	7	6	10	28:31	-3	27
1. FC Nürnberg	23	6	9	8	23:30	-7	27
VfL Wolfsburg	23	7	6	10	23:33	-10	27
FC Augsburg	23	3	9	11	20:36	-16	18
1899 Hoffenheim	23	4	4	15	27:48	-21	16
SpVgg Greuther Fürth	23	2	7	14	13:36	-23	13

Jede Zeile beinhaltet dabei folgende Informationen:

- Name des Vereins
- Anzahl Spiele
- Anzahl Siege (S)
- Anzahl Unentschieden (U)
- Anzahl Niederlagen (N)
- Torverhältnis (Geschossen (Plus) : Reinbekommen (Minus))
- Tordifferenz (Diff.)
- Pluspunkte (Pkt.)

Die der Ausgabe zugrunde liegende Datenstruktur ist dabei wie folgt vorgegeben:

```
class Liga {
     private Verein[] vereine;

     public Verein[] getVereine() {
           return vereine;
     }
}


class Verein {
     private String name;
     private int spiele;
     private Punkte punkte;
     private Tore tore;
     private Statistik statistik;


     public String getName() { return name;    }

     public int getSpiele() {
           return spiele;
     }

     public Punkte getPunkte() {
           return punkte;
     }

     public Tore getTore() {
           return tore;
     }

     public Statistik getStatistik() {
           return statistik;
     }
}

class Punkte {
     private int plus;

     public int getPlus() {
           return plus;
     }
}

class Tore extends Punkte {
     private int minus;

     public int getMinus() {
           return minus;
     }
}

class Statistik {
     private int siege;
     private int unentschieden;
     private int niederlagen;

     public int getSiege() {
           return siege;
```

```
    }

public int getUnentschieden() {
        return unentschieden;
    }

    public int getNiederlagen() {
        return niederlagen;
    }
}
```

Weiterhin existiert folgendes Hauptprogramm:

```
public class Fussball {

    public static void main(String[] args) {
        Liga bundesliga = new Liga();
        // Spielbetrieb ...
        Arrays.sort(bundesliga.getVereine(),
                new VereinsComparator());
        printTabelle(bundesliga);
    }
}
```

Aufgaben:

Teilaufgabe 1:

Implementieren Sie die fehlende Methode `printTabelle`, die auf der Grundlage des übergebenen Parameters vom Typ `Liga` die aktuelle Tabelle der Liga in der oben angegebenen Form auf die Console ausgibt. Die Ausgabe einer Überschrift sowie eine formatierte Ausgabe (mit Leerzeichen) sind dabei nicht notwendig.

Teilaufgabe 2:

Implementieren Sie die zum Sortieren notwendige, aber fehlende Klasse `VereinsComparator`, die das folgende Interface für `T = Verein` implementiert:

```
public interface Comparator<T> {
    /**
     * Compares its two arguments for order.  Returns a negative integer,
     * zero, or a positive integer as the first argument is less than,
     * equal to, or greater than the second.<p>
     *
     *
     * @param o1 the first object to be compared.
     * @param o2 the second object to be compared.
     * @return a negative integer, zero, or a positive integer as the
     *         first argument is less than, equal to, or greater than the
     *         second.
     */
    int compare(T o1, T o2);
}
```

Der Klasse soll folgende Metrik zugrunde liegen:

- Ein Verein ist „kleiner“ (also besser platziert) als ein anderer Verein, wenn er mehr Pluspunkte hat.
- Haben beide Vereine gleich viele Punkte, ist der Verein mit dem größeren Torverhältnis „kleiner“.
- Ist auch das Torverhältnis gleich, ist der Verein, der mehr Tore erzielt hat, „kleiner“.
- Ist auch dieser Wert identisch, sind beide Verein „gleich groß“.

Aufgabe 12:

Seit Java 8 gibt es im Paket `java.util.function` folgendes Interfaces:

```
interface Predicate<T> {

     // Evaluates this predicate on the given argument.
     // @param t the input argument
     // @return true if the input argument matches the
     //         predicate, otherwise false
     boolean test(T t);
}
```

Leiten Sie eine Klasse *MyArrayList* von der Klasse *java.util.ArrayList* ab, die String-Objekte abspeichert. Die Klasse *MyArrayList* soll das Interface *Predicate* implementieren. Die Methode *test* soll dabei genau dann true liefern, wenn der als Parameter übergebene String aktuell in der ArrayList enthalten ist.

Testprogramm:

```
public class UE23Aufgabe12 {

     public static void main(String[] args) {
           MyArrayList list = new MyArrayList();
           list.add("hallo");
           list.add("lieber");
           list.add("dibo");
           // ...
           while (true) {
                 String text = IO.readString("String: ");
                 if (list.test(text)) {
                       System.out.println("enthalten");
                 } else {
                       System.out.println("nicht enthalten");
                 }
           }
     }
}
```

Aufgabe 13:

Gegeben seien folgende Interfaces und Klassen:

```
interface Verkuppler<T> {
  T[] verkuppeln();
}

class Person {
```

```
  private String name;

  public Person(String name) {
    this.name = name;
  }

  public String getName() {
    return this.name;
  }
}

class Paar {
  private Person person1;
  private Person person2;

  public Paar(Person p1, Person p2) {
    this.person1 = p1;
    this.person2 = p2;
  }

  public String toString() {
    return this.person1.getName() + "+" + this.person2.getName();
  }
}

public class Verkuppeln {

  // Testprogramm
  public static void main(String[] args) {
    PersonArrayList list = new PersonArrayList();
    list.add(new Person("Karl"));
    list.add(new Person("Luise"));
    list.add(new Person("Otto"));
    list.add(new Person("Maria"));
    list.add(new Person("Paul"));
    for (Paar paar : list.verkuppeln()) {
      System.out.println(paar.toString());
    }
  }
}
```

Implementieren Sie die fehlende Klasse *PersonArrayList*. Bei einem Objekt dieser Klasse handelt es sich um eine *ArrayList*, die Objekte vom Typ *Person* speichern kann. Leiten Sie die Klasse also entsprechend von der Klasse *java.util.ArrayList* ab.

Die Klasse *PersonArrayList* soll weiterhin das Interface *Verkuppler* implementieren, und zwar auf folgende Art und Weise:

Die Methode *verkuppeln* soll ein Array von *Paar*-Objekten liefern. Jedes *Paar*-Objekt kapselt dabei zwei *Person*-Objekte. Das Array von Paar-Objekten wird dabei gebildet aus den Person-Objekten, die aktuell in der PersonArrayList gespeichert sind, und zwar unter Berücksichtigung der entsprechenden Reihenfolge: Die erste Person wird mit der zweiten verkuppelt, die dritte mit der vierten, usw. Bei einer ungeraden Anzahl an gespeicherten Person-Objekten geht die letzte Person dabei leer aus (d.h. wird ignoriert).

Beispiel:

In der main-Methode der Klasse *Verkuppeln* werden anfangs 5 Personen einer PersonArrayList hinzugefügt. Die Methode *verkuppeln* in der foreach-Schleife liefert daher ein Array mit 2 Paar-Objekten. Im ersten Paar-Objekt sind die Personen Karl und Luise verkuppelt, im zweiten Paar-Objekt die Personen Otto und Maria. Paul als fünfte Person der PersonArrayList geht beim Verkuppeln leer aus. Als Ausgabe liefert das Programm daher:

```
Karl+Luise
Otto+Maria
```