

Programmierkurs Java

Dr. Dietrich Boles

Aufgaben zu UE38-Generics (Stand 28.09.2012)

Aufgabe 1:

Schauen Sie sich die generischen Klassen des Paketes `java.util` an.

Aufgabe 2:

Ausgangslage sind die folgenden Klassen:

```
abstract class Spieler {}

class FussballSpieler {
    FussballSpieler(String name)
    String getName()
    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 3:

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 Objekten eines bestimmten Typs aufnehmen können, jedes Objekt aber maximal einmal.

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

Auf dem Datentypen Menge sollen folgende Funktionen möglich sein:

- a) Erzeugen einer neuen leeren Menge.
- b) Klonen einer Menge, d.h. Erzeugen einer neuen Menge aus einer alten
- c) Überprüfen auf Gleichheit zweier Mengen
- d) Hinzufügen eines Objektes zu einer Menge.
- e) Entfernen eines Objektes aus einer Menge.
- f) Überprüfung, ob ein bestimmtes Objekt in der Menge enthalten ist.
- g) Schnittmengenbildung zweier Mengen.
- h) Vereinigung zweier Mengen.
- i) Differenzbildung zweier Mengen.

Wenn Fehler auftreten können, setzen Sie Exceptions ein. Dokumentieren Sie die Methoden mit Javadoc. Schreiben Sie weiterhin ein kleines Testprogramm für die Klasse *Menge*.

Aufgabe 4:

Implementieren Sie selbst eine generische Klasse `ArrayList`

Aufgabe 5:

Implementieren Sie eine generische Klasse `Stack`

Aufgabe 6:

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.

- (a) Nutzen Sie dann die Methode `Collections.sort` zum Sortieren der `ArrayList`.
- (b) 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 7:

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 8:

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 9:

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 10:

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 11:

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
```


