Kapitel 5: Generics

Lernziele

[LZ 5.1] Die Vorteile von typsicheren Datenstrukturen kennen
[LZ 5.2] Parametrisierbare Datentypen instanziieren und nutzen können
[LZ 5.3] Collection-Klassen und dazugehörige Interfaces kennen und anwenden können
[LZ 5.4] Verstehen, warum auch die Collection-Interfaces generisch sein müssen
[LZ 5.5] Generische Interfaces nutzen und implementieren können
[LZ 5.6] Das Iterator-Konzept verstehen und anwenden können
[LZ 5.7] Eigene Klassen und Interfaces parametrisierbar definieren und einsetzen können

5. Generics

[LZ 5.1] Die Vorteile von typsicheren Datenstrukturen kennen

- Welche Nachteile hat die Verwendung von „raw types“ z.B. im Zusammenhang mit ArrayList? Erläutern Sie anhand eines Beispiels.
- Erläutern Sie anhand Ihres obigen Beispiels die Vorteile der Verwendung eines Typ-Parameters.

5. Generics

[LZ 5.2] Parametrisierbare Datentypen instanziieren und nutzen können

- Zeigen Sie anhand verschiedener Deklarationen und anschließender Zugriffe die Verwendung von Listen, Sets und Maps

5. Generics

[LZ 5.3] Collection-Klassen und dazugehörige Interfaces kennen und anwenden können

- Nennen Sie die wesentlichen Klassen und Interfaces im Bereich der Collections
- Welches sind jeweils die wesentlichen Operationen ?
- Zeigen Sie die Anwendung jeder Klasse / Interface anhand eines kurzen Beispiels!

5. Generics

[LZ 5.4] Verstehen, warum auch die Collection-Interfaces generisch sein müssen

- Warum verwendet man Interfaces im Zusammenhang mit Collections?
- Welche Interfaces sind dies?

Gegeben sei folgende Klasse:

```
class Data implements Comparable {
    private int datum;
    public Data(int i) { datum = i; }
    public String toString() { return "" + datum; }
    public int compareTo(Object o) {
        Data d = (Data) o;
        return -(this.datum - d.datum);
    }
    public static void main(String[] args) {
    Data test[] = { new Data(3), new Data(2) };
    java.util.Arrays.sort(test);
    for (Data s : test) System.out.print(s + " ");
    }
}
```

Data implementiert Comparable bezogen auf Object, d.h. ein Data-Objekt kann mit jedem anderen Objekt verglichen werden (prinzipiell, nicht mit der aktuellen Implementierung!) Ändern Sie obige Klasse, so dass Data-Objekte nur mit Ihresgleichen verglichen werden können und der Compiler das bereits überprüft!

- Gegeben sei folgendes Array:

```
int[] intArray = new int[] { 7, 34, 9, 19 };
```

  Arrays implementieren Iterable. Geben Sie – unter Rückgriff auf das Iterable-Interface – eine Iteration über obiges Array an, welche alle Elemente des Arrays zeilenweise auf die Console ausgibt.

- Bauen sie die Wrapper-Klasse Integer nach, z.B. als MyInteger. Ein MyInteger soll seinen Wert über den Konstruktor erhalten und den Wert per intValue-Methode liefern. Testen Sie diese Eigenschaften über eine geeignete main-Methode.

5. Generics

[LZ 5.7] Eigene Klassen und Interfaces parametrisierbar definieren und einsetzen können

- Definieren Sie ein parametrisierbares Interface IMeineListe<T>, welches Elemente vom Typ T speichert und Operationen zum Hinzufügen am Ende der Liste, zum Entfernen anhand eines Index und zum Lesen anhand eines Index bietet.
- IMeineListe soll von Iterable erben
- Implementieren Sie dieses Interface durch eine Klasse MeineListe<T>
- Testen Sie Beides anhand der Typen String und Integer

5. Generics

Übungen zu Collections

Listen

Erzeugen Sie eine Liste (ArrayList) mit 1400 ganzzahligen Zufallszahlen zwischen 0 und 42. Verwenden Sie dazu die Methode nextInt() aus der Klasse Random. Bilden Sie den Durchschnitt der Zufallszahlen, indem Sie einen ListIterator verwenden. Sortieren Sie die Zufallszahlen anschließend mit Hilfe der Collection-Algorithmen. Kopieren Sie nun den Inhalt der Liste in eine LinkedList und ändern Sie die Reihenfolge der Elemente zufällig (Methode shuffle). Sortieren Sie die Liste erneut. Bestimmen Sie Minimum und Maximum der Liste.

Sets

Speichern Sie die obige unsortierte Liste in einem Set. Fügen Sie dazu mit Hilfe eines Iterators die Elemente der Liste in umgekehrter Reihenfolge der Set hinzu. Erzeugen Sie aus dem Set eine sortierte Menge (SortedSet).

Maps

Bestimmen Sie die Häufigkeiten der Zahlen in der obigen Liste mit Hilfe einer HashMap. Erzeugen Sie dann eine sortierte Map (SortedMap), in der die Schlüssel in absteigender Reihenfolge gespeichert werden. Verwenden Sie dazu das Comparator-Interface. Wie ließe sich das gleiche mit dem Comparable-Interface realisieren?

Eigene Datenobjekte

Speichern Sie Studentendaten (s. Klassendef. rechts) in einer Liste und sortieren Sie die Liste einmal nach den Namen (mit Hilfe des Comparable-Interface) und einmal nach Matrikelnummern (mit Hilfe des Comparator-Interface)

```
class Student {
    protected String name;
    protected int matrNr;
}
```

Erweiterung der Studentenverwaltung

Die Studentenverwaltung (s. Kursraum) soll nun folgendermaßen erweitert werden:

Ergänzen das Menü und das Programm um eine Suchfunktion, die effizient einen Studenten nach seiner Matrikelnummer findet. Für die Suchfunktion soll zusätzlich zu der bestehenden Liste (LinkedList) eine weitere Datenstruktur aufgebaut werden, die die schnelle Suche ermöglicht. Um zu gewährleisten, dass beide Datenstrukturen dieselben Daten enthalten, soll eine Methode studentEintragen(Student) entwickelt werden, die einen Studenten-Datensatz einträgt.

Entwickeln Sie eine Java-Klasse „Hochschule“, die eine Iteration über Studenten-Objekte folgendermaßen zulässt:

Schritte

1. Erstellen Sie ein UML-Design (Klassenmodell)
2. Implementieren Sie ihr Design

```
public static void main(String[] args) {
    List<Student> studenten = new ArrayList<Student>();
    studenten.add(new Student(1, "Anna");
    studenten.add(new Student(2, "Peter"));
    Hochschule hochschule = new Hochschule(studenten);
    for (Student student : hochschule)
        System.out.println(student);
}
```

5.5 Übung zu Generics: eigene parametrisierbare Klasse definieren

- Entwickeln Sie eine parametrisierbare Klasse ‚ValueHolder', die folgendermaßen genutzt werden kann:

```
ValueHolder<String> vh1 = new ValueHolder<String>("Hallo");
ValueHolder<Integer> vh2 = new ValueHolder<Integer>(1);
ValueHolder<Double> vh3 = new ValueHolder<Double>(1.2);

System.out.println(vh1);
System.out.println(vh2);
System.out.println(vh3);
```

- Obiger Code führt zur Ausgabe

```
Hallo
1
1.2
```