Proseminar Programmiermethodik

Universität Innsbruck - Institut für Informatik

A. Blaas, M. Mayerl, B. Murauer, M. Tschuggnall

Blatt 6:Generics

Diskussionsteil

a) ★ Implementieren Sie eine generische Klasse `Pair<T, U>`. Ein `Pair` ist ein Objekt, das aus zwei Werten besteht: einem Wert vom Typ `T` und einem Wert vom Typ `U`. Die beiden Werte sollen über die Getter/Setter `getFirst()`, `getSecond`, `setFirst(T value)` und `setSecond(U value)` angesprochen werden können.

Verwenden Sie folgende `main`-Methode (ohne Modifikationen!), um Ihre Klasse zu testen:

```
1       public static void main(String[] args) {
2           Pair<String, Integer> myPair = new Pair<>("Hallo", 10);
3           System.out.format("Pair is: (%s, %d)\n",
4               myPair.getFirst(), myPair.getSecond());
5
6           myPair.setFirst("World");
7           System.out.format("Pair is: (%s, %d)\n",
8               myPair.getFirst(), myPair.getSecond());
9
10          myPair.setSecond(20);
11          System.out.format("Pair is: (%s, %d)\n",
12              myPair.getFirst(), myPair.getSecond());
13      }
```

b) ★ Diskutieren Sie mit Ihren Kolleginnen und Kollegen, welche Vorteile Generics gegenüber nicht generischem Code bieten. Denken Sie dabei vor allem an

- Typsicherheit,
- sauberen, kurzen Code und
- Redundanzen.

c) ★★ Besteht zwischen folgenden beiden Methoden ein Unterschied?

```
1       public <T> void doSomething(List<T> input) { /* ... */ }
2       public void doSomething(List<?> input) { /* ... */ }
```

Wenn ja, welcher? Was ist mit diesen beiden Methoden?

```
1       public <T> void doSomething(Map<T, T> input) { /* ... */ }
2       public void doSomething(Map<?, ?> input) { /* ... */ }
```

d) ★★ Implementieren Sie eine statische generische Methode `max(List<T> list)`, die das maximale Element der übergebenen Liste ermittelt. Für den Fall, dass die übergebene Liste leer oder `null` ist, soll eine `IllegalArgumentException` geworfen werden.

Hinweis ⚠

Damit das Maximum ermittelt werden kann, müssen die Element der Liste das Interface `Comparable<T>` implementieren.

Verwenden Sie folgende `main`-Methode, um Ihre Implementation zu testen:

```
1       public static void main(String[] args) {
2           List<String> list1 = new ArrayList<>();
3           list1.add("Naja");
4           list1.add("Zyzzyva");
5           list1.add("Aardvark");
6           list1.add("Velociraptor");
7           System.out.println(max(list1));
8
9           List<Integer> list2 = new LinkedList<>();
10          list2.add(3);
11          list2.add(2);
12          list2.add(1);
13          list2.add(3);
14          list2.add(2);
15          System.out.println(max(list2));
16
17          try {
18              max(null);
19          }
20          catch (IllegalArgumentException ex ) {
21              System.out.println("IllegalArgumentException was thrown!");
22          }
23      }
```

Übungsteil (selbstständig zu lösen)

Aufgabe 1 (Generic Observer) [4 Punkte]

In dieser Aufgabe werden Sie eine generische Version des Observer-Patterns[1] implementieren. Das Observer-Pattern bietet eine Möglichkeit, den Programmfluss von einem Pull-basierten Modell (ein

[1] https://en.wikipedia.org/wiki/Observer_pattern

Objekt fragt wiederholt den Zustand eines anderen Objektes ab, um Änderungen zu erkennen) auf ein Push-basiertes Modell (ein beobachtetes Objekt – Subject oder Observable genannt – informiert alle interessierten Objekte – die Observer – über Änderungen an seinem Zustand) umzustellen.
In dieser Aufgabe werden Sie dieses Pattern benutzen, um smarte Glühbirnen und Temperatursensoren zu modellieren.

a) *0.5 Punkte* Erstellen Sie die beiden generischen Interfaces `Observable<T>` und `Observer<T>`.

   `Observer<T>` stellt die Methode `update(Observable<T> sender, T value)` zur Verfügung. Die Methode wird von einem `Observable` aufgerufen, wenn es eine Zustandsänderung gegeben hat.

   `Observable<T>` stellt die Methode `subscribe(Observer<T> observer)` zur Verfügung. Mit dieser Methode kann sich ein `Observer` zur Liste der Beobachter eines `Observables` hinzufügen.

b) *1 Punkt* Erstellen Sie eine Klasse `Temperature`. Diese Klasse soll Temperatur-Werte verschiedener Einheiten (Celsius, Kelvin und Fahrenheit) abbilden können. Speichern Sie dazu intern die Temperatur in einer beliebigen Einheit und bieten Sie für den Benutzer der Klasse die Methoden `getCelsius()`, `getKelvin()` und `getFahrenheit()` sowie `setCelsius(double value)`, `setKelvin(double value)` und `setFahrenheit(double value)` an, welche die Temperatur entsprechend umrechnen.

   Zum Erstellen neuer `Temperature`-Instanzen verwenden Sie statische Factory-Methoden, wie Sie sie in Aufgabenblatt 3 kennengelernt haben. Eine neue `Temperature`-Instanz soll dann beispielsweise wie folgt angelegt werden können:

   ```
   1        Temperature t = Temperature.fromCelsius(25.0);
   ```

c) *1 Punkt* Erstellen sie nun zwei Klassen für unterschiedliche Arten von Temperatur-Sensoren: `CelsiusTemperatureSensor` und `FahrenheitTemperatureSensor`. Der einzige Unterschied zwischen den beiden Klassen besteht in der Einheit, in der sie ihre Messungen machen. Beide Klassen implementieren das Interface `Observable<Temperature>`.

   Außerdem stellen beide Klassen die Methode `startMeasuring(int numberOfMeasurements)` zur Verfügung. Wenn diese Methode aufgerufen wird, führt der Sensor `numberOfMeasurements` Messungen durch und informiert alle registrierten Observer über jede gemachten Messung, indem ihre `update`-Methode aufgerufen wird. Verwenden Sie für die Messungen einfach Zufallszahlen in einem realistischen Temperaturbereich. Bevor die registrierten Observer von einer Messung informiert werden, wird die gemessene Temperatur auf der Konsole ausgegeben.

   Überlegen Sie sich auch, wie Sie Redundanzen im Code (also den gleichen Code in beiden Klassen) vermeiden können.

d) *1 Punkt* Erstellen Sie jetzt die Klasse `SmartLightBulb`. Die Klasse implementiert das Interface `Observer<Temperature>`. Darüber hinaus stellt sie einen Konstruktor zur Verfügung, der ein `Observable<Temperature>` – also etwa einen Temperatur-Sensor – als Parameter entgegennimmt. Wird eine Instanz der Klasse über diesen Konstruktor angelegt, registriert sich die neue Glühbirne bei dem als Parameter übergebenen Sensor als Observer, indem sie dessen `subscribe`-Methode aufruft.

   Die `update`-Methode von `SmartLightBulb` gibt die übergebene Temperatur auf der Konsole aus. Die Ausgabe muss dabei deutlich von der Ausgabe der Sensoren unterscheidbar sein.

   Sie müssen keine weitere Logik der Glühbirne implementieren – was sie mit der gemessenen Temperatur macht, bleibt also auf ewig ein Mysterium.

e) *0.5 Punkte* Testen Sie ihre Implementation mit folgender `main`-Methode und speichern Sie die Ausgabe des Programms als Textdatei ab.

```
1       public static void main(String[] args) {
2           TemperatureSensor sensor1 = new CelsiusTemperatureSensor();
3
4           SmartLightBulb bulb1 = new SmartLightBulb(sensor1);
5           SmartLightBulb bulb2 = new SmartLightBulb(sensor1);
6
7           sensor1.startMeasuring(10);
8
9           TemperatureSensor sensor2 = new FahrenheitTemperatureSensor();
10
11          SmartLightBulb bulb3 = new SmartLightBulb(sensor2);
12          SmartLightBulb bulb4 = new SmartLightBulb(sensor2);
13          SmartLightBulb bulb5 = new SmartLightBulb(sensor2);
14
15          sensor2.startMeasuring(10);
16      }
```

Aufgabe 2 (Generische Algorithmen) [3 Punkte]

In dieser Aufgabe werden Sie einige generische Algorithmen auf beliebigen Collections von Zahlen implementieren.

Hinweis ⚠

Alle relevanten Zahlen-Klassen (`Byte`, `Short`, `Integer`, `Long`, `Float` und `Double`) leiten von `java.lang.Number` ab.

Kollege Testardo hat für diese Algorithmen bereits einige Unit Tests verfasst. Importieren Sie daher das Projekt `GenericAlgorithms.zip` (OLAT). In diesem Projekt befinden sich zwei Java-Dateien. Die Datei `NumberAlgorithms.java` soll letzten Endes die von Ihnen implementierten Algorithmen enthalten. Die Datei `NumberAlgorithmsTest.java` enthält die von Toni erstellten Unit Tests. Öffnen sie nun also die Datei `NumberAlgorithms.java` und implementieren Sie dort die unten angegebenen statischen Methoden. Stellen Sie dabei sicher, dass jede Methode eine `IllegalArgumentException` wirft, wenn eine leere Collection oder `null` übergeben wird.

a) *0.5 Punkte* `double min(Collection<T> col)`: Ermittelt den kleinsten Wert in der übergebenen Collection und gibt diesen als `double` zurück.

b) *0.5 Punkte* `double max(Collection<T> col)`: Ermittelt den größten Wert in der übergebenen Collection und gibt diesen als `double` zurück.

c) *0.5 Punkte* `double sum(Collection<T> col)`: Ermittelt die Summe aller Elemente in der übergebenen Collection und gibt diese als `double` zurück.

d) *0.5 Punkte* `double avg(Collection<T> col)`: Ermittelt das arithmetische Mittel aller Elemente in der übergebenen Collection und gibt dieses als `double` zurück.

e) *0.5 Punkte* `double var(Collection<T> col)`: Ermittelt die Varianz aller Elemente in der übergebenen Collection und gibt diese als `double` zurück. Verwenden Sie für die Berechnung der Varianz die Formel für Populationen, sprich:

$$\sigma^2 = \frac{1}{N}\sum_{i=1}^{n}(x_i - \mu)^2 \tag{1}$$

wobei N die Anzahl der Elemente in der Collection, x_i das i-te Elemente in der Collection und μ die Durchschnitt aller Elemente in der Collection ist.

f) *0.5 Punkte* `double stddev(Collection<T> col)`: Ermittelt die Standardabweichung aller Elemente in der übergebenen Collection und gibt diese als `double` zurück. Verwenden Sie für die Ermittlung der Standardabweichung die Formel für Populationen, sprich:

$$\sigma = \sqrt{\frac{1}{N}\sum_{i=1}^{n}(x_i - \mu)^2} = \sqrt{\sigma^2} \tag{2}$$

Hinweis ⚠

Um den Wert einer `Number`-Instanz als `double` auszulesen, können Sie die `doubleValue()`-Methode verwenden.

Mithilfe der Unit Tests können Sie jederzeit überprüfen, ob ihre Methoden richtig funktionieren. Führen Sie diese also zumindest nachdem Sie alle geforderten Methoden fertig implementiert haben aus, um Ihre Arbeit zu überprüfen.

Aufgabe 3 (Theorie) [3 Punkte]

In dieser Aufgabe werden Sie einige Theoriefragen zu Java-Generics beantworten.

a) *1 Punkt* Java implementiert Generics – im Gegensatz zu manch anderen Programmiersprachen – als Compiler-Feature. Dafür benutzt es eine Technik namens Type Erasure. Erklären Sie in einigen Sätzen, was Type Erasure ist und wie es funktioniert. Geben Sie anschließend ein kurzes Beispiel dafür, wie eine generische Klasse vor und nach Type Erasure aussieht.

b) *2 Punkte* Aufgrund der Art und Weise, wie Java Generics implementiert – siehe vorherige Teilaufgabe – unterliegen Generics in Java einigen Einschränkungen. So ist es beispielsweise nicht möglich, einen primitiven Typen als Typ-Parameter zu verwenden (aus diesem Grund müssen Sie zum Beispiel `List<Integer>` statt `List<int>` verwenden).

Recherchieren Sie (zum Beispiel im Internet), welche weiteren Einschränkungen es für Generics in Java gibt. Finden Sie mindestens vier solcher Einschränkungen und erklären Sie jede davon mit einigen Sätzen und einem kurzen Beispiel.

Wichtig: Laden Sie bitte Ihre Lösung (.txt, .java oder .pdf) in OLAT hoch und geben Sie mittels der Ankreuzliste auch unbedingt an, welche Aufgaben Sie gelöst haben. Die Deadline dafür läuft am Vortag des Proseminars um 23:59 (Mitternacht) ab.