Übung 2 – Generics 1

Arbeiten Sie im Skript die *Kapitel Generische Klassen* und *Generische Schnittstellen* durch. Suchen Sie sich weitere Informationen, wo Sie die Ausführungen nicht verstehen. Sprechen Sie mit Ihren Kommilitoninnen und Kommilitonen. Fragen Sie im Forum. **Lesen Sie die Aufgaben vollständig und markieren Sie sich zentrale Aspekte.** Verwenden Sie für die Benennung Ihrer Testmethoden die im Skript definierten Konventionen (*Kapitel Unittesting*).

Aufgabe 1

Laden Sie sich aus dem LEA-Kurs die Datei `Annotationen.java` herunter. Kompilieren Sie die Datei über die Kommandozeile mit folgendem Befehl:
`javac -Xlint:unchecked Annotationen.java`
Was fällt Ihnen auf? Wo werden Probleme gemeldet? Fügen Sie im Anschluss die folgende Annotation im Quellcode ein `@SuppressWarnings(\string"unchecked")` und kompilieren Sie die `.java`-Datei mit dem oben angegebenen Befehl erneut. Was bewirkt die Annotation?

Aufgabe 2

Wir wollen in den folgenden Aufgaben `Puffer`, `Stapel` und `Schlange` generisch machen.
Betrachten Sie ihr UML-Diagramm vom ersten Aufgabenblatt. Überlegen Sie zuerst welche Stellen Sie ändern müssen, sodass `Puffer`, `Stapel` und `Schlange` generische Schnittstellen und `StapelMitArray` und `SchlangeMitArray` generische Klassen werden. Zeichnen Sie ein neues UML-Diagramm in welchem Sie Ihre Änderungen übernehmen.

Aufgabe 3

Implementieren Sie jetzt die veränderten Klassen und Schnittstellen Ihres UML-Diagramms.
Hinweis: Beachten Sie dabei, dass Sie keine Objekte vom generischen Typ `T` instanzieren können und somit auch kein `T[] Objekt`!
(→ *Video: Programmierbeispiel Generics*)

Aufgabe 4

Erstellen Sie neue JUnit5 Tests, die die veränderten Klassen für mindestens zwei weitere Datentypen testen. *Welche Datentypen können Sie für `T` einsetzen?*

Aufgabe 5

Laden Sie sich den Code zu `GenericsFehler` aus LEA herunter. Kommentieren Sie in der `main` in jeder wichtigen Zeile genau was passiert. Wo entstehen welche Fehler? Wo sind die Unterschiede zwischen `MyObject` und `MyGeneric`?

Aufgabe 6

Modellieren Sie im Folgenden ein *Paar*. Ein *Paar* besteht aus zwei Werten, die als Einheit betrachtet werden. Diese Werte müssen nicht den selben Typ haben. Zulässige Paare wären also bspw. zwei Personen, oder auch eine Einheit wie Name und Geburtsdatum.

Schreiben Sie eine generische Klasse `Paar<E,Z>` zur Repräsentation von Paaren von zwei Werten von beliebigen Referenztypen `E` und `Z`. Die beiden Referenzen sollen dem `Paar`-Konstruktor als Parameter übergeben werden.

Die generische Klasse `Paar` soll folgende Instanzmethoden anbieten:

- `getErstes()` und `getZweites()`
  geben die beiden Komponenten des Paares zurück.
- `setErstes(E e)` und `setZweites(Z z)`
  setzen die jeweilige Komponente auf den übergebenen Wert. Sie geben den vorherigen Wert der jeweiligen Komponente zurück, den sie vor dem Überschreiben hatte.
- `setBeide(E e, Z z)`
  setzen die beiden Komponenten jeweils auf den übergebenen Wert. Geben Sie keinen Wert zurück.
- `equals(Paar<E,Z> p)`
  gibt als Wahrheitswert zurück, ob das übergebene Paar wertgleich zu den Werten der Instanz ist. Dazu vergleicht die Methode die Komponenten der beiden Paare.

- `toString()`
  erzeugt für das Paar eine Darstellung als String in der Form "(e,z)", worin die beiden Platzhalter für die Stringdarstellungen der beiden Objekte stehen und die Anführungszeichen nicht Teil der Zeichenkette sind.

Welche Typen können Sie für `E` und `Z` bei Instanziierung einsetzen?

Aufgabe 7

Schreiben Sie JUnittests für Ihre Klasse `Paar<E,Z>`.

- Verwenden Sie für Ihre Tests Paare aus einer Kombination von `Integer` und `String`.
- Implementieren Sie Unittests für die Methoden `setErstes(E e)` und `getErstes()`, sowie `setZweites(Z z)` und `getZweites()`.
- Testen Sie außerdem die Methoden `setBeides(E e, Z z)`,`equals(Paar<E,Z>p)` und `toString()`.

Aufgabe 8

Im Folgenden wollen wir jetzt unsere generischen Klassen bzw. Schnittstellen verbinden und einen `Puffer` mit `Paaren` erzeugen. Wir wollen damit zwei Sachen modellieren:

1. Den ServicePoint, der Fragen von Studierenden nacheinander annimmt. Die erste Frage von einem/einer Studierenden soll zuerst bearbeitet werden können.
2. Ein Event mit Boxkämpfen, bei dem wir den Hauptkampf zuerst planen, dieser soll aber zuletzt am Abend stattfinden.

Gehen Sie dabei wie folgt vor:

- Erstellen Sie eine Klasse `PufferPaar`, die ledigliche eine `main`-Methode enthält.
- Deklarieren Sie in der `main` zwei Variablen vom Typ `Puffer`. Die eine Variable nennen Sie `servicePoint`. Als Typparameter soll sie ein `Paar` mit `Student` und `String` bekommen. Die zweite Variable nennen Sie `boxEvent`. Diese Variable soll als Typparameter ein `Paar` mit jeweils `Boxer` bekommen. Überlegen Sie vorher genau wie die Deklaration aussehen muss.
- Instanziieren Sie nun `servicePoint` als Objekt der Klasse `SchlangeMitArray` und `boxEvent` als Objekt der Klasse `StapelMitArray`.
- Überlegen Sie sich vier Studierende mit ihren zugehörigen Fragen für den `servicePoint` und drei Paarungen für das `boxEvent`.

- Fügen Sie diese Objekte in Ihre jeweilige `Puffer`-Variablen hinzu.
- Entnehmen Sie schließlich in jeweils einer Schleife die Elemente aus den Puffer-Objekten und geben diese zeilenweise auf dem Bildschirm aus.

Zusatzaufgabe (Programmierübung Projekt Euler)

Wenn wir alle natürlichen Zahlen unter 10 auflisten, die Vielfache von 3 oder 5 sind, so erhalten wir 3, 5, 6 und 9. Die Summe dieser Vielfachen ist 23. Finden Sie die Summe aller Vielfachen von 3 oder 5 unter 1000.

`https://projekteuler.de/problems/1`

a. Implementieren Sie die Aufgabe als statische Methode, die einen `int`-Wert `range` an nimmt (für das obige Beispiel 10 bzw. 1000) und die Summe als Ganzzahl zurück gibt.

b. Testen Sie Ihre Methode mit JUnit Tests.