**Generics Workshop Übungen**

**Part I**

**Aufgabe 1**

Gegeben sie folgende Implementierung einer Queue, die zwar praktisch ist, aber lediglich für den Typ String ausgelegt ist. Wandeln diese in eine generische Klasse um: (<https://math.hws.edu/javanotes/c10/s5.html>)

**class** QueueOfStrings {

**private** LinkedList<String> items = **new** LinkedList<>();

**public** **void** enqueue(String item) {

items.addLast(item);

}

**public** String dequeue() {

**return** items.removeFirst();

}

**public** **boolean** isEmpty() {

**return** (items.size() == 0);

}

}

**Aufgabe 2**

Schreibe eine generische Mapping-Methode, die die Daten aus einem Array in eine typisierte Liste überführt und zudem eine Aktion ausführen kann. Starte mit folgendem Code: (<https://www.baeldung.com/java-generics>)

**public** **static** <T> List<T> fromArrayToList(T[] a) {

**return** Arrays.*stream*(a).collect(Collectors.*toList*());

}

**Tipp**: Nutze ein passendes Functional Interface

**Bonus**: Sichere die Funktionalität mit einem JUnit Test ab.

**Aufgabe 3 – Sort-Methode auf Generics umbauen**

Gegeben sei eine einfache Implementierung von InsertionSort auf int. Gestalte diese so um, dass sie Generics und den Typ T nutzen und sich auf Comparable<T> abstützen.

**Tipp**: Nutze <T extends Comparable<T>> für die Methodendefinition und zum Vergleich dann compareTo auf den generischen Objekten vom Typ T

**PART II + III**

**Aufgabe 4 – Klassenhierarchie**

Gegeben sei eine einfache Klassenhierarchie mit der Basisklasse BaseFigure und Subklassen Rect und Circle.

1. Wieso funktioniert die Methode BaseFigure[] auch für Rect[]?
2. Erstelle eine neue Methode, die auf der von BaseFigure[] basiert, sodass sie List<BaseFigure> nutzt.

**Aufgabe 5**

Gegeben sei folgender generischer Datencontainer sowie die passenden zu verwaltenden Klassen: (<https://java.codexion.com/java/generics/QandE/generics-answers.html>)

**public** **static** **class** AnimalHouse<E> {

**private** E animal;

**public** **void** setAnimal(E x) {

animal = x;

}

**public** E getAnimal() {

**return** animal;

}

}

**public** **static** **class** Animal {

}

**public** **static** **class** Cat **extends** Animal {

}

**public** **static** **class** Dog **extends** Animal {

}

Was gilt für die Code-Schnipel

* fails to compile,
* compiles with a warning,
* generates an error at runtime, or
* none of the above (compiles and runs without problem.)

Löse es zuerst im Kopf und prüfe danach dann in der IDE. Was sind die Begründungen und wie könnte eine Lösung aussehen?

a. AnimalHouse<Animal> house = new AnimalHouse<Cat>();

b. AnimalHouse<Dog> house = new AnimalHouse<Animal>();

c. AnimalHouse<?> house = new AnimalHouse<Cat>();

house.setAnimal(new Cat());

d. AnimalHouse house = new AnimalHouse();

house.setAnimal(new Dog());

**Aufgabe 6 – copy()-Methode**

Erstelle eine copy()-Methode, die Listen von grafische Figuren ineinander kopieren kann.

**Aufgabe 7 – Allgemeinere sort()-Methode**

Erstelle basierend auf der sort()-Methode aus Aufgabe 3 eine Variante, sodass sie nun einen beliebigen Comparator nutzen kann und als Eingabe eine Liste von Werten erhält.

**PART IV**

**Aufgabe 8**

Gegeben sei folgende Sourceode einer Klasse zur Modellierung von Matrizen. Wie lösen wir das Problem der dynamischen Erzeugung eines Arrays?

**static** **class** Matrix<T> {

**private** T[][] values;

**private** **int** rows;

**private** **int** cols;

**public** Matrix(**int** r, **int** c) {

rows = r;

cols = c;

//values = new T[rows][cols]; // COMPILE ERROR

}

**public** **void** set(**int** r, **int** c, T v) {

values[r][c] = v;

}

**public** T get(**int** r, **int** c) {

**return** values[r][c];

}

}

**Aufgabe 9**

Gegeben sei folgende Sourceode einer Klasse für ein Singleton. Kompiliert dieser?

**public** **class** Singleton<T> {

**public** **static syncronized** T getInstance() {

**if** (instance == **null**)

instance = **new** Singleton<T>();

**return** instance;

}

**private** **static** T instance = **null**;

}