JAVA 10

Generics

JAVA 10 - Generics

1. **Beispiel Collections**
2. **Warum Generics?**
3. **Eigene Generische Klassen**
4. **Eigene Generische Methoden**
5. **Generics und Vererbung**

1. Beispiel Collections

Allgemeine Definition:

```
ArrayList<T> list = new ArrayList<>();
```

Explizites Beispiel:

```
ArrayList<Integer> list = new ArrayList<>();
list.add(1);    // Es können nur Zahlen übergeben werden
```

Hinweis: Wenn der Typ für die Collection einmal definiert wurde, kann er nicht mehr verändert werden. Wird kein Typ definiert, wird alles als Objekt der Klasse „**Object**“ interpretiert.

2. Warum Generics?

- Unübersichtlicher Code durch **Casts (**bzw. **„Boxing“)**
- Casts können **ClassCastExceptions** verursachen

```
Object o = "7";          // ein allgemeines Objekt
                         // das zu einem String “wird”
int i;                   // eine definitive Zahl
i = (int) o;             // Programm startet zwar, doch
                         // stürzt es mit einem Fehler ab!
```

- Compiler kann **Typsicherheit** nicht sicherstellen!
- Mit Generics kann es nicht passieren (Programm startet nicht)

3. Eigene Generische Klassen

```
public class MyGeneric<T> {
	private T member;
	public MyGeneric(T member) {
		this.member = member;
	}
}
MyGeneric<Integer> myInt = new MyGeneric<>();
MyGeneric<Float> myFloat = new MyGeneric<>();
```

- Die **generische Klasse** ist in der Lage theoretisch **alles** aufzunehmen
- Das **konkrete Objekt** arbeitet zur Laufzeit dann (nur) mit dem **expliziten Typ**

4. Eigene Generische Methoden

```
public class MyClass {
        private T something;
        public <T> void setSomething(T something) {
                this.something = something;
        }
        public T getSomething() { return this.something; }
        public <T> T combine(T something) {
                // ...
                return this.something;
        }
}
```

Generell funktionieren generische Methoden genauso wie normale Methoden, nur das die explizite Klasse/der explizite Typ erst bei Verwendung – also bei der Erstellung des Objekts definiert wird!

5. Generics und Vererbung

```
public class Father {}
public class <T extends Father> MyClass {}
```

- Um die generischen Typen einzuschränken, kann man definieren, dass das explizite Objekt zumindest von einer Oberklasse (hier Father) oder einer von dieser ableitenden Klasse sein muss
- Macht Sinn um Eigenschaften vorauszusetzen, die in der generischen Klasse genutzt werden sollen.
- Dies funktioniert auch mit Interfaces – also nur Objekte von Klassen zulassen, die das gewünschte Interface explizit implementieren:

```
public interface Countable {}
public class <T extends Countable> MyClass {}
```

- Selbiges gilt auch für generische Methoden
- Es gibt weitere Detail-Aspekte, die wir aber nicht behandeln werden…