# GENERICS IN JAVA

# WARUM GENERICS?

```java
List animals = new ArrayList();
animals.add(new Animal("Rex"));
animals.add(new Animal("Kitty"));
animals.add("hallo");

for (int i = 0; i < animals.size(); i++) {
    Animal animal = (Animal) animals.get(i);
    System.out.println(animal.getName());
}
```

Ausgabe (Laufzeit)

```
Rex
Kitty
Exception in thread "main"
java.lang.ClassCastException: class
java.lang.String cannot be cast to class
Animal
…
```

Ohne Generics: Casting nötig, Fehler zur **Laufzeit**

```java
List<Animal> animals = new ArrayList<>();
animals.add(new Animal("Rex"));
animals.add(new Animal("Kitty"));
animals.add("hallo");

for (int i = 0; i < animals.size(); i++) {
    Animal animal = animals.get(i);
    System.out.println(animal.getName());
}
```

Ausgabe (Compiler)

```
/Users/…/MeineDatei.java:21:21
incompatible types: java.lang.String
cannot be converted to Animal
```

Mit Generics: Kein Casting, Fehler zur **Compile-Zeit**

```
public static <U> String foo(U myThing) {
    …
}
```

Definition eines Typ-Parameters

Nutzung des Typ-Parameters

Auch bei Interfaces möglich!

```
public class Shelter<T> {

    private final List<T> residents = new ArrayList<>();

    public void addResident(T resident) {
        residents.add(resident);
    }
}
```

<T> kann ausgelassen werden, da es sich der Compiler herleiten kann

## ERWEITERTE TYP-PARAMETER DEFINITION

- Typ-Parameter können bei der Definition eingeschränkt werden:

```
public static <T extends Animal> String foo(T myThing) {…}

public class Something<T extends MyClass & MyInterface> {…}
```

- Beim Erben können Typ-Parameter festgelegt oder weiter eingeschränkt werden:

```
public class AnimalShelter<T extends Animal> extends Shelter<T> {…}

public class CatShelter extends AnimalShelter<Cat> {…}
```

- Es können mehrere Typ-Parameter auf einmal definiert werden:

```
public class Something<T extends MyClass & MyInterface, U, V extends Animal> {…}
```

## WILDCARDS <?>

- Falls der Typ-Parameter bei der Benutzung egal ist, kann <?> eingesetzt werden
- Kann auch eingeschränkt werden, z.B. <? extends Animal>

```
public class Shelter<T> {
    private List<T> residents;
    private String name;

    public Shelter(String name) {
        this.name = name;
        this.residents = new ArrayList<>();
    }

    public void addResident(T resident) {
        residents.add(resident);
    }

    public String getName() {
        return name;
    }
}
```

```
public void printName(Shelter<?> shelter) {
    String name = shelter.getName();
    System.out.println(name);
}
```

## FAZIT

- Vorteile von Generics
  - Fehler werden zur Compile-Zeit sichtbar
  - Vermeidet Casting
- Generics/ Typ-Parameter definierbar bei
  - Klassen
  - Interfaces
  - Methoden
- Generic-Typen lassen sich einschränken: <T extends MyClass & MyInterface>
- Wildcard <?> nutzbar, wenn Typ bei Benutzung egal

# QUELLEN

- https://docs.oracle.com/javase/tutorial/java/generics/index.html