

Lernzettel

Generische Datentypen in Java

Universität: Technische Universität Berlin
Kurs/Modul: Einführung in die Informatik - Vertiefung
Erstellungsdatum: September 20, 2025



Zielorientierte Lerninhalte, kostenlos!
Entdecke zugeschnittene Materialien für deine Kurse:

<https://study.AllWeCanLearn.com>

Einführung in die Informatik - Vertiefung

Lernzettel: Generische Datentypen in Java

(1) **Grundlagen.** Generische Typen ermöglichen eine Typ-Abstraktion. In Java werden Typ-Parameter wie T, K oder V verwendet, z. B. `List<T>` oder `Map<K,V>`. Generics erhöhen Typsicherheit und Wiederverwendbarkeit, da der Compiler Typen prüft und Fehlermeldungen früher liefert.

(2) **Deklaration generischer Klassen.** Beispiel einer generischen Klasse:

```
public class Box<T> {
    private T value;
    public Box(T value) { this.value = value; }
    public T get() { return value; }
}
```

(3) **Generische Methoden.** Generische Methoden führen Typ-Parameter unabhängig von der Klasse ein:

```
public static <T> void printArray(T[] arr) {
    for (T e : arr) System.out.println(e);
}
```

(4) **Standard-Sammlungen mit Generics.** Beispiele:

```
List<String> names = new ArrayList<>();
names.add("Alice");
String s = names.get(0);

Map<String, Integer> counts = new HashMap<>();
counts.put("A", 1);
Integer n = counts.get("A");
```

(5) **Gebundene Typ-Parameter (Bounded Type Parameters).** Man kann Generics einschränken, um Typenbeschränkungen festzulegen, z. B. damit nur Zahlen benutzt werden dürfen:

```
public class Box<T extends Number> {
    private T value;
    public Box(T value) { this.value = value; }
    public T get() { return value; }
}
```

(6) **Wildcards und PECS-Prinzip.** Wildcard-Typen ermöglichen flexible Typenverwendung:

```
List<? extends Number> nums = new ArrayList<Integer>();
for (Number n : nums) System.out.println(n);
```

PECS:

- Producer extends: Lies von einer Struktur ab, z. B. `List<? extends T>`.
- Consumer super: Schreibe in eine Struktur, z. B. `List<? super T>`.

(7) Diamond Operator und Typinferenz. Seit Java 7 kann der RHS oft leer bleiben:

```
List<String> list = new ArrayList<>();  
Map<String, Integer> map = new HashMap<>();
```

(8) Einschränkungen und Best Practices.

- Primitive Typen können nicht direkt als Typparameter verwendet werden (z. B. `List<int>` ist verboten).
- Typ-Parameter existieren zur Compile-Zeit; zur Laufzeit wird Typ-Erasure verwendet.
- Arrays vs. Generics: Arrays kennen Typen, Generics arbeiten mit Typ-Parametern; dies hat Auswirkungen z. B. auf Casting.
- Generics sinnvoll einsetzen, um Typsicherheit und API-Wiederverwendung zu erhöhen.

(9) Übungen zum Üben.

```
public class Pair<K,V> {  
    private K key;  
    private V value;  
    public Pair(K k, V v){ key = k; value = v; }  
    public K getKey(){ return key; }  
    public V getValue(){ return value; }  
}
```

(10) Fazit. Generische Typen ermöglichen abstrakteres Design von Datenstrukturen und Algorithmen in Java. Sie erhöhen Typsicherheit, Reuse und Klarheit des Codes, insbesondere in Sammlungen, Methoden und API-Design.