Kurzübersicht: Generische Datentypen in Java

Was sind generische Datentypen?

Generics ermoeglichen es dir, Klassen, Interfaces und Methoden mit Typ-Parametern zu schreiben.

Beispiel:

List<String> namen = new ArrayList<>();

Ohne Generics:

List namen = new ArrayList(); // unsicher, kann alles enthalten

Warum braucht man Generics?

- Typsicherheit - Verhindert, dass falsche Objekte gespeichert werden.
- Keine Casts noetig - Spart laestiges und fehleranfaelliges Downcasting.
- Code-Wiederverwendung - Eine Klasse/Methode funktioniert fuer viele Typen.

Haeufige Anwendungen

List<T> -> List<Integer> zahlen = new ArrayList<>();

Map<K, V> -> Map<String, Integer> alterMap = new HashMap<>();

Eigene Klasse -> class Box<T> { T inhalt; }

Eigene generische Klasse

```
class Box<T> {
    private T inhalt;

    public void set(T inhalt) { this.inhalt = inhalt; }
    public T get() { return inhalt; }
}
```

Verwendung:

Box<String> box = new Box<>();

box.set("Hallo");

Kurzübersicht: Generische Datentypen in Java

System.out.println(box.get()); // Gibt: Hallo

Wildcards (?)

List<?> -> Liste mit beliebigem Typ

List<? extends Number> -> Liste mit Unterklassen von Number (Integer, Double, ...)

List<? super Integer> -> Liste mit Integer oder Oberklassen (Number, Object)

Merksatz

Generics = mehr Sicherheit + weniger Casts + wiederverwendbarer Code