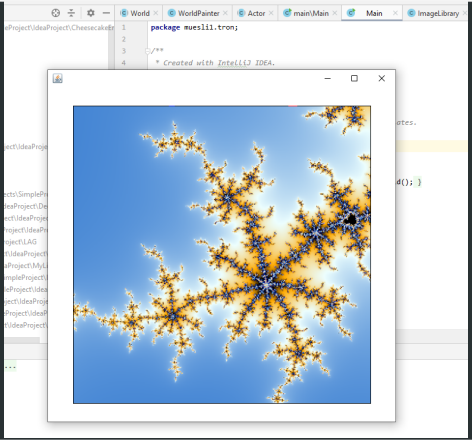
FOP Recap #9

Generics



Das steht heute auf dem Plan

TECHNISCHE UNIVERSITÄT DARMSTADT

- Mehrdimensionale Arrays
  - Initialisierung
  - Zugriff auf Elemente
  - Beispiel
- Generics-Grundlagen
- Diamond operator
- Generische Methoden
- Eingeschränkte Typparameter
- Erben von generischen Klassen
- Erstellen von generischen Arrays
- Wildcards
- Type Erasure

Mehrdimensionale Arrays

TECHNISCHE UNIVERSITÄT DARMSTADT

- Haben als Komponentypen wieder Arrays
- Typischerweise 2- oder 3-dimensionale Arrays
- Zugriff über mehrere Indizes
- Wichtig: Subarrays müssen nicht die gleiche Länge haben
- Zusätzliche Kniffe fürs Erstellen

Mehrdimensionale Arrays

Initialisierung

TECHNISCHE UNIVERSITÄT DARMSTADT

```java
int[][] bigArray = new int[2][5];
```

```java
int[][] bigArray = new int[2][];
bigArray[0] = new int[5];
bigArray[1] = new int[5];
```

```java
int[][] bigArray = new int[2][];
bigArray[0] = new int[] {0, 0, 0, 0, 0};
bigArray[1] = new int[] {0, 0, 0, 0, 0};
```

```java
int[][] bigArray = new int[][] {{0, 0, 0, 0, 0}, {0, 0, 0, 0, 0}};
```

Mehrdimensionale Arrays

Zugriff auf Elemente

TECHNISCHE UNIVERSITÄT DARMSTADT

```java
int[][] bigArray = new int[2][5];
```

```java
int[] smallArray = bigArray[0];
smallArray[2] = 3;
```

```java
bigArray[0][2] = 3;
```

Mehrdimensionale Arrays

Zugriff auf Elemente

TECHNISCHE UNIVERSITÄT DARMSTADT

```java
int[][] bigArray = new int[2][5];
```

```java
int[] smallArray = bigArray[0];
int value = smallArray[2];
```

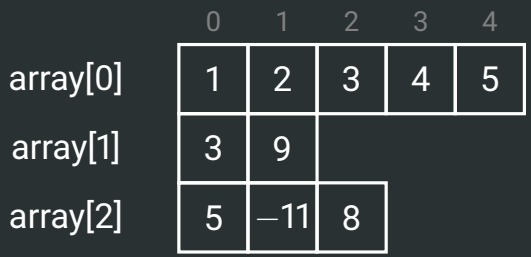
```java
int value = bigArray[0][2];
```

Mehrdimensionale Arrays

Beispiel

TECHNISCHE UNIVERSITÄT DARMSTADT

```java
int[][] array = new int[3][];
array[0] = new int[] {1, 2, 3, 4, 5};
array[1] = new int[] {3, 9};
array[2] = new int[] {5, -11, 8};
```



Mehrdimensionale Arrays

Beispiel

TECHNISCHE UNIVERSITÄT DARMSTADT

```java
int[][] array = ....; // von vorheriger slide übernommen
array[0][0] = 10;
array[2][1] = -10;
```

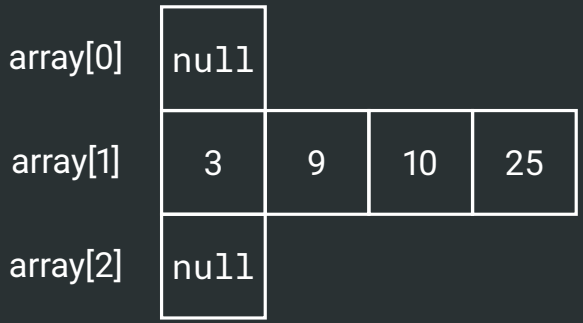
	0	1	2	3	4
array[0]	10	2	3	4	5
array[1]	3	9			
array[2]	5	−10	8		

Mehrdimensionale Arrays

Beispiel

TECHNISCHE UNIVERSITÄT DARMSTADT

```java
int[][] array = new int[3][];
array[1] = new int[] {3, 9, 10, 25};
```



Das steht heute auf dem Plan

TECHNISCHE UNIVERSITÄT DARMSTADT

- Mehrdimensionale Arrays
- Generics-Grundlagen
  - Instanziierung
  - Nutzen von Typparameter
  - Konventionen für Typparameter-Bennenung
  - Mehrere Typparameter
- Diamond operator
- Generische Methoden
- Eingeschränkte Typparameter
- Erben von generischen Klassen
- Erstellen von generischen Arrays
- Wildcards
- Type Erasure

Generics-Grundlagen

TECHNISCHE UNIVERSITÄT DARMSTADT

ohne Generics

```
1  public class StringHolder {
2      public String value;
3  }
4  public class IntegerHolder {
5      public Integer value;
6  }
```

mit Generics

```
1  public class Holder<T> {
2      public T value;
3  }
```

Generics-Grundlagen

TECHNISCHE UNIVERSITÄT DARMSTADT

ohne Generics

```
1  public class StringHolder {
2      public String value;
3  }
4  public class IntegerHolder {
5      public Integer value;
6  }
```

mit Generics

```
1  public class Holder<T> {
2      public T value;
3  }
```

Vorteile von Generics

- Lösung mit Generics kürzer

Generics-Grundlagen

TECHNISCHE UNIVERSITÄT DARMSTADT

ohne Generics

```
1  public class StringHolder {
2      public String value;
3  }
4  public class IntegerHolder {
5      public Integer value;
6  }
```

mit Generics

```
1  public class Holder<T> {
2      public T value;
3  }
```

Vorteile von Generics

- Lösung mit Generics kürzer
- Vermeidet Redundanz ⇒ Fehler vermeiden, Wartbarkeit verbessern

Generics-Grundlagen

Instanziierung

TECHNISCHE UNIVERSITÄT DARMSTADT

```java
public class Holder<T> {
    public T value;
}
```

Generics-Grundlagen

Instanziierung

TECHNISCHE UNIVERSITÄT DARMSTADT

```java
public class Holder<T> {
    public T value;
}
```

```java
Holder<Integer> integerHolder = new Holder<Integer>();
Holder<Boolean> booleanHolder = new Holder<Boolean>();
```

Generics-Grundlagen

Nutzen von Typparameter

TECHNISCHE UNIVERSITÄT DARMSTADT

```java
public class Holder<T> {
    public T value;

    public T getValue() {
        return value;
    }
    public void killItWithFire() {
        value = null;
    }
}
```

Generics-Grundlagen

Konventionen für Typparameter-Bennenung

TECHNISCHE UNIVERSITÄT DARMSTADT

- Ein Großbuchstabe
- T für Type
- K für Key
- V für Value
- E für Element
- ....

Generics-Grundlagen

Mehrere Typparameter

TECHNISCHE UNIVERSITÄT DARMSTADT

ohne Generics

```
1  public class
   ↪  StringAndDoubleHolder {
2      public String valueA;
3      public Double valueB;
4  }
5  public class
   ↪  IntegerAndShortHolder {
6      public Integer valueA;
7      public Short valueB;
8  }
```

Generics-Grundlagen

Mehrere Typparameter

TECHNISCHE UNIVERSITÄT DARMSTADT

ohne Generics

```
1  public class
   ↪  StringAndDoubleHolder {
2      public String valueA;
3      public Double valueB;
4  }
5  public class
   ↪  IntegerAndShortHolder {
6      public Integer valueA;
7      public Short valueB;
8  }
```

mit Generics

```
1  public class Tuple<A, B> {
2      public A valueA;
3      public B valueB;
4  }
```

Das steht heute auf dem Plan

- Mehrdimensionale Arrays
- Generics-Grundlagen
- Diamond operator
- Generische Methoden
- Eingeschränkte Typparameter
- Erben von generischen Klassen
- Erstellen von generischen Arrays
- Wildcards
- Type Erasure

Diamond operator

TECHNISCHE UNIVERSITÄT DARMSTADT

Diamond Operator

```
1  ArrayList<String> myList1 = new ArrayList<String>();
2  ArrayList<String> myList2 = new ArrayList<>(); // String wird impliziert
3  List<String> myList3 = new Arraylist<>();
```

Diamond operator

TECHNISCHE UNIVERSITÄT DARMSTADT

Diamond Operator

```
1  ArrayList<String> myList1 = new ArrayList<String>();
2  ArrayList<String> myList2 = new ArrayList<>(); // String wird impliziert
3  List<String> myList3 = new Arraylist<>();
4
5  Tuple<String, Integer> myTuple1 = new Tuple<>();
6  Tuple<Tuple<String, Integer>, Tuple<List<String>, Integer>> myTuple1
7      = new Tuple<>();
```

Das steht heute auf dem Plan

TECHNISCHE UNIVERSITÄT DARMSTADT

- Mehrdimensionale Arrays
- Generics-Grundlagen
- Diamond operator
- Generische Methoden
- Eingeschränkte Typparameter
- Erben von generischen Klassen
- Erstellen von generischen Arrays
- Wildcards
- Type Erasure

Generische Methoden

TECHNISCHE UNIVERSITÄT DARMSTADT

```java
public List<Double> nonGenericFilter(List<Double> list,
    Predicate<Double> predicate) {
    // ....
}
public List<Boolean> nonGenericFilter(List<Boolean> list,
    Predicate<Boolean> predicate) {
    // ....
}
```

Generische Methoden

TECHNISCHE UNIVERSITÄT DARMSTADT

```java
public List<Double> nonGenericFilter(List<Double> list,
    Predicate<Double> predicate) {
    // ....
}
public List<Boolean> nonGenericFilter(List<Boolean> list,
    Predicate<Boolean> predicate) {
    // ....
}
```

```java
public <T> List<T> filter(List<T> list, Predicate<T> predicate) {
    // ....
}
```

Das steht heute auf dem Plan

TECHNISCHE UNIVERSITÄT DARMSTADT

Mehrdimensionale Arrays

Generics-Grundlagen

Diamond operator

Generische Methoden

Eingeschränkte Typparameter

- `extends` Klassen
- `extends` Interfaces
- `extends` Klassen und Interfaces

Erben von generischen Klassen

Erstellen von generischen Arrays

Wildcards

Type Erasure

Eingeschränkte Typparameter

`extends` Klassen

TECHNISCHE UNIVERSITÄT DARMSTADT

```java
public class Holder<T> {
    public T value;
    public void test() {
        int i = // ...?
    }
}
```

Eingeschränkte Typparameter

`extends` Klassen

TECHNISCHE UNIVERSITÄT DARMSTADT

```java
public class Holder<T extends Number> {
    public T value;
    public void test() {
        int i = value.intValue();
    }
}
```

Eingeschränkte Typparameter

`extends` Klassen

TECHNISCHE UNIVERSITÄT DARMSTADT

```java
public class Holder<T extends Number> {
    public T value;
    public void test() {
        int i = value.intValue();
    }
}
```

```java
Holder<Integer> holder1 = new Holder<Integer>();
Holder<Boolean> holder2 = new Holder<Boolean>(); // !!
```

Eingeschränkte Typparameter

`extends` Interfaces

TECHNISCHE UNIVERSITÄT DARMSTADT

```java
public class Holder<T extends Inf1 & Inf2 & Inf3> {
    public T value;
    // ...
}
```

Eingeschränkte Typparameter

`extends` Klassen und Interfaces

TECHNISCHE UNIVERSITÄT DARMSTADT

```java
public class Holder<T extends Number & Inf1 & Inf2 & Inf3> {
    public T value;
    // ...
}
```

Das steht heute auf dem Plan

TECHNISCHE UNIVERSITÄT DARMSTADT

- Mehrdimensionale Arrays
- Generics-Grundlagen
- Diamond operator
- Generische Methoden
- Eingeschränkte Typparameter
- Erben von generischen Klassen
- Erstellen von generischen Arrays
- Wildcards
- Type Erasure

Erben von generischen Klassen

TECHNISCHE UNIVERSITÄT DARMSTADT

```
1  public class MyGenericClass<T> {
2      // ...
3  }
```

Erben von generischen Klassen

TECHNISCHE UNIVERSITÄT DARMSTADT

```java
public class MyGenericClass<T> {
    // ...
}
```

```java
public class Sub<Q> extends MyGenericClass<Q> {
    // ...
}
```

Erben von generischen Klassen

TECHNISCHE UNIVERSITÄT DARMSTADT

```java
public class MyGenericClass<T> {
    // ...
}
```

```java
public class Sub<Q> extends MyGenericClass<Q> {
    // ...
}
```

```java
public class IntegerSub extends MyGenericClass<Integer> {
    // ...
}
```

Erben von generischen Klassen

TECHNISCHE UNIVERSITÄT DARMSTADT

```java
public class MyGenericClass<T extends Integer> {
    // ...
}
```

Erben von generischen Klassen

TECHNISCHE UNIVERSITÄT DARMSTADT

```java
public class MyGenericClass<T extends Integer> {
    // ...
}
```

```java
public class IntegerSub extends MyGenericClass<Integer> {
    // ...
}
```

Erben von generischen Klassen

TECHNISCHE UNIVERSITÄT DARMSTADT

```java
public class MyGenericClass<T extends Integer> {
    // ...
}
```

```java
public class IntegerSub extends MyGenericClass<Integer> {
    // ...
}
```

```java
public class BooleanSub extends MyGenericClass<Boolean> { // !!
// ...
}
```

Das steht heute auf dem Plan

TECHNISCHE UNIVERSITÄT DARMSTADT

- Mehrdimensionale Arrays
- Generics-Grundlagen
- Diamond operator
- Generische Methoden
- Eingeschränkte Typparameter
- Erben von generischen Klassen
- Erstellen von generischen Arrays
- Wildcards
- Type Erasure

Erstellen von generischen Arrays

TECHNISCHE UNIVERSITÄT DARMSTADT

```java
public class MyGenericClass<T> {
    public T[] createMyArray() {
        T[] testArray = (T[]) new Object[25];
        return testArray;
    }
}
```

Erstellen von generischen Arrays

TECHNISCHE UNIVERSITÄT DARMSTADT

```java
public class MyGenericClass<T> {
    public T[] createMyArray() {
        T[] testArray = (T[]) new Object[25];
        return testArray;
    }
}
```

```java
public <Q> Q[] createMyArray() {
    Q[] testArray = (Q[]) new Object[25];
    return testArray;
}
```

Das steht heute auf dem Plan

TECHNISCHE UNIVERSITÄT DARMSTADT

- Mehrdimensionale Arrays
- Generics-Grundlagen
- Diamond operator
- Generische Methoden
- Eingeschränkte Typparameter
- Erben von generischen Klassen
- Erstellen von generischen Arrays
- Wildcards
  - Grundlagen
  - Wann werden Wildcards verwendet?
  - ? Wildcards mit `super`
  - Kovarianz - Parameter
  - Kovarianz - Rückgabetyp

Wildcards

Grundlagen

TECHNISCHE UNIVERSITÄT DARMSTADT

```java
public <T> void printList(List<T> list) {
    for (T item : list) {
        System.out.println(item);
    }
}
```

Wildcards

Grundlagen

TECHNISCHE UNIVERSITÄT DARMSTADT

```java
public <T> void printList(List<T> list) {
    for (T item : list) {
        System.out.println(item);
    }
}
```

```java
public void printList(List<?> list) {
    for (Object item : list) {
        System.out.println(item);
    }
}
```

Wildcards

Grundlagen

TECHNISCHE UNIVERSITÄT DARMSTADT

```java
public <T extends Number> void printList(List<T> list) {
    for (T item : list) {
        System.out.println(item.intValue() + 1);
    }
}
```

Wildcards

Grundlagen

TECHNISCHE UNIVERSITÄT DARMSTADT

```java
public <T extends Number> void printList(List<T> list) {
    for (T item : list) {
        System.out.println(item.intValue() + 1);
    }
}
```

```java
public void printList(List<? extends Number> list) {
    for (Number item : list) {
        System.out.println(item.intValue() + 1);
    }
}
```

Wildcards

Wann werden Wildcards verwendet?

TECHNISCHE UNIVERSITÄT DARMSTADT

```
1  public void nonGenMethod(Holder<? extends Number> a, Holder<?> b) {
2      // ...
3  }
```

Wildcards

Wann werden Wildcards verwendet?

TECHNISCHE UNIVERSITÄT DARMSTADT

```java
public void nonGenMethod(Holder<? extends Number> a, Holder<?> b) {
    // ...
}
```

```java
public <T extends Number> void genMethodA(Holder<T> a, Holder<T> b) {
    // ...
}
```

Wildcards

Wann werden Wildcards verwendet?

TECHNISCHE UNIVERSITÄT DARMSTADT

```java
public void nonGenMethod(Holder<? extends Number> a, Holder<?> b) {
    // ...
}
```

```java
public <T extends Number> void genMethodA(Holder<T> a, Holder<T> b) {
    // ...
}
```

```java
public <T extends Number, Q> void genMethodB(Holder<T> a, Holder<Q> b) {
    // ...
}
```

Wildcards

? Wildcards mit super

TECHNISCHE UNIVERSITÄT DARMSTADT

```
1  public void nonGenMethod(Predicate<? super Number> a) {
2      a.intValue(); // !! Nicht erlaubt
3  }
```

Wildcards

? Wildcards mit super

TECHNISCHE UNIVERSITÄT DARMSTADT

```java
public void nonGenMethod(Predicate<? super Number> a) {
    a.intValue(); // !! Nicht erlaubt
}
```

```java
Predicate<? super Number> a = (Object x) -> true;
Predicate<? super Number> b = (Serializable x) -> true;
Predicate<? super Number> c = (Number x) -> true;
```

Wildcards

? Wildcards mit `super`

TECHNISCHE UNIVERSITÄT DARMSTADT

Generic Variance

- ? `extends` T - Kovarianz (Lesender Zugriff)
- ? `super` T - Kontravarianz (Schreibender Zugriff)
- T - Invarianz (Beides)

Wildcards

Kovarianz - Parameter

TECHNISCHE UNIVERSITÄT DARMSTADT

```
1  public class A {}
2  public class B extends A {}
```

Wildcards

Kovarianz - Parameter

TECHNISCHE UNIVERSITÄT DARMSTADT

```java
public class A {}
public class B extends A {}
```

```java
public void foo(A a) { ... }
```

Wildcards

Kovarianz - Parameter

TECHNISCHE UNIVERSITÄT DARMSTADT

```java
public class A {}
public class B extends A {}
```

```java
public void foo(A a) { ... }
```

```java
foo(new A());
foo(new B()); // parameter covariance
```

Wildcards

Kovarianz - Rückgabetyp

TECHNISCHE UNIVERSITÄT DARMSTADT

```java
public class A {}
public class B extends A {}
```

Wildcards

Kovarianz - Rückgabetyp

TECHNISCHE UNIVERSITÄT DARMSTADT

```java
public class A {}
public class B extends A {}
```

```java
public class Baz {
    public A foo() {...}
}
public class SpecialBaz extends Baz {
    @Override
    public B foo() {...} // return type covariance
}
```

Wildcards

Kovarianz - Arrays

TECHNISCHE UNIVERSITÄT DARMSTADT

```
1  public class A {}
2  public class B extends A {}
```

Wildcards

Kovarianz - Arrays

TECHNISCHE UNIVERSITÄT DARMSTADT

```java
public class A {}
public class B extends A {}
```

```java
A[] myArray = new B[2]; // array covariance
```

Wildcards

Kovarianz - Arrays

TECHNISCHE UNIVERSITÄT DARMSTADT

```java
public class A {}
public class B extends A {}
```

```java
A[] myArray = new B[2]; // array covariance
```

```java
A[] myArray = new A[2];
myArray[0] = new B(); // array covariance
```

Wildcards

Kovarianz - Arrays

TECHNISCHE UNIVERSITÄT DARMSTADT

```java
public class A {}
public class B extends A {}
public class C extends A {}
```

Wildcards

Kovarianz - Arrays

TECHNISCHE UNIVERSITÄT DARMSTADT

```java
public class A {}
public class B extends A {}
public class C extends A {}
```

```java
A[] myArray = new B[2];
myArray[0] = new B();
```

Wildcards

Kovarianz - Arrays

TECHNISCHE UNIVERSITÄT DARMSTADT

```java
public class A {}
public class B extends A {}
public class C extends A {}
```

```java
A[] myArray = new B[2];
myArray[0] = new B();
```

```java
A[] myArray = new B[2];
myArray[0] = new C(); // !! array covariance
```

Wildcards

Invarianz

TECHNISCHE UNIVERSITÄT DARMSTADT

```
1  public class A {}
2  public class B extends A {}
```

Wildcards

Invarianz

TECHNISCHE UNIVERSITÄT DARMSTADT

```java
public class A {}
public class B extends A {}
```

```java
List<B> myList = new ArrayList<B>();
List<A> otherList = myList; // !!

List<? extends A> list = myList;
```

Wildcards

Invarianz

TECHNISCHE UNIVERSITÄT DARMSTADT

```java
public class A {}
public class B extends A {}
```

```java
List<B> myList = new ArrayList<B>();
List<A> otherList = myList; // !!

List<? extends A> list = myList;
list.add(new A()); // !! compile error
list.add(new B()); // !! compile error
```

Wildcards

Invarianz

TECHNISCHE UNIVERSITÄT DARMSTADT

```java
public void readFromList(List<? extends Number> list) {
    // z.B: List<Double>,List<Integer>,List<Number>
    Number a = list.get(0);
}
public void readFromList(List<? super Number> list) {
    // z.B: List<Number>,List<Object>
    Number a = list.get(0); // !!
}
```

Wildcards

Wann super, wann extends?

TECHNISCHE UNIVERSITÄT DARMSTADT

```java
public void writeToList(List<? extends Number> list) {
    // z.B: List<Double>,List<Integer>,List<Number>
    list.add(Double.valueOf(2.0)); // !!
}
public void readFromList(List<? super Number> list) {
    // z.B: List<Number>,List<Object>
    list.add(Double.valueOf(2.0));
}
```

Wildcards

Wann super, wann extends?

TECHNISCHE UNIVERSITÄT DARMSTADT

```java
public void both(List<? extends Number> list) {
    // z.B: List<Double>,List<Integer>,List<Number>
    Number a = list.get(0);
    list.add(Double.valueOf(2.0)); // !!
}
public void readFromList(List<? super Number> list) {
    // z.B: List<Number>,List<Object>
    Number a = list.get(0); // !!
    list.add(Double.valueOf(2.0));
}
```

Wildcards

Wann super, wann extends?

TECHNISCHE UNIVERSITÄT DARMSTADT

```java
public void both(List<Double> list) {
    // nur List<Double>
    Number a = list.get(0);
    list.add(Double.valueOf(2.0));
}
```

Wildcards

Wann `super`, wann `extends`? Beispiele

TECHNISCHE UNIVERSITÄT DARMSTADT

- Supplier: In-Parameter
- Consumer: Out-Parameter
- Function: Beides

Wildcards

Wann `super`, wann `extends`? Beispiele


```java
public interface Supplier<T> {
    T get();
}
```

Wildcards

Wann super, wann extends? Beispiele

TECHNISCHE UNIVERSITÄT DARMSTADT

? super T (Kontravarianz)

```
1  public void foo(Supplier<? super T> supplier) {
2      T a = supplier.get(); // !!
3  }
```

Wildcards

Wann super, wann extends? Beispiele

TECHNISCHE UNIVERSITÄT DARMSTADT

T (Invarianz)

```
1  public void foo(List<T> list, Supplier<T> supplier) {
2      list.add(supplier.get());
3  }
4  public void bar() {
5      List<Number> list = new ArrayList<>();
6      Supplier<Integer> supplier = () -> 1;
7      foo(list, supplier); // !!
8  }
```

Wildcards

Wann super, wann extends? Beispiele

TECHNISCHE UNIVERSITÄT DARMSTADT

? extends T (Kovarianz)

```java
public void foo(List<T> list, Supplier<? extends T> supplier) {
    list.add(supplier.get());
}
public void bar() {
    List<Number> list = new ArrayList<>();
    Supplier<Integer> supplier = () -> 1;
    foo(list, supplier); // geht jetzt
}
```

Wildcards

Wann `super`, wann `extends`? Beispiele

TECHNISCHE UNIVERSITÄT DARMSTADT

```java
public interface Consumer<T> {
    void accept(T t);
}
```

Wildcards

Wann super, wann extends? Beispiele

TECHNISCHE UNIVERSITÄT DARMSTADT

? extends T (Kovarianz)

```
1  public void foo(Consumer<? extends Number> consumer) {
2      consumer.accept(42); // !!
3  }
```

Wildcards

Wann super, wann extends? Beispiele

TECHNISCHE UNIVERSITÄT DARMSTADT

T (Invarianz)

```
1  public void foo(List<T> list, Consumer<T> consumer) {
2      consumer.accept(list.get(0));
3  }
4  public void bar() {
5      List<Integer> list = new ArrayList<>();
6      Consumer<Number> consumer = n -> System.out.println(n.intValue());
7      foo(list, consumer); // !!
8  }
```

Wildcards

Wann super, wann extends? Beispiele

TECHNISCHE UNIVERSITÄT DARMSTADT

? super T (Kontravarianz)

```
1  public void foo(List<T> list, Consumer<? super T> consumer) {
2      consumer.accept(list.get(0));
3  }
4  public void bar() {
5      List<Integer> list = new ArrayList<>();
6      Consumer<Number> consumer = n -> System.out.println(n.intValue());
7      foo(list, consumer); // geht jetzt
8  }
```

Wildcards

Wann super, wann extends? Beispiele

TECHNISCHE UNIVERSITÄT DARMSTADT

```java
public interface Function<T, R> {
    R apply(T t);
}
```

Wildcards

Wann super, wann extends? Beispiele

TECHNISCHE UNIVERSITÄT DARMSTADT

- `Supplier` → `Supplier<? extends T>`
- `Consumer` → `Consumer<? super T>`
- `Function` → `Function<? super T, ? extends R>`

Das steht heute auf dem Plan

TECHNISCHE UNIVERSITÄT DARMSTADT

Mehrdimensionale Arrays
Generics-Grundlagen
Diamond operator
Generische Methoden
Eingeschränkte Typparameter
Erben von generischen Klassen
Erstellen von generischen Arrays
Wildcards
Type Erasure

Type Erasure

TECHNISCHE UNIVERSITÄT DARMSTADT

- Typparameter sind zur Laufzeit nicht bekannt!
- Im Prinzip nur Schutz vor Fehlern beim Kompilieren

Type Erasure

TECHNISCHE UNIVERSITÄT DARMSTADT

```java
public class Test<T extends Number> {
    T val;
    public void test(T p) {
    }
}
```

Type Erasure

TECHNISCHE UNIVERSITÄT DARMSTADT

```java
public class Test<T extends Number> {
    T val;
    public void test(T p) {
    }
}
```

```java
public class Test<Number> {
    Number val;
    public void test(Number p) {
    }
}
```

Type Erasure

TECHNISCHE UNIVERSITÄT DARMSTADT

```
1  public class Test<T> {
2      T val;
3      public void test(T p) {
4      }
5  }
```

Type Erasure

TECHNISCHE UNIVERSITÄT DARMSTADT

```java
public class Test<T> {
    T val;
    public void test(T p) {
    }
}
```

```java
public class Test<Object> {
    Object val;
    public void test(Object p) {
    }
}
```

Live-Coding!