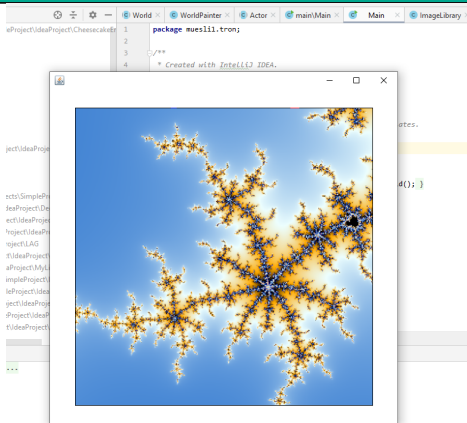


FOP Recap #10



TECHNISCHE
UNIVERSITÄT
DARMSTADT

Java Generics





Guten Tag!

Das steht heute auf dem Plan



TECHNISCHE
UNIVERSITÄT
DARMSTADT

Generics Basiskonzept

- Instanziierung

- Nutzen von Typparameter

- Benennung von Typparameter

- Mehrere Typparameter

Diamond operator

Generische Methoden

Statische generische Methoden

Eingeschränkte Typparameter

Generics!

„**Generics** – You think you understand it, and then you don't. . . “



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 }
```

</>

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

</>

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 \Rightarrow Fehler vermeiden, Wartbarkeit verbessern



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




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

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

Generics Basiskonzept

Nutzen von Typparameter



TECHNISCHE
UNIVERSITÄT
DARMSTADT

```
1 public class Holder<T> {  
2     public T value;  
3  
4     public T getValue() {  
5         return value;  
6     }  
7     public void killItWithFire() {  
8         value = null;  
9     }  
10 }
```

Generics Basiskonzept

Benennung von Typparameter



TECHNISCHE
UNIVERSITÄT
DARMSTADT

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

Generics Basiskonzept

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 Pair<A, B> {
2     public A valueA;
3     public B valueB;
4 }
```

Das steht heute auf dem Plan



TECHNISCHE
UNIVERSITÄT
DARMSTADT

Generics Basiskonzept

Diamond operator

Generische Methoden

Statische generische Methoden

Eingeschränkte Typparameter

Erben von generischen Klassen

Erstellen von generischen Arrays



Diamond Operator



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

Das steht heute auf dem Plan



TECHNISCHE
UNIVERSITÄT
DARMSTADT

Generics Basiskonzept

Diamond operator

Generische Methoden

Statische generische Methoden

Eingeschränkte Typparameter

Erben von generischen Klassen

Erstellen von generischen Arrays



```
1 public List<Double> nonGenericFilter(List<Double> list,  
   ↪ Predicate<Double> predicate) {  
2     // ....  
3 }  
4 public List<Boolean> nonGenericFilter(List<Boolean> list,  
   ↪ Predicate<Boolean> predicate) {  
5     // ....  
6 }
```




```
1 public List<Double> nonGenericFilter(List<Double> list,  
   ↪ Predicate<Double> predicate) {  
2     // ....  
3 }  
4 public List<Boolean> nonGenericFilter(List<Boolean> list,  
   ↪ Predicate<Boolean> predicate) {  
5     // ....  
6 }
```

```
1 public <T> List<T> filter(List<T> list, Predicate<T> predicate) {  
2     // ....  
3 }
```

Das steht heute auf dem Plan



TECHNISCHE
UNIVERSITÄT
DARMSTADT

Generics Basiskonzept

Diamond operator

Generische Methoden

Statische generische Methoden

Eingeschränkte Typparameter

Erben von generischen Klassen

Erstellen von generischen Arrays



```
1 public static <T> List<T> filter(List<T> list, Predicate<T> predicate) {  
2     // ....  
3 }
```

Das steht heute auf dem Plan



TECHNISCHE
UNIVERSITÄT
DARMSTADT

Generics Basiskonzept

Diamond operator

Generische Methoden

Statische generische Methoden

Eingeschränkte Typparameter

- extends Klassen

- extends Interfaces

- extends Klassen und Interfaces

Eingeschränkte Typparameter

extends Klassen



TECHNISCHE
UNIVERSITÄT
DARMSTADT

```
1 public class Holder<T> {  
2     public T value;  
3     public void test() {  
4         int i = // ...?  
5     }  
6 }
```

Eingeschränkte Typparameter

extends Klassen



TECHNISCHE
UNIVERSITÄT
DARMSTADT

```
1 public class Holder<T extends Number> {  
2     public T value;  
3     public void test() {  
4         int i = value.intValue();  
5     }  
6 }
```

Eingeschränkte Typparameter

extends Klassen



TECHNISCHE
UNIVERSITÄT
DARMSTADT

```
1 public class Holder<T extends Number> {  
2     public T value;  
3     public void test() {  
4         int i = value.intValue();  
5     }  
6 }
```

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

Eingeschränkte Typparameter

extends Interfaces



TECHNISCHE
UNIVERSITÄT
DARMSTADT

```
1 public class Holder<T extends Inf1 & Inf2 & Inf3> {  
2     public T value;  
3     // ...  
4 }
```


Eingeschränkte Typparameter

extends Klassen und Interfaces



TECHNISCHE
UNIVERSITÄT
DARMSTADT

```
1 public class Holder<T extends Number & Inf1 & Inf2 & Inf3> {  
2     public T value;  
3     // ...  
4 }
```

Das steht heute auf dem Plan



TECHNISCHE
UNIVERSITÄT
DARMSTADT

Generics Basiskonzept

Diamond operator

Generische Methoden

Statische generische Methoden

Eingeschränkte Typparameter

Erben von generischen Klassen

Erstellen von generischen Arrays



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

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

```
1 public class Sub<Q> extends MyGenericClass<Q> {  
2     // ...  
3 }
```

Erben von generischen Klassen



TECHNISCHE
UNIVERSITÄT
DARMSTADT

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

```
1 public class Sub<Q> extends MyGenericClass<Q> {  
2     // ...  
3 }
```

```
1 public class IntegerSub extends MyGenericClass<Integer> {  
2     // ...  
3 }
```



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

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

```
1 public class IntegerSub extends MyGenericClass<Integer> {  
2     // ...  
3 }
```

Erben von generischen Klassen

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

```
1 public class IntegerSub extends MyGenericClass<Integer> {  
2     // ...  
3 }
```

```
1 public class BooleanSub extends MyGenericClass<Boolean> { // !!  
2     // ...  
3 }
```



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



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

```
1 public class IntegerSub extends MyGenericClass<Integer> {  
2     // ...  
3 }
```

Erben von generischen Klassen



TECHNISCHE
UNIVERSITÄT
DARMSTADT

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

```
1 public class IntegerSub extends MyGenericClass<Integer> {  
2     // ...  
3 }
```

```
1 public class BooleanSub extends MyGenericClass<Boolean> { // !!  
2     // ...  
3 }
```

Das steht heute auf dem Plan



TECHNISCHE
UNIVERSITÄT
DARMSTADT

Generics Basiskonzept

Diamond operator

Generische Methoden

Statische generische Methoden

Eingeschränkte Typparameter

Erben von generischen Klassen

Erstellen von generischen Arrays

Erstellen von generischen Arrays

```
1 public class MyGenericClass<T> {  
2     public T[] createMyArray() {  
3         T[] testArray = (T[]) new Object[25];  
4         return testArray;  
5     }  
6 }
```

Erstellen von generischen Arrays



```
1 public class MyGenericClass<T> {  
2     public T[] createMyArray() {  
3         T[] testArray = (T[]) new Object[25];  
4         return testArray;  
5     }  
6 }
```

```
1 public <Q> Q[] createMyArray() {  
2     Q[] testArray = (Q[]) new Object[25];  
3     return testArray;  
4 }
```

Das steht heute auf dem Plan



TECHNISCHE
UNIVERSITÄT
DARMSTADT

Generics Basiskonzept

Diamond operator

Generische Methoden

Statische generische Methoden

Eingeschränkte Typparameter

Erben von generischen Klassen

Erstellen von generischen Arrays

Generic Variance

? Wildcards



TECHNISCHE
UNIVERSITÄT
DARMSTADT

```
1 public <T> void printList(List<T> list) {  
2     for (T item : list) {  
3         System.out.println(item);  
4     }  
5 }
```


Generic Variance

? Wildcards



```
1 public <T> void printList(List<T> list) {  
2     for (T item : list) {  
3         System.out.println(item);  
4     }  
5 }
```

```
1 public void printList(List<?> list) {  
2     for (Object item : list) {  
3         System.out.println(item);  
4     }  
5 }
```

Generic Variance

? Wildcards



```
1 public <T extends Number> void printList(List<T> list) {  
2     for (T item : list) {  
3         System.out.println(item.intValue() + 1);  
4     }  
5 }
```

Generic Variance

? Wildcards



```
1 public <T extends Number> void printList(List<T> list) {  
2     for (T item : list) {  
3         System.out.println(item.intValue() + 1);  
4     }  
5 }
```

```
1 public void printList(List<? extends Number> list) {  
2     for (Number item : list) {  
3         System.out.println(item.intValue() + 1);  
4     }  
5 }
```

Generic Variance

Wann werden wildcards verwendet?



TECHNISCHE
UNIVERSITÄT
DARMSTADT

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

Generic Variance

Wann werden wildcards verwendet?



TECHNISCHE
UNIVERSITÄT
DARMSTADT

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

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

Generic Variance

Wann werden wildcards verwendet?



TECHNISCHE
UNIVERSITÄT
DARMSTADT

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

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

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

Generic Variance

? Wildcards mit super



TECHNISCHE
UNIVERSITÄT
DARMSTADT

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

Generic Variance

? Wildcards mit super



TECHNISCHE
UNIVERSITÄT
DARMSTADT

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

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


Generic Variance

Wann super, wann extends?



TECHNISCHE
UNIVERSITÄT
DARMSTADT

Producers Extend Consumers Super
(PECS)

Generic Variance

Wann super, wann extends?



TECHNISCHE
UNIVERSITÄT
DARMSTADT

Generic Variance

- ? extends T - Covariance (Lesender Zugriff)
- ? super T - Contravariance (Schreibender Zugriff)
- T - Invariance (Beides)

Generic Variance

Covariance - Parameters



TECHNISCHE
UNIVERSITÄT
DARMSTADT

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

Generic Variance

Covariance - Parameters



TECHNISCHE
UNIVERSITÄT
DARMSTADT

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

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

Generic Variance

Covariance - Parameters



TECHNISCHE
UNIVERSITÄT
DARMSTADT

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

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

```
1 foo(new A());  
2 foo(new B()); // parameter covariance
```

Generic Variance

Covariance - Return type



TECHNISCHE
UNIVERSITÄT
DARMSTADT

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

Generic Variance

Covariance - Return type



TECHNISCHE
UNIVERSITÄT
DARMSTADT

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

```
1 public class Baz {  
2     public A foo() {...}  
3 }  
4 public class SpecialBaz extends Baz {  
5     @Override  
6     public B foo() {...} // return type covariance  
7 }
```

Generic Variance

Covariance - Arrays



TECHNISCHE
UNIVERSITÄT
DARMSTADT

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


Generic Variance

Covariance - Arrays



TECHNISCHE
UNIVERSITÄT
DARMSTADT

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

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

Generic Variance

Covariance - Arrays



TECHNISCHE
UNIVERSITÄT
DARMSTADT

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

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

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

Generic Variance

Covariance - Arrays



TECHNISCHE
UNIVERSITÄT
DARMSTADT

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

Generic Variance

Covariance - Arrays



TECHNISCHE
UNIVERSITÄT
DARMSTADT

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

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

Generic Variance

Covariance - Arrays



TECHNISCHE
UNIVERSITÄT
DARMSTADT

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

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

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

Generic Variance

Invariance



TECHNISCHE
UNIVERSITÄT
DARMSTADT

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

Generic Variance

Invariance



TECHNISCHE
UNIVERSITÄT
DARMSTADT

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

```
1 List<B> myList = new ArrayList<B>();  
2 List<A> otherList = myList; // !!  
3  
4 List<? extends A> list = myList;
```

Generic Variance

Invariance



TECHNISCHE
UNIVERSITÄT
DARMSTADT

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

```
1 List<B> myList = new ArrayList<B>();  
2 List<A> otherList = myList; // !!  
3  
4 List<? extends A> list = myList;  
5 list.add(new A()); // !! compile error  
6 list.add(new B()); // !! compile error
```


Generic Variance

Invariance



TECHNISCHE
UNIVERSITÄT
DARMSTADT

```
1 public void readFromList(List<? extends Number> list) {  
2     // z.B: List<Double>, List<Integer>, List<Number>  
3     Number a = list.get(0);  
4 }  
5 public void readFromList(List<? super Number> list) {  
6     // z.B: List<Number>, List<Object>  
7     Number a = list.get(0); // !!  
8 }
```

Generic Variance

Wann super, wann extends?



TECHNISCHE
UNIVERSITÄT
DARMSTADT

```
1 public void writeToList(List<? extends Number> list) {  
2     // z.B: List<Double>, List<Integer>, List<Number>  
3     list.add(Double.valueOf(2.0)); // !!  
4 }  
5 public void readFromList(List<? super Number> list) {  
6     // z.B: List<Number>, List<Object>  
7     list.add(Double.valueOf(2.0));  
8 }
```

Generic Variance

Wann super, wann extends?



TECHNISCHE
UNIVERSITÄT
DARMSTADT

```
1 public void both(List<? extends Number> list) {  
2     // z.B: List<Double>, List<Integer>, List<Number>  
3     Number a = list.get(0);  
4     list.add(Double.valueOf(2.0)); // !!  
5 }  
6 public void readFromList(List<? super Number> list) {  
7     // z.B: List<Number>, List<Object>  
8     Number a = list.get(0); // !!  
9     list.add(Double.valueOf(2.0));  
10 }
```

Generic Variance

Wann super, wann extends?



TECHNISCHE
UNIVERSITÄT
DARMSTADT

```
1 public void both(List<Double> list) {  
2     // nur List<Double>  
3     Number a = list.get(0);  
4     list.add(Double.valueOf(2.0));  
5 }
```

Generic Variance

Wann super, wann extends? Beispiele



TECHNISCHE
UNIVERSITÄT
DARMSTADT

■ Supplier

Generic Variance

Wann super, wann extends? Beispiele



TECHNISCHE
UNIVERSITÄT
DARMSTADT

- Supplier
- Consumer

Generic Variance

Wann super, wann extends? Beispiele



TECHNISCHE
UNIVERSITÄT
DARMSTADT

- Supplier
- Consumer
- Function

Generic Variance

Wann super, wann extends? Beispiele



TECHNISCHE
UNIVERSITÄT
DARMSTADT

```
1 public interface Supplier<T> {  
2     T get();  
3 }
```


Generic Variance

Wann *super*, wann *extends*? Beispiele



TECHNISCHE
UNIVERSITÄT
DARMSTADT

? *super* T (Contravarianz)

Generic Variance

Wann super, wann extends? Beispiele



TECHNISCHE
UNIVERSITÄT
DARMSTADT

? **super** T (Contravarianz)

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

Generic Variance

Wann super, wann extends? Beispiele



TECHNISCHE
UNIVERSITÄT
DARMSTADT

T (Invarianz)



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 }
```

Generic Variance

Wann super, wann extends? Beispiele



TECHNISCHE
UNIVERSITÄT
DARMSTADT

? **extends** T (Covarianz)



? **extends** T (Covarianz)

```
1 public void foo(List<T> list, Supplier<? extends 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); // geht jetzt  
8 }
```

Generic Variance

Wann super, wann extends? Beispiele



TECHNISCHE
UNIVERSITÄT
DARMSTADT

? **extends** T (Covarianz)



? **extends** T (Covarianz)

```
1 public void foo(List<T> list, Supplier<? extends 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); // geht jetzt  
8 }
```


Generic Variance

Wann super, wann extends? Beispiele



TECHNISCHE
UNIVERSITÄT
DARMSTADT

```
1 public interface Consumer<T> {  
2     void accept(T t);  
3 }
```

Generic Variance

Wann super, wann extends? Beispiele

? **extends** T (Covarianz)

Generic Variance

Wann super, wann extends? Beispiele



TECHNISCHE
UNIVERSITÄT
DARMSTADT

? **extends** T (Covarianz)

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

Generic Variance

Wann super, wann extends? Beispiele



TECHNISCHE
UNIVERSITÄT
DARMSTADT

T (Invarianz)



T (Invarianz)

```
1 public void foo(List<T> list, Consumer<T> consumer) {  
2     list.add(supplier.get());  
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 }
```

Generic Variance

Wann `super`, wann `extends`? Beispiele



TECHNISCHE
UNIVERSITÄT
DARMSTADT

? `super` T (Contravarianz)



? **super** T (Contravarianz)

```
1 public void foo(List<T> list, Consumer<T> consumer) {  
2     list.add(supplier.get());  
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 }
```

Generic Variance

Wann super, wann extends? Beispiele



TECHNISCHE
UNIVERSITÄT
DARMSTADT

```
1 public interface Function<T, R> {  
2     R apply(T t);  
3 }
```


Generic Variance

Wann super, wann extends? Beispiele



TECHNISCHE
UNIVERSITÄT
DARMSTADT

■ Supplier

Generic Variance

Wann super, wann extends? Beispiele

■ `Supplier` → `Supplier<? extends T`

Generic Variance

Wann super, wann extends? Beispiele

- Supplier → Supplier<? extends T
- Consumer

Generic Variance

Wann super, wann extends? Beispiele



TECHNISCHE
UNIVERSITÄT
DARMSTADT

- `Supplier` → `Supplier<? extends T`
- `Consumer` → `Consumer<? super T`

Generic Variance

Wann super, wann extends? Beispiele



TECHNISCHE
UNIVERSITÄT
DARMSTADT

- `Supplier` → `Supplier<? extends T`
- `Consumer` → `Consumer<? super T`
- `Function`

Generic Variance

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

Generics Basiskonzept

Diamond operator

Generische Methoden

Statische generische Methoden

Eingeschränkte Typparameter

Erben von generischen Klassen

Erstellen von generischen Arrays



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



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



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

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



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



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

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

Generics!

„**Generics** — You think you understand it, and then you don't. . .
. . . until you eventually do.“