04-Objekte-4-Generics-2

Objektorientierte Programmierung | Matthias Tichy

Software Engineering
Programming Languages

universität ulm

Lernziele

- Keywords super und extends in Typargumenten

Collection<E>

- Auswahl von Methoden:

```
boolean add(E e);

boolean contains(Object o);

boolean addAll(Collection<? extends E> c);

boolean removeAll(Collection<?> c);

default boolean removeIf(Predicate<? super E> filter) {
  ...
}
```

https://docs.oracle.com/en/java/javase/20/docs/api/java.base/java/util/Collection.html

? super T / ? extends T

Type Arguments

> " Type arguments may be either reference types or wildcards. Wildcards are useful in situations where only partial knowledge about the type parameter is required.“

```
List<Integer> inList = new LinkedList<Integer>();
inList.add(22);

List<Number> numList = new LinkedList<Number>();
numList.add(12);
numList.add(12.0);

List<? super Number> superNumberList;
List<? extends Number> extendsNumberList;
```

James Gosling, Bill Joy, Guy Steele, Gilad Bracha, Alex Buckley, Daniel Smith, Gavin Bierman:
The Java® Language Specification - Java SE 20 Edition – 2023-03-03 - §4.5.1. Type Arguments of Parameterized Types
https://docs.oracle.com/javase/specs/jls/se19/html/jls-4.html#jls-4.5.1

Wofür gibt es die Wildcards?

Beispiel ohne

- Generics sind invariant, d.h.
- Generische Typen sind nicht typkompatibel bei unterschiedlichen Referenztypen als Typargumenten

```
List<Integer> inList = new LinkedList<Integer>();
List<Number> numList = inList;
numList.add(12);
```

- Vermeidet folgendes Problem:

```
numList.add(12.0);          // one can add doubles into a List of Integers
Integer i = inList.get(0); // would lead to a ClassCastException at runtime
```

Wofür gibt es die Wildcards?

Parameter von Methoden

- Wie wird denn dann Typkompatibilität bei Generics für Unter- bzw. Oberklassen ausgedrückt?

```
interface Collection<E> {
  boolean add(E e);

  boolean contains(Object o);

  boolean addAll(Collection<? extends E> c);

  boolean removeAll(Collection<?> c);

  default boolean removeIf(Predicate<? super E> filter) {
    ...
  }
}
```

Wofür gibt es die Wildcards?

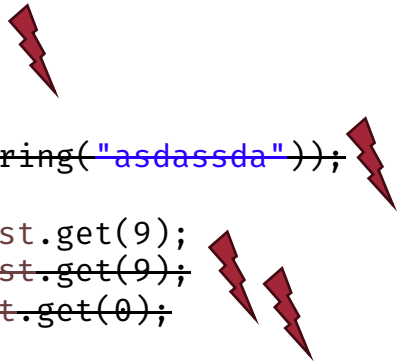
Parameter von Methoden

- Beispiele für super:

```
List<Integer> inList = new LinkedList<Integer>();
List<Number> numList = new LinkedList<Number>();
List<Object> objList = new LinkedList<Object>();

List<? super Number> superNumberList;
superNumberList = numList;
superNumberList = objList;
superNumberList = inList;
superNumberList.add(12);
superNumberList.add(12.0);
superNumberList.add(new String("asdassda"));

Object obj = superNumberList.get(9);
Number num = superNumberList.get(9);
Integer i = superNumberList.get(0);
```



Schreiben → super

Wofür gibt es die Wildcards?

Parameter von Methoden

- Beispiele für extend:

```
List<Integer> inList = new LinkedList<Integer>();
List<Number> numList = new LinkedList<Number>();
List<Object> objList = new LinkedList<Object>();
List<Double> doubleList = new LinkedList<Double>();

List<? extends Number> extendsNumberList;
extendsNumberList = numList;
extendsNumberList = objList;
extendsNumberList = inList;
extendsNumberList = doubleList;

extendsNumberList.add(Integer.valueOf(42));
Number num = extendsNumberList.get(0);
```

lEsen → extends

Stream<T>

- Auswahl von Methoden:

```
Stream<T> filter(Predicate<? super T> predicate);
Stream<T> distinct();
Stream<T> sorted();

<R> Stream<R> map(Function<? super T, ? extends R> mapper);

T reduce(T identity, BinaryOperator<T> accumulator);
Optional<T> min(Comparator<? super T> comparator);

boolean anyMatch(Predicate<? super T> predicate);
void forEach(Consumer<? super T> action);
```

Später bei Functional Java

https://docs.oracle.com/en/java/javase/20/docs/api/java.base/java/util/stream/Stream.html

Lernziele

- Keywords super und extends in Typargumenten