Funktionale und objektorientierte Programmierkonzepte

TECHNISCHE UNIVERSITÄT DARMSTADT

Präsenz-Sprechstunde B

Simon Hock, Nhan Huynh, Daniel Mangold

Überblick

TECHNISCHE UNIVERSITÄT DARMSTADT

Generics [5min]

Kovarianz [10min]
- Rückgabewert
- Arrays

Wildcards [10min]
- Lower bounded Wildcard
- Upper bounded Wildcard

Arbeitsphase [80min]

Generics

TECHNISCHE UNIVERSITÄT DARMSTADT

- Generizität nur auf **Referenztypen** möglich!
- Abstraktion, Wiederverwendbarkeit von Klassen bzw. Methoden
- Weniger redundanter Code, da man eine Klasse/ Methode durch Generizität auf mehrere Referenztypen anwenden kann, ohne diese speziell für einen Typ zu spezifizieren.

Kovarianz

TECHNISCHE UNIVERSITÄT DARMSTADT

- Beruhen auf dem Ersetzbarkeitsprinzip
- Der zurückgelieferte Wert der Unterklassen muss mit dem der Oberklasse vereinbar sein, also es darf kein allgemeinerer Typ sein
- Wo kann das auftreten?
  - Parameter
  - Rückgabetyp
  - `throws`-klausel

Rückgabewert

TECHNISCHE UNIVERSITÄT DARMSTADT

- Der aktuale Rückgabetyp entspricht entweder dem formalen Rückgabetyp oder ist ein Subtyp vom formalen Rückgabetyp
  - Rückgatetyp $\neq$ Rückgabewert

Rückblick:

- *Statischer Typ*:
  - Typ, der bei der Variablendeklaration angegeben wird
  - Ist bei der Kompilierung bekannt
- *Dynamischer Typ*:
  - Typ des tatsächlichen Objekts
  - Ist erst zur Laufzeit bekannt
  - Kann vom statischen Typ abweichen $\rightarrow$ Gleich oder Subtyp des statischen Typs

Beispiel

TECHNISCHE UNIVERSITÄT DARMSTADT

```java
public interface NumberAdder {

    Number add(Number a, Number b);
}
```

```java
public class IntegerAdder implements NumberAdder {

    @Override
    public Integer add(Number a, Number b) {
        return a.intValue() + b.intValue();
    }
}

public class DoubleAdder implements NumberAdder {
    @Override
    public Number add(Number a, Number b) {
        return a.doubleValue() + b.doubleValue();
    }
}
```

1. Welchen Typ hat die Zeile 7?
2. Welchen Typ hat die Zeile 8?

```
1 NumberAdder adder1 = new IntegerAdder();
2 IntegerAdder adder2 = new IntegerAdder();
3
4 Number a = 5.312; // Type Double
5 Number b = 2; // Type Integer
6
7 adder1.add(a,b);
8 adder2.add(a,b);
```

1. Welchen Typ hat die Zeile 7?
2. Welchen Typ hat die Zeile 8?

```
1 NumberAdder adder1 = new IntegerAdder();
2 IntegerAdder adder2 = new IntegerAdder();
3
4 Number a = 5.312; // Type Double
5 Number b = 2; // Type Integer
6
7 adder1.add(a,b);
8 adder2.add(a,b);
```

1. `Number`
2. `Integer`

Arrays

TECHNISCHE UNIVERSITÄT DARMSTADT

- Die Komponententypen im Array können gleich dem Komponententypen oder ein Subtyp dessen sein
- Der dynamische Typ des Arrays kann gleich dem statischen Typ oder ein Subtyp dessen sein

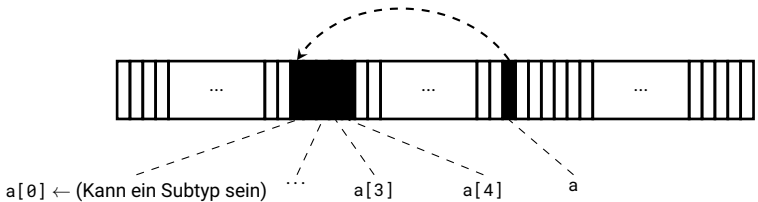


Abbildung: Abstrakte Visualisierung des Speicherplatzes eines Arrays a

Beispiel: Komponententypen

TECHNISCHE UNIVERSITÄT DARMSTADT

```
1 byte b = 1;
2 short s = 3;
3 int i = 4;
4 float f = 5.6f;
5 long l = 7;
6 double d = 8.9;
7 Number[] numbers = new Number[6];
8 numbers[0] = b; numbers[1] = s;
9 numbers[2] = i; numbers[3] = f;
10 numbers[4] = l; numbers[5] = d;
```

Beispiel: Dynamischer Typ

TECHNISCHE UNIVERSITÄT DARMSTADT

```
1 Number[] integers = new Integer[5];
2 Number[] doubles = new Double[integers.length];
3 for (int i = 0; i < integers.length; i++) {
4   integers[i] = i;
5   doubles[i] = (double) i;
6 }
```

Wirft die Zeile 5 einen Fehler? (Begründung)
Wenn ja:

1. Welcher Fehler wird geworfen?
2. Wird der Fehler während der Kompilierung oder erst zur Laufzeit erkannt?

```
1 Number[] integers = new Integer[5];
2 Number[] doubles = new Double[integers.length];
3 for (int i = 0; i < integers.length; i++) {
4   integers[i] = i;
5   doubles[i] = i;
6 }
```

Wirft die Zeile 5 einen Fehler? (Begründung)
Wenn ja:

1. Welcher Fehler wird geworfen?
2. Wird der Fehler während der Kompilierung oder erst zur Laufzeit erkannt?

```
1 Number[] integers = new Integer[5];
2 Number[] doubles = new Double[integers.length];
3 for (int i = 0; i < integers.length; i++) {
4   integers[i] = i;
5   doubles[i] = i;
6 }
```

1. `ArrayStoreException`
2. Der Fehler wird erst zur Laufzeit erkannt, da der Komponententyp ein des statischen Typs des Arrays ist

Wildcards [15min]

TECHNISCHE UNIVERSITÄT DARMSTADT

- Wie sieht es mit Wilcards aus?
- Die Typparameter sind fest d.h. es ist nicht erlaubt, anstelle des eigentlichen Typparameters jetzt andere direkt oder indirekt abgeleitete Klassen einzusetzen
  - Nichtübertragbarkeit von Vererbung auf Typparameter
- „Die Typparameter werden vom Compiler fest in die Klasse gebrannt“

```java
public class Printer<T> {

    public void print(T element) {
        System.out.println("Printed element: " + element.toString());
    }
}
```

```java
Printer<Number> printer = new Printer<Number>();
printer = new Printer<Integer>(); // Compile-Error
```

Wildcards

TECHNISCHE UNIVERSITÄT DARMSTADT

- Ermöglicht Kovarianz bei Typparametern
- Definition der Einschränkungen bei der ***Instanziierung*** von Typparametern!
- Flexibilität von Typparamer von Objekten
- Nach untend beschränkt: `? super T` – *Lower bounded Wildcard*
- Nach oben beschränkt `? extends T` – *Upper bounded Wildcard*

Beispiel

TECHNISCHE UNIVERSITÄT DARMSTADT

```
1 Printer<? extends Number> printer = new Printer<Number>();
2 printer = new Printer<Integer>();
3 printer = new Printer<Double>();
```

Lower bounded Wildcard

TECHNISCHE UNIVERSITÄT DARMSTADT

- Typparameter nach unten in der Vererbungshierarchie beschränkt
- Typparameter ist entweder
  - T oder
  - direkt oder indirekte Oberklassen von T oder
  - direkt oder indirekte implementierte Interfaces von T
- Formaler Aufbau: `? super T`

Katzenhierachie

TECHNISCHE UNIVERSITÄT DARMSTADT

```
1 public class Animal {}
2 public class Cat extends Animal{}
3 public class RagdollCat extends Cat {}
4 public class MaineCoonCat extends Cat {}
5 public class BlackMaineCoonCat extends MaineCoonCat {}
```

Beispiel

TECHNISCHE UNIVERSITÄT DARMSTADT

Wir möchten eine schwarze Maine Coon Katze in einer Liste hinzufügen:

```
1 public static void addCat(List<BlackMaineCoonCat> cats,
2     BlackMaineCoonCat cat) {
3     cats.add(cat);
4 }
```

Beispiel

TECHNISCHE UNIVERSITÄT DARMSTADT

Wir möchten eine schwarze Maine Coon Katze in einer Liste hinzufügen:

```
1 public static void addCat(List<BlackMaineCoonCat> cats,
2     BlackMaineCoonCat cat) {
3     cats.add(cat);
4 }
```

Das erlaubt uns nur das Hinzufügen einer schwarzen Maine Coon Katze in einer `List<BlackMaineCoonCat>`. Kann man das nicht flexibler machen?

```
1 public static void addCat(List<? super BlackMaineCoonCat> cats,
2                BlackMaineCoonCat cat) {
3   cats.add(cat);
4 }
```

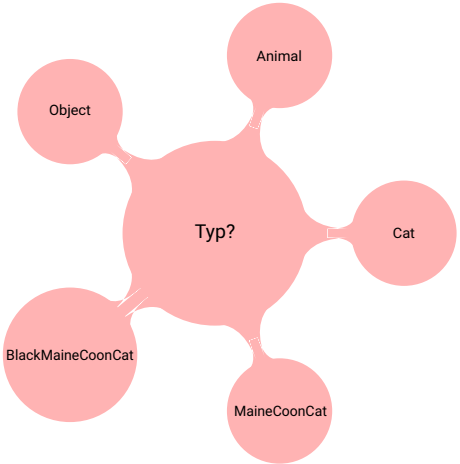
```
1  List<BlackMaineCoonCat> bl = new ArrayList<>();
2  List<MaineCoonCat> ml = new ArrayList<>();
3  List<Cat> cl = new ArrayList<>();
4  List<Animal> al = new ArrayList<>();
5  List<Object> ol = new ArrayList<>();
6  BlackMaineCoonCat cat = new BlackMaineCoonCat();
7
8  addCat(bl, cat);
9  addCat(ml, cat);
10 addCat(cl, cat);
11 addCat(al, cat);
12 addCat(ol, cat);
```

Lesen

TECHNISCHE UNIVERSITÄT DARMSTADT

Welchen Typ können wir in der Zeile 4 lesen?

```
1 public static void addCat(List<? super BlackMaineCoonCat> cats,
2                BlackMaineCoonCat cat) {
3   cats.add(cat);
4   cats.get(0); // Type?
5 }
```


Animal
Object
Typ?
Cat
BlackMaineCoonCat
MaineCoonCat

Welchen Typ können wir in allen Fällen garantieren?

TECHNISCHE UNIVERSITÄT DARMSTADT

Alle Klassen erben direkt oder indirekt von der Klasse `Object`, also können wir mindestens den Typ `Object` garantieren!

```
1 public static void addCat(List<? super BlackMaineCoonCat> cats,
2                BlackMaineCoonCat cat) {
3   cats.add(cat);
4   Object o = cats.get(0);
5 }
```

Schreiben

TECHNISCHE UNIVERSITÄT DARMSTADT

Welche Zeilen laufen fehlerfrei durch?

```
1 public static void addCat(List<? super BlackMaineCoonCat> cats,
2                BlackMaineCoonCat cat) {
3   cats.add(cat); // Pass or error?
4   cats.add(new MaineCoonCat()); // Pass or error?
5   cats.add(new Cat()); // Pass or error?
6   cats.add(new Animal()); // Pass or error?
7   cats.add(new Object()); // Pass or error?
8 }
```

Listentyp ist entweder:

- Tatsächlicher Listentyp: `BlackMaineCoonCat`
  - Kann nur `BlackMaineCoonCat` oder Subtypen dessen einfügen
  - Fehlerhafte Zeilen: 4, 5, 6,7
- Tatsächlicher Listentyp: `MaineCoonCat`
  - Kann nur `MaineCoonCat` oder Subtypen dessen einfügen
  - Fehlerhafte Zeilen: 5, 6,7
- Tatsächlicher Listentyp: `Cat`
  - Kann nur `Cat` oder Subtypen dessen einfügen
  - Fehlerhafte Zeilen: 6, 7
- Tatsächlicher Listentyp: `Animal`
  - Kann nur `Animal` oder Subtypen dessen einfügen
  - Fehlerhafte Zeilen: 7
- Tatsächlicher Listentyp: `Object`
  - Kann nur `Object` oder Subtypen dessen einfügen
  - Fehlerhafte Zeilen: Keine

Welchen Typ können wir in allen Fällen garantieren?

TECHNISCHE UNIVERSITÄT DARMSTADT

Wir können in jeder Liste eine `BlackMaineCoonCat` oder ein Subtypen dessen auf jeden Fall hinzufügen!

```
1 public static void addCat(List<? super BlackMaineCoonCat> cats,
2                BlackMaineCoonCat cat) {
3   cats.add(cat); // Pass
4   cats.add(new MaineCoonCat()); // Error
5   cats.add(new Cat()); // Error
6   cats.add(new Animal()); // Error
7   cats.add(new Object()); // Error
8 }
```

Upper bounded Wildcard

TECHNISCHE UNIVERSITÄT DARMSTADT

- Typparameter nach oben in der Vererbungshierarchie beschränkt.
- Typparameter ist entweder
  - T oder
  - direkt oder indirekte Unterklasse von T oder
- Formaler Aufbau: `? extends T`
- `? = ? extends Object`

Beispiel

TECHNISCHE UNIVERSITÄT DARMSTADT

Wir möchten dieser Summe von Zahlen in einer Liste bilden:

```
1 public static double sum(List<Number> numbers) {
2   double sum = 0;
3   for (Number number : numbers) {
4     sum += number.doubleValue();
5   }
6   return sum;
7 }
```

Beispiel

TECHNISCHE UNIVERSITÄT DARMSTADT

Wir möchten dieser Summe von Zahlen in einer Liste bilden:

```
1 public static double sum(List<Number> numbers) {
2   double sum = 0;
3   for (Number number : numbers) {
4     sum += number.doubleValue();
5   }
6   return sum;
7 }
```

Das erlaubt uns nur dass Bilden einer Summe für den Typparameter `Number`. Was machen wir, wenn wir die Summe von `Integer` oder `Double` bilden wollen, ohne redundanten Code zu schreiben?

```java
public static double sum(List<? extends Number> numbers) {
  double sum = 0;
  for (Number number : numbers) {
    sum += number.doubleValue();
  }
  return sum;
}
```

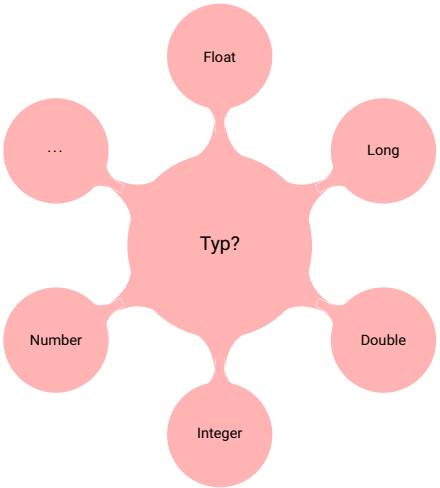
```
1 List<Number> nl = new ArrayList<>();
2 List<Integer> il = new ArrayList<>();
3 List<Double> dl = new ArrayList<>();
4 List<Long> ll = new ArrayList<>();
5 List<Float> fl = new ArrayList<>();
6 sum(nl);
7 sum(il);
8 sum(dl);
9 sum(ll);
10 sum(fl);
```

Lesen

TECHNISCHE UNIVERSITÄT DARMSTADT

Welchen Typ können wir in der Zeile 3 lesen? (Außerdem Number)

```
1 public static double sum(List<? extends Number> numbers) {
2   double sum = 0;
3   for (Number number : numbers) {
4     sum += number.doubleValue();
5   }
6   return sum;
7 }
```


Float
...
Long
Typ?
Number
Double
Integer

Welchen Typ können wir in allen Fällen garantieren?

TECHNISCHE UNIVERSITÄT DARMSTADT

Alle Klassen sind Subtypen von `Number`, also können wir mindestens den Typ `Number` garantieren!

```
1 public static double sum(List<Number> numbers) {
2   double sum = 0;
3   for (Number number : numbers) {
4     sum += number.doubleValue();
5   }
6   return sum;
7 }
```

Schreiben

TECHNISCHE UNIVERSITÄT DARMSTADT

Welche Zeilen laufen fehlerfrei durch?

```
1 public static double sum(List<? extends Number> numbers) {
2   ...
3   Number number = 1; Integer integer = 2;
4   Double d = 3.4;
5   numbers.add(number); // Pass or Error?
6   numbers.add(integer); // Pass or Error?
7   numbers.add(d); // Pass or Error?
8   return sum;
9 }
```

Listentyp ist entweder:

- Tatsächlicher Listentyp: `Number`
  - Kann `Number` oder ein Subtyp dessen einfügen
  - Fehlerhafte Zeilen: Keine
- Tatsächlicher Listentyp: `Integer`
  - Kann `Integer` oder ein Subtyp dessen einfügen
  - Fehlerhafte Zeilen: 5, 7
- Tatsächlicher Listentyp: `Double`
  - Kann `Double` oder ein Subtyp dessen einfügen
  - Fehlerhafte Zeilen: 5, 6
- Tatsächlicher Listentyp: `Long`
  - Kann `Double` oder ein Subtyp dessen einfügen
  - Fehlerhafte Zeilen: 5, 6, 7
- ...

Welchen Typ können wir in allen Fällen garantieren?

TECHNISCHE UNIVERSITÄT DARMSTADT

Wir können in jeder Liste auf jeden Fall `null` hinzufügen!

```
1 public static double sum(List<? extends Number> numbers) {
2   ...
3   Number number = 1; Integer integer = 2; Double d = 3.4;
4   numbers.add(number); // Error
5   numbers.add(integer); // Error
6   numbers.add(d); // Error
7   numbers.add(null); // Pass
8   return sum;
9 }
```

Zusammenfassung

TECHNISCHE UNIVERSITÄT DARMSTADT

Gegeben sei eine Liste `List<E>` mit einem beliebigen Typ `E`.

- Schreiben erfolgt mittels der Methode `add(E e)`
- Lesen erfolgt mittels der Methode `get(int index)`

Statischer Typ der Liste	Dynamischer Typ der Liste	Schreiben	Lesen
`List<?>`	`Object` und alle Subklassen von `Object`	Nur `null`	`Object`
`List<? super E>`	`E` und alle Basisklassen und implementierten Interfaces von `E`	Nur `null`	`T`
`List<? extends E>`	`E` und alle Subklassen von `E`	`E` und alle Subtypen von `E`	`Object`

Tabelle: Überblick Wildcards

Selbstständiges Arbeiten