ALP II

…weiter mit generischen Datentypen

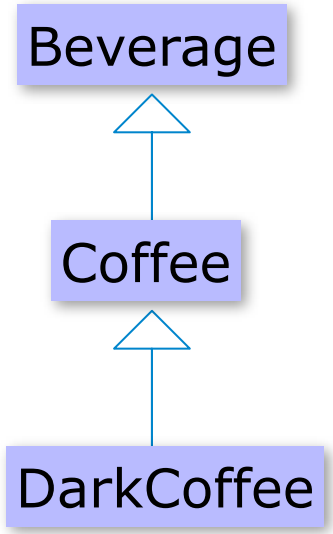
Teil 2

SS 2012

Prof. Dr. Margarita Esponda

Generische Datentypen in Java

```
public class Cup<T> {
        T beverage;
        public Cup(T beverage) {
                this.beverage = beverage;
        }
        ...
}
```



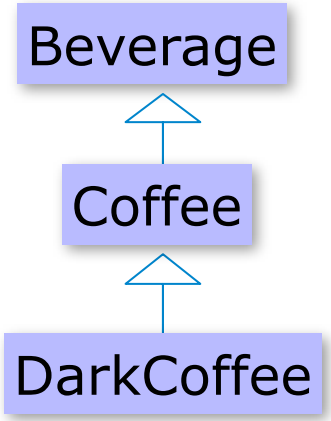
Methodenschablonen

```
public class Example {

        public static <T> T randomChoose( T m, T n ){
                return Math.random() > 0.5 ? m : n;
        }

        public static void main(String args[]){
                System.out.println(Example.randomChoose("Ja", "Nein"));
                System.out.println(Example.randomChoose("Ja", 5));
                System.out.println(Example.randomChoose(4.5, new Rectangle()));
                String s = Example.randomChoose(new Rectangle, "Text");
        }
}
```

Macht wenig Sinn!

Übersetzungsfehler nur hier!

Generische Datentypen und Vererbung in Java



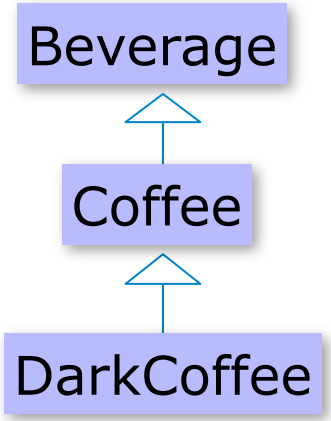
folgende Zuweisungen sind **legal**.

```
DarkCoffee darkCoffee = new DarkCoffee();
Beverage beverage = new DarkCoffee();

Cup<Coffee> cup = new Cup<Coffee>(coffee);
Cup<DarkCoffee> cup = new Cup<DarkCoffee>(coffee);
```

Typanpassungen

```
List<Integer> list = new ArrayList<Integer>();
```

Generische Datentypen und Vererbung in Java



folgende Zuweisung ist **illegal**

```
Cup<Coffee> cup2 = new Cup<DarkCoffee>(darkCoffee);
```

1. Fall Invarianz

- der Typparameter ist eindeutig
- keine Einschränkung auf den Typparameter
- aber innerhalb von Zuweisungen keinerlei Flexibilität
- Typfehler können besser kontrolliert werden.

erlaubt

```
Number  n = new Integer(3);
```

nicht erlaubt!

```
ArrayList<Number> list = new ArrayList<Integer>();
```

müssen gleich sein!

Schlechte Erfahrung mit Arrays

Beispiel:

```
Number[] nums = new Integer[100];
```

ist OK, weil Integer von Number abgeleitet wird,

aber

```
nums[1]  = 1.0;
nums[2]  = new Double(1.0);
```

verursachen ***ArrayStoreExceptions*** zur Laufzeit, weil

Integer <- Double

nicht zuweisungskompatibel sind.

2. Fall Kovarianz

Referenzen müssen explizit mit der Syntax

<? extends T>

gekennzeichnet werden.

T ist der generellste Typ, der zugelassen ist (obere Einschränkung).

Beispiel:

```
Cup<? extends Beverage> cup =
                    new Cup<DarkCoffee>(darkCoffee);

Box<? extends Number> nBox =
                              new Box<Integer>(9);
```

Eingeschränkte Parametrisierung

```
public class MathBox<E extends Number>
                              extends Box<Number>
{...}
```

Die Klasse **MathBox** kann mit beliebigen Datentypen, die als Unterklassen von **Number** definiert sind, parametrisiert werden.

Beispiele:

```
new MathBox<Integer>(5)
new MathBox<Double>(32.1)
```

Legal

```
new MathBox<String>("Zahlen");
```

Illegal

Eingeschränkte Parametrisierung

Der Compiler kontrolliert, dass mindestens B und C Interfaces sind.

```
public class M<E extends A & B & C>
{...}
```

3. Fall Kontravarianz

Referenzen müssen explizit mit der Syntax

<? super T>

gekennzeichnet werden.

untere Einschränkung

Beispiel:

```
Cup<? super DarkCoffee> cup =
                        new Cup<Beverage>(beverage);
```

```
Cup<? super DarkCoffee> cup =
                        new Cup<Object>(beverage);
```

4. Fall Bivarianz

Referenzen müssen explizit mit der Syntax

<?>

gekennzeichnet werden.

Beispiel:

Es gibt keine Einschränkung

```
Cup<?> cup = new Cup<String>(coffee);
```

```
Cup<?> cup5 = new Cup<String>("Hi");
Cup<?> cup6 = new Cup<Object>(7);
Cup<?> cup7 = new Cup<String>("Hi");
cup5 = cup6;
cup6 = cup7;
```

legale Zuweisungen

Generische Datentypen

Eine Klasse kann mit mehreren Datentypen parametrisiert werden.

```
public class  Generic <T1, T2, T3> { ... }
```

Anwendungsbeispiel:

```
public static void main(String[] args) {


      Generic<String, Integer, Integer> sg =
            new Generic<String, Integer, Integer>("Text", 3, 5);
}
```

Generische Datentypen

Eine generische Klasse kann als Unterklasse einer anderen generischen Klasse definiert werden.

```
public class SpecialBox<E> extends Box<E> { ... }
```

Folgende Zuweisung ist dann legal.

```
Box<String> special = new SpecialBox<String>("Text");
```

Gebundene Typparameter

Innerhalb einer parametrisierten Klasse ist der Typparameter ein gültiger Datentyp, der innerhalb innerer Klassendefinitionen gebunden sein kann.

Beispiel:

```
public class OuterClass<T>{
    …
    private class  InnerClass<E extends T>{
        …
    }
}
```

Generische Datentypen in Java

```
public class Cup<T extends Beverage> {
        T beverage;
        public Cup(T beverage) {
                this.beverage = beverage;
        }
        ...
}
```