Java SE 8

Generics

Generische Klasse

```
public class MyClass<T> {
    private T t;

    // verwendet den generischen Typ der Klasse
    private void set(T t) {
    }

    private T get() {
        return t;
    }

    // nicht generische Methode
    public String machWas() {
        return "Moin";
    }
}
```

diamond

- Seit Java 1.7

```
List<String> mc = new ArrayList<>();
```

- Vor Java 1.7

```
List<String> mc =
     new ArrayList<String>();
```

Typ Parameter

- Kann verwendet werden in
  - Deklaration der Klasse
  - Variablen
  - Methoden Parametern
  - Rückgabetypen

Namenskonventionen

- Großbuchstabige Bezeichner
  - Containerklassen und -schnittstellen
    - meist **E** für "Element"
      - Typparameter bezeichnet den Typ der enthaltenen Elemente.
  - Parametrisierte Klassen
    - meist ein **T** für den ersten generischen "Typ"
    - ein **S** für zweiten generischen Typ
    - **U** für dritten, **V** für vierten etc.
  - Manchmal
    - **K** und **V** für "Key" und "Value"
    - **R** für "Return"
    - **N** für "Number"

Generisches erweiterte Klasse

```
class MyClass<T> {}

class NextClass<T> extends MyClass<T> {}

// auch möglich
class MyClass<T> {}

class NextClass<T, S> extends MyClass<S> {}

// nicht möglich
class MyClass<T> {}

class NextClass<S> extends MyClass<T> {}
```

Raw-Types

- Ein generischer Typ, der nicht als parametrisierter Typ, also ohne Typargument, genutzt wird, heißt Raw-Type

Type-Erasure

- Bei der Kompilierung einer generischen Code wird die Typeinformation gelöscht, da die Java-Laufzeitumgebung keine Generics kennt.

Generisches Klasse als Basis für nicht generische Klassen

```
class MyClass<T> {}

class OtherClass
    extends MyClass<String> {}
```

Generisches Interface

```
interface MyInterface<K,V> {
  void put(K key, V value);
  V get(K key);
}

// Implementierung
class MyClass implements MyInterface<String, Integer> {
  public void put(String s, Integer i) {}
  public Integer get(String s) { return null; }
}

// Compilererror
class MyClass implements MyInterface<String, Integer> {
  public void put(String s, Integer i) {}
  public String get(String s) { return null; }
}
```

Generisches Interface

```
interface MyInterface<K,V> {
  void put(K key, V value);
  V get(K key);
}

// Implementierung
class GenClass<K,V> implements MyInterface<K, V> {
  public void put(K k, V v) {}
  public V get(K k) { return null; }
}

// Kombinationen sind auch möglich
class HalfGenClass<K> implements MyInterface<K, String> {
  public void put(K k, String v) {}
  public String get(K k) { return null; }
}
```

Generics

- Eine generische Klasse kann nicht-generische Methoden enthalten
- Eine nicht-generische Klasse kann generische Methoden enthalten

Generische Methode

```
class MyClass {
  // definiert einen generischen Typ
  public <T> void machWas(T t) {
  }
}

interface MyInterface<T,S> {
  // definiert einen eigenen generischen Typ
  <T> void machWas(T t);
}

class OtherClass<T> {
  // definiert einen eigenen generischen Typ
  <T> OtherClass(T a) {
    //…
  }
}
```

Bounded Parameters

```
class MyClass<T extends OtherClass> {
  private T t;

  private void set(T t) {
  }
}
```

Bounded Parameters

```
class MyClass<T extends OtherClass> {}

//Compilererror
MyClass<String> c = new MyClass<>();
```

Bounds

- `class MyClass<T extends C & I & I2 > {}`
- Der Typ muss ein Subtyp aller Bounds sein
- Bound kann eine Klasse oder ein oder Mehrere Interfaces sein

Bounds

- Bounds können class, interface oder enum sein
- Keyword implements wird für interfaces **nicht** verwendet

Wildcards

- ? ist ein unbekannter Typ
- Kann ein Parameter, lokale-, instanz- oder statische Variable sein

Wildcards

```
class MyClass {}

class OtherClass extends MyClass {}

//Compilererror: Typ muss gleich sein
List<MyClass> l =
     new ArrayList<OtherClass>();
```

Wildcards

```
class MyClass {}

class OtherClass extends MyClass {}

List<?> l = new ArrayList<OtherClass>();
```

Wildcards

```
class MyClass {}

class OtherClass extends MyClass {}

List<?> l = new ArrayList<OtherClass>();

//Compilererror
l.add(new OtherClass())
```

(upper) Bounded Wildcards

```
class MyClass {}

class OtherClass extends MyClass {}

List<? extends OtherClass> l =
    new ArrayList<OtherClass>();
```

(upper) Bounded Wildcards

```
List<? extends String> l =
     new ArrayList<String>();
```

(lower) Bounded Wildcards

```
List<? super Mensch> l =
     new ArrayList<>();

l.add(new Object());
//Compilererror
l.add(new Student());
```

Kannst du dich noch erinnern?