OCP Java SE 8

Generics

Generische Klasse

```
class X<T> {
  private T t;

  private void set(T t) {
  }

  private T get() {
    return t;
  }

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

diamond

```
MyClass<String> mc = new MyClass<>();

MyClass<String> mc =
    new MyClass<String>();
```

Typ Parameter

- Kann verwendet werden in
  - Deklaration der Klasse
  - Variablen
  - Methoden Parametern
  - Rückgabetypen
  - class ClassName <T1, T2, ..., Tn>

Generisches erweiterte Klasse

```
class X<T> {}

class NextX<T> extends X<T> {}

// auch möglich

class X<T> {}

class NextX<X, T> extends X<T> {}

// nicht möglich

class X<T> {}

class NextX<X> extends X<T> {}
```

Generisches Klasse als Basis für nicht generische Klassen

```
class X<T> {}

class NextX extends X<String> {}
```

Generisches Interface

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

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

// Compiler Fehler
class B implements A<String, Integer> {
  public void put(String s, Integer i) {}
  public String get(String s) { return null; }
}
```

Generisches Interface

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

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

// Kombinationen sind auch möglich
class HalfGenericB<K> implements A<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 Y {
  public <T> void machWas(T t) {
  }
}

interface MyInterface<A,B> {
  <C> void machWas(C c);
}

class MyClass<A> {
  <A> MyClass(A a) {
    //…
  }
}
```

Bounded Parameters

```
class X<T extends Y> {
  private T t;

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

Bounded Parameters

```
class MyClass1<T extends MyClass2> {}

MyClass1<String> c = new MyClass1<>();
//Compilerfehler
```

Bounds

- class X<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 Thing {}

class NewThing extends Thing {}

List<Thing> l = new ArrayList<NewThing>();
//Compilerfehler
//Klammertyp muss gleich sein
```

Wildcards

```
class Thing {}

class NewThing extends Thing {}

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

Wildcards

```
class Thing {}

class NewThing extends Thing {}

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

l.add(new NewThing()) //Kompilerfehler
```

(upper)Bounded Wildcards

```
class Thing {}

class NewThing extends Thing {}

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

(upper)Bounded Wildcards

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

(lower)Bounded Wildcards

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

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

Generics

```
Test<String> x = new Test<String>();

Test<String> x = new Test<>();

Test<String> x = new Test();

Test<> x = new Test<String>();

Test x = new Test<String>();
```