Type Erasure

Carsten Gips (HSBI)

Typ-Löschung (*Type-Erasure*)

```java
class Studi<T> {
    T myst(T m, T n) { return n; }

    public static void main(String[] args) {
        Studi<Integer> a = new Studi<>();
        int i = a.myst(1, 3);
    }
}
```

Typ-Löschung (*Type-Erasure*)

```java
class Studi<T> {
    T myst(T m, T n) { return n; }

    public static void main(String[] args) {
        Studi<Integer> a = new Studi<>();
        int i = a.myst(1, 3);
    }
}
```

```java
class Studi {
    Object myst(Object m, Object n) { return n; }

    public static void main(String[] args) {
        Studi a = new Studi();
        int i = (Integer) a.myst(1, 3);
    }
}
```

Type-Erasure bei Nutzung von Bounds

```
class Cps<T extends Number> {
    T myst(T m, T n) {
        return n;
    }

    public static void main(String[] args) {
        Cps<Integer> a = new Cps<>();
        int i = a.myst(1, 3);
    }
}
```

```
class Cps {
    Number myst(Number m, Number n) {
        return n;
    }

    public static void main(String[] args) {
        Cps a = new Cps();
        int i = (Integer) a.myst(1, 3);
    }
}
```

Raw-Types: Ich mag meine Generics “well done” :-)

Raw-Types: Instanziierung ohne Typ-Parameter => `Object`

```
Stack s = new Stack(); // Stack von Object-Objekten
```

- Wegen Abwärtskompatibilität zu früheren Java-Versionen noch erlaubt.
- Nutzung wird nicht empfohlen! (Warum?)

Folgen der Typ-Löschung: *new*

`new` mit parametrisierten Klassen ist nicht erlaubt!

```
class Fach<T> {
    public T foo() {
        return new T();  // nicht erlaubt!!!
    }
}
```

Grund: Zur Laufzeit keine Klasseninformationen über `T` mehr

Folgen der Typ-Löschung: *static*

`static` mit generischen Typen ist nicht erlaubt!

```
class Fach<T> {
    static T t;                    // nicht erlaubt!!!
    static Fach<T> c;              // nicht erlaubt!!!
    static void foo(T t) { ... }; // nicht erlaubt!!!
}

Fach<String>  a;
Fach<Integer> b;
```

Grund: Compiler generiert nur eine Klasse! Beide Objekte würden sich die statischen Attribute teilen (Typ zur Laufzeit unklar!).

Hinweis: Generische (statische) Methoden sind erlaubt.

Folgen der Typ-Löschung: *instanceof*

`instanceof` mit parametrisierten Klassen ist nicht erlaubt!

```
class Fach<T> {
    void printType(Fach<?> p) {
        if (p instanceof Fach<Number>)
            ...
        else if (p instanceof Fach<String>)
            ...
    }
}
```

```
class Fach {
void printType(Fach p) {
    if (p instanceof Fach)
        ...
    else if (p instanceof Fach)
        ...
    }
}
```

Folgen der Typ-Löschung: *.class*

`.class` mit parametrisierten Klassen ist nicht erlaubt!

```
boolean x;
List<String>  a = new ArrayList<String>();
List<Integer> b = new ArrayList<Integer>();

x = (List<String>.class == List<Integer>.class);  // Compiler-Fehler
x = (a.getClass() == b.getClass());               // true
```

Grund: Es gibt nur `List.class` (und kein `List<String>.class` bzw. `List<Integer>.class`)!

Wrap-Up

- Generics existieren eigentlich nur auf Quellcode-Ebene
- “Type-Erasure”:
  - Compiler entfernt generische Typ-Parameter => im Byte-Code nur noch Raw-Typen
  - Compiler baut passende Casts in Byte-Code ein
  - Transparent für User; Auswirkungen beachten!

LICENSE