Propädeutikum: Programmierung in der Bioinformatik

Mini-Exkurs: Generics

Thomas Mauermeier
04.02.2020
Ludwig-Maximilians-Universität München

```java
public class IntegerTupel {
    private int value1;
    private int value2;

    public IntegerTupel(int value1, int value2) {
        this.value1 = value1;
        this.value2 = value2;
    }

    public void setValue1(int value1) {
        this.value1 = value1;
    }

    public int getValue1() {
        return value1;
    }
}
```

Was, wenn ich jetzt aber beliebige Objekte zu Tupeln zusammenfassen will?

Naive Idee: Ich mach einfach ein Tupel das Objects speichert!

```java
public class NonGenericTupel {
    private Object value1;
    private Object value2;

    public NonGenericTupel(Object value1, Object value2) {
        this.value1 = value1;
        this.value2 = value2;
    }

    public void setValue1(Object value1) {
        this.value1 = value1;
    }

    public Object getValue1() {
        return value1;
    }
}
```

(Ein) Problem:
Hier wird mir immer etwas vom Typ Object returned!
Nerviges Typecasting nötig..

Lösung: Ich mache eine Tupel-Klasse mit Generics!

```
public class GenericTupel<T> {
    private T value1;
    private T value2;

    public GenericTupel(T value1, T value2) {
        this.value1 = value1;
        this.value2 = value2;
    }

    public void setValue1(T value1) {
        this.value1 = value1;
    }

    public T getValue1() {
        return value1;
    }
}
```

Hier oben: Typvariable spezifizieren

Wird dann je nachdem wie Objekt vom Typ GenericTupel erstellt wird, in der restlichen Klasse “ersetzt”

Beispiel: GenericTupel<**Double**> gt = new GenericTupel<**Double**>(1.0, 2.0);

```java
public class GenericTupel<T> {
    private T value1;
    private T value2;

    public GenericTupel(T value1, T value2) {
        this.value1 = value1;
        this.value2 = value2;
    }

    public void setValue1(T value1) {
        this.value1 = value1;
    }

    public T getValue1() {
        return value1;
    }
}
```

Resultierende Klasse kann man sich dann vorstellen, als würde jedes T mit einem Double ersetzt

Beispiel: GenericTupel<**Double**> gt = new GenericTupel<**Double**>(1.0, 2.0);

```java
public class GenericTupel<Double> {
    private Double value1;
    private Double value2;

    public GenericTupel(Double value1, Double value2) {
        this.value1 = value1;
        this.value2 = value2;
    }

    public void setValue1(Double value1) {
        this.value1 = value1;
    }

    public Double getValue1() {
        return value1;
    }
}
```

Resultierende Klasse kann man sich dann vorstellen, als würde jedes T mit einem Double ersetzt

Deswegen: Typ**variable!**

Mit Hilfe von **extends** lässt sich auch noch einschränken welche Typen es sein dürfen:

```
public class GenericNumberTupel<T extends Number> {
    private T value1;
    private T value2;

    public GenericNumberTupel(T value1, T value2) {
        this.value1 = value1;
        this.value2 = value2;
    }

    public void setValue1(T value1) {
        this.value1 = value1;
    }

    public T getValue1() {
        return value1;
    }
}
```