

Method References

Was sind Method References?

- **Ab Java 8**
- **gehört zum funktionaler Programmierstil**
- **Möglichkeit, Methodenaufrufe prägnant auszudrücken**
 - **Spezielle Form von Lambda-Ausdrücken**
- **Code wird lesbarer und verständlicher**

Lambda-Ausdrücke vs. Method Reference

```
List<String> list = Arrays.asList("apple", "banana", "cherry");  
list.forEach(item -> System.out.println(item));
```

Was passiert?

Der Ausdruck `item -> System.out.println(item)` implementiert eine abstrakte Methode innerhalb eines funktionalen Interfaces (Consumer)

```
list.forEach(new Consumer<String>() {  
    @Override  
    public void accept(String item) {  
        System.out.println(item);  
    }  
});
```

Lambda-Ausdrücke vs. Method Reference

```
List<String> list = Arrays.asList("apple", "banana", "cherry");  
list.forEach(System.out::println);
```

Was passiert?

Man kann sich *System.out::println* als eine Kurzform des zuvor genannten Lambda-Ausdrucks vorstellen.

Es wird auf eine bestehende Methode verwiesen (referenziert) .

Compiler erkennt automatisch,
welche Parameter eingesetzt
werden müssen

Wann sollte man am besten Lambdas nutzen?

- Wenn der Lambda-Ausdruck mehr als einen Methodenaufruf enthält

```
list.forEach(item -> {  
    System.out.println("Schaue nach dem nächsten Element. Ob es mit a beginnt?");  
    if(item.startsWith("a")){  
        System.out.println(item);  
    }else{  
        System.out.println("Element startet nicht mit a");  
    }  
});
```

- Method References nutzt man zum Aufruf einer Methode

Arten von Method-References

- **Referenz auf statische Methode:**
 - Hier wird auf eine statische Methode einer Klasse verwiesen
 - Bsp.: eigene statische Methode

```
public static void main(String[] args) {  
    List<Integer> list = Arrays.asList(1,2,3,4);  
    list.replaceAll(StaticMethodRef::printSquare);  
    System.out.println(list);  
}
```

```
private static int printSquare(int number){  
    return number*number;  
}
```

Methode,
das quadrierte
Zahlen zurückgibt

```
[1, 4, 9, 16]
```

Arten von Method-References

- **Referenz auf eine Instanzmethode eines bestimmten Objekts**
 - Hier wird auf eine Methode einer Klasse verwiesen
 - Man muss dafür ein Objekt der Klasse erstellen
 - Bsp.: eigene Methode

```
public class AnyMethodRef {  
    public static void main(String[] args) {  
        BicycleBrandComparator bicycleBrandComparator = new BicycleBrandComparator();  
  
        List<Bicycle> bicycleList = Arrays.asList(new Bicycle("Canyon"),  
            new Bicycle("BMC"), new Bicycle("Pegasus"));  
  
        bicycleList.sort(bicycleBrandComparator::compare);  
    }  
}
```

```
class Bicycle{  
    private String brand;  
    Bicycle(String brand){  
        this.brand = brand;  
    }  
  
    public String getBrand() {  
        return brand;  
    }  
}  
  
class BicycleBrandComparator {  
    public int compare(Bicycle a, Bicycle b){  
        return a.getBrand().compareTo(b.getBrand());  
    }  
}
```

Arten von Method-References

- Referenz auf eine Instanzmethode eines beliebigen Objekts eines bestimmten Typs
 - Hier wird auf eine Methode einer Klasse verwiesen, ohne ein Objekt von dieser Klasse erstellen zu müssen
 - Bsp.: `.toUpperCase()` der Klasse `String`

```
public static void main(String[] args) {  
    List<String> list = Arrays.asList("apple", "banana", "cherry");  
    list.replaceAll(String::toUpperCase);  
    System.out.println(list);  
}
```


Arten von Method-References

- Referenz auf einen Konstruktor

```
public static void main(String[] args) {  
    List<String> bikeBrands = Arrays.asList("Giant", "Scott", "Trek", "GT");  
    List<Bicycle> bicycleList =  
        bikeBrands  
            .stream()  
            .map(Bicycle::new)  
            .collect(Collectors.toList());  
}
```

ermöglicht, dass unsere ursprüngliche List<String>, in eine List<Bicycle> umgewandelt wird

Aufgabe

Sortieren einer Liste

- Erstelle eine Liste von Namen und sortieren Sie diese alphabetisch mithilfe einer Methodenreferenz. Nutze dafür die Methode `compareTo` der Klasse `String`.
- Ergänze den Code, sodass die Namen in der Konsole ausgegeben werden. Nutze hierfür ebenfalls Methodenreferenzen

Aufgabe

- Erstelle eine Klasse MathHelper mit einer statischen Methode `doubleTheValue(int value)`, die den Wert verdoppelt.
- Erstelle in der Main-Methode eine Liste an Zahlen
- Ersetze alle Elemente (`replaceAll`) mit dem verdoppelten Wert.
- Gebe nun jedes Element der Liste in der Konsole aus.
- Nutze Methoden-Referenzen!

Aufgabe

- Erstelle eine Liste von Strings.
- Erstelle eine zusätzliche Methode, welche die Länge ausgibt.
- Rufe nun innerhalb der Main-Methode die in Punkt 2 erstellte Methode mittels Method References auf.
- Programmiere das Gleiche auch mithilfe eines Lambdas.
- Welches ist von der Schreibweise leserlicher?

Ausblick

- Mit Lambdas und Method References kann noch viel mehr machen!
 - dafür braucht man einen näheren Einblick in funktionale Interfaces
 - mit Streams kann man verstärkt mit Lambdas und Method References arbeiten!