LAMBDAS UND METHODENREFERENZEN

VON INTERFACE ÜBER KLASSE BIS LAMBDA

```
public interface Calculator {

    double calculate(double x, double y);
}
```

Funktionales Interface

```
public class SumCalculator implements Calculator {

    @Override
    public double calculate(double x, double y) {
        return x + y;
    }
}
```

(Benannte) Klasse

```
public static void main(String[] args) {
    Calculator sumCalculator = new SumCalculator();

    Calculator divisionCalculator = new Calculator() {
        @Override
        public double calculate(double x, double y) {
            return x / y;
        }
    };

    Calculator subtractionCalculator = (x, y) -> x - y;

    System.out.println("3 + 4 = " + sumCalculator.calculate(3, 4));

    System.out.println("3 - 4 = " + subtractionCalculator.calculate(3, 4));
}
```

Anonyme Klasse

Lambda-Ausdruck

AUFBAU LAMBDA-AUSDRUCK

- (Parameter) -> Methodenbody

```
(x, y) -> x - y
```

```
(String input) -> System.out.println(input)
```

- Angabe von Parametertypen optional

```
(input) -> System.out.println(input)
```

```
input -> System.out.println(input)
```

- Klammer bei einem Parameter optional

```
input -> input + "yeah!"

input -> {
    String upperCaseInput = input.toUpperCase();
    System.out.println(upperCaseInput);
}
```

- Bei Methodenbody {} für komplexere Ausdrücke
  ggf. return-Statement notwendig

```
input -> {
    String upperCaseInput = input.toUpperCase();
    return upperCaseInput + "yeah!";
}
```

VERWENDUNG LAMBDAS

Wo kann ich Lambdas verwenden?
Überall wo ein funktionales Interface benötigt wird!

Funktionales Interface = Interface mit einer einzigen abstrakten Methode

```java
List<String> list = new ArrayList<>();
list.add("Hallo");
list.add("Welt");
list.add("!");

// Ohne Lambda
for (String input : list) {
    System.out.println(input);
}

// Mit Lambda
list.forEach(input -> System.out.println(input));
```

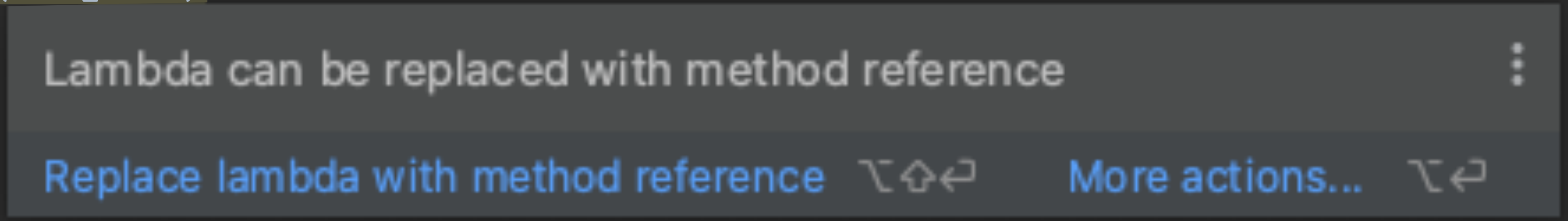
DIE WICHTIGSTEN FUNKTIONALEN INTERFACES

Name	Parameter	Rückgabe	Beispiel
Function	✅	✅	`Function<String, Integer> function = input -> input.length();`
Consumer	✅	❌	`Consumer<String> consumer = input -> System.out.println(input);`
Supplier	❌	✅	`Supplier<Integer> supplier = () -> new Random().nextInt();`
Runnable	❌	❌	`Runnable runnable = () -> System.out.println("Hello world!");`
Predicate	✅	✅ (boolean)	`Predicate<String> predicate = input -> input.contains("Cat");`
BiFunction	✅ ✅	✅	`BiFunction<Double, Double, Double> biFunction = (x, y) -> x - y;`

METHODENREFERENZEN

- Lambda-Ausdruck, um Methode aufzurufen
- Aufbau: Objekt/Klasse::Methode

```java
public class Example {

    public void foo(List<String> list) {
        // Mit Lambda
        list.forEach(input -> System.out.println(input));

        // Mit Methodenreferenz
        list.forEach(System.out::println);

        BiFunctionEx<String, Object, Integer> biFunctionWithLambda =
            (string, object) -> this.doSomeMagic(string, object);
        BiFunctionEx<String, Object, Integer> biFunctionWithMethodReference = this::doSomeMagic;

        Consumer<String> consumerWithLambda = name -> Example.greet(name);
        Consumer<String> consumerWithStaticMethodReference = Example::greet;

        Supplier<Object> supplierWithConstructor = Object::new;
    }

    private Integer doSomeMagic(String string, Object object) {
        return string.length() + object.hashCode();
    }

    private static void greet(String name) {
        System.out.println("Hello " + name);
    }
}
```



Lambda can be replaced with method reference

Replace lambda with method reference ⌥⇧↩ More actions... ⌥↩