Lambdas sind eine kompakte Möglichkeit, um anonyme Funktionen in Java zu schreiben. Sie wurden mit **Java 8** eingeführt und machen den Code kürzer und lesbarer.

1. **Klassische Implementierung mit einer konkreten Klasse**

Beispiel: Ein Interface Calculator mit einer Methode calculate(int a, int b), das in verschiedenen Klassen implementiert wird:

```
public interface Calculator {
    int calcualte(int a, int b);
}
```

```
public class Plus implements Calculator{
    @Override
    public int calcualte(int a, int b) {
        return a+b;
    }
}
```

**Nachteil:** Es erfordert viele Klassen, auch wenn die Methoden nur wenige Zeilen lang sind.

2. **Anonyme Klassen als kürzere Alternative**

Anstatt eine eigene Klasse zu schreiben, kann man eine anonyme Klasse verwenden:

```
Calculator c = new Calculator() {
    @Override
    public int calcualte(int a, int b) {
        return a + b;
    }
};
```

**Nachteil:** Der Code ist immer noch relativ lang.

3. **Lambda-Ausdrücke als noch kürzere Alternative**

```
Calculator minus = (int a, int b) -> a - b;
Calculator teilen = (int a, int b) -> a / b;
Calculator multiplizieren = (int a, int b) -> a * b;
```

### 4. Syntax von Lambda-Ausdrücken

Lambdas haben folgende allgemeine Struktur:

**(Parameter) -> { Anweisung(en) }**

| Beschreibung | Lambda-Ausdruck | Erläuterung |
|---|---|---|
| Ohne Parameter | `() -> System.out.println("Hello")` | Kein Eingabewert, nur eine Ausgabe |
| Mit einem Parameter | `(x) -> x * x` | Berechnet das Quadrat von x |
| Mit mehreren Parametern | `(a, b) -> a + b` | Berechnet die Summe von a und b |
| Mit Codeblock | `(a, b) -> { System.out.println(a + b); return a + b; }` | Mehrere Anweisungen innerhalb der {} |

### 5. Predicate als weiteres Beispiel

Das **Predicate<T>-Interface** ist ein **funktionales Interface**, das eine Bedingung testet und true oder false zurückgibt.

```
import java.util.function.Predicate;

public class PredicateExample {
    public static void main(String[] args) {
        Predicate<Integer> istGerade = x -> x % 2 == 0;

        System.out.println(istGerade.test(4)); // true
        System.out.println(istGerade.test(7)); // false
    }
}
```

### 6. Predicate als Methodenparameter

Man kann Predicate auch als Parameter in Methoden übergeben und sie dann innerhalb von Methoden nutzen. Bspw. Übergibt man //d-> d.getAge() < 4

```
static ArrayList<Dog> getSpecificDogs(ArrayList<Dog> dogs, Predicate<Dog>
predicate){ /
    ArrayList<Dog> dogsUnderGivenAge = new ArrayList<>();
    for(Dog d: dogs){
        if(predicate.test(d)){
            dogsUnderGivenAge.add(d);
        }
    }
    return dogsUnderGivenAge;
}
```

➔ In der Liste werden alle Hunde gespeichert, deren Alter unter 4 Jahren ist.