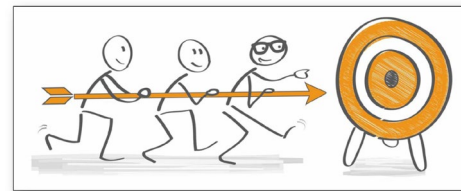


Ziele: Methoden, Parameterübergabe

4 Lektionen



TEKO
SCHWEIZERISCHE FACHSCHULE

▲ Lernfelder / Lerninhalte

- Methoden (Funktion) in Kotlin kennen und ihre Anwendung beherrschen

▲ Lernziele

- Sie kennen die Funktionsweise von Methoden und können selbständig Methoden programmieren
- Sie kennen die Parameterübergabe und können sie zweckmässig einsetzen.

Die Main-Methode

Eintrittspunkt der Programmausführung

- Die Ausführung beginnt bei der statischen Methode main()
- Es darf nur eine Methode mit dieser Signatur in einem Kotlin-Projekt geben

Keyword

Name der Methode
(Bezeichner)

Parameter der Methode («Formalparameter»)
bestehend aus Parametername und Datentyp

```
fun main(args: Array<String>) {  
    println("Hello World!")  
  
    // Try adding program arguments  
    println("Arguments: ${args.joinToString()}")  
}
```

Anweisungen (Statements)

Repetition

- Programm-“Befehle“ (verrichten eine Aufgabe)
- Werden mit **Zeilenumbruch oder Semikolon ;** abgeschlossen
- **Code-Blöcke { }** kapseln zusammengehörige Anweisungen (Blöcke werden nicht mit ; abgeschlossen)
- Können **Eingabe-Parameter** verlangen.
- Können **Rückgabewerte** liefern.
- Spezielle Anweisungen steuern den **Kontrollfluss** (Selektion, Iteration)

```
fun main(args: Array<String>) {  
    println("Hello World!")  
}
```

Methodenaufruf («Anweisung»)

Methodenname

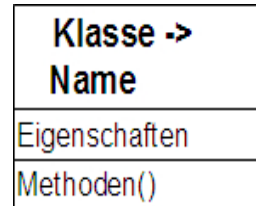
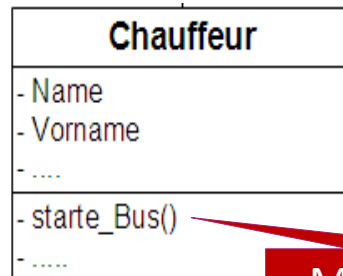
Parameter-Wert («Aktualparameter»)

*„**Methoden** (englisch method oder member function) sind in der objektorientierten Programmierung **Unterprogramme** in der Form von **Funktionen** oder **Prozeduren**, die das **Verhalten von Objekten** beschreiben und implementieren. Über die Methoden des Objekts können Objekte untereinander in Verbindung treten.“ [Wikipedia]*

Methoden – Zweck

„Funktionen“ in einer objektorientierten Sprache

- Wir haben gelernt: Variablen dienen der Wiederverwendung von Werten
- Analog dienen Methoden der **Wiederverwendung von Programmlogik** (mit unterschiedlichen Werten)
- Des Weiteren werden Sie auch zur **logischen Strukturierung** eines Programmes verwendet
- Aus objektorientierter Sicht gilt (vergleiche Klassendiagramm in UML):
 - Variablen definieren die Eigenschaften einer Klasse bzw. eines Objekts
 - Methoden dagegen **definieren das Verhalten einer Klasse bzw. eines Objekts**

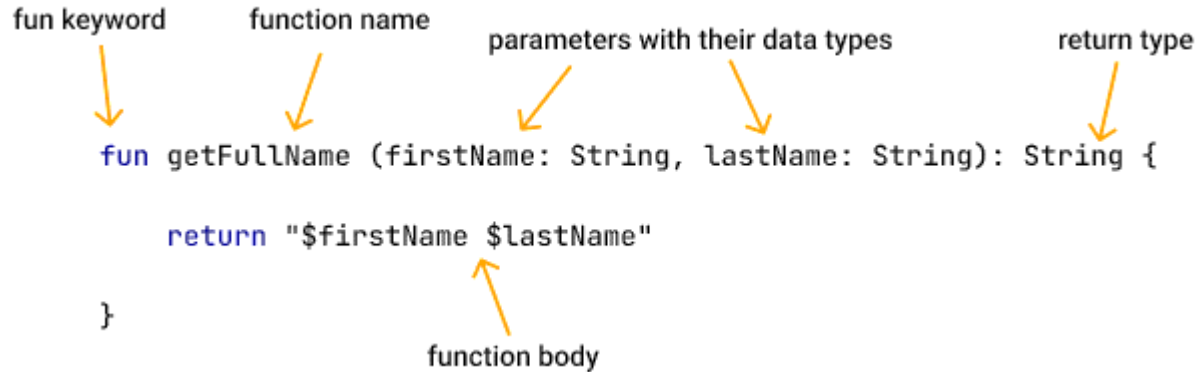


Klasse

Geordnete Darstellung der in der Analyse erkannten Eigenschaften und Methoden

Methode = Verhalten

Methoden – Aufbau bzw. Definition




```

fun keyword      function name      parameters with their data types      return type
    ↓              ↓                  ↓                ↓
fun getFullName (firstName: String, lastName: String): String {
    return "$firstName $lastName"
}
                ↑
            function body
  
```

- Die Definition einer Methode oder Funktion besteht aus folgenden Elementen
 - Schlüsselwort „fun“ (steht für „Funktion“)
 - Methodensignatur (Methodenname, Liste der Parameter mit Datentypen, optional Rückgabedatentyp)
 - Methodenrumpf („Body“) – falls Rückgabewert definiert: Letzte Codezeile beginnt mit Schlüsselwort „return“

Methoden – Verwendung bzw. Aufruf (1)

```
getFullName("Michael", "Smith")
```



```
fun getFullName (firstName: String, lastName: String): String {  
    return "$firstName $lastName"  
}
```

The diagram consists of two orange arrows. The first arrow originates from the string "Michael" in the function call above and points to the parameter "firstName" in the function definition below. The second arrow originates from the string "Smith" in the function call and points to the parameter "lastName" in the function definition.

Um eine zuvor definierte Methode im Programm zu verwenden, wird sie „aufgerufen“

Methoden – Verwendung bzw. Aufruf (2)

```
val fullName = getFullName("Michael", "Smith")
```

oder ohne Zuweisung

```
println(getFullName("Michael", "Smith"))
```

oder mit Objektname

```
val fullName = myCoolObject.getFullName("Michael", "Smith")
```

Um eine zuvor definierte Methode im Programm zu verwenden, wird sie „aufgerufen“:

- Zuweisung des Rückgabewertes an eine Variable, falls erforderlich und von Definition her möglich
- Objekt- oder Klassenname, falls es sich nicht um eine lokale oder eine eingebaute Funktion handelt
- Methodenname (ohne Schlüsselwort „fun“)
- Liste aller gemäss Definition erforderlicher Parameter vom richtigen Datentyp (ohne Typangabe)

Anweisungs-Blöcke (Block Statements)

- Anweisungsliste: eine oder mehrere Anweisungen in Folge
- Blockanweisung: eine Anweisungsliste innerhalb von geschweiften Klammern { ... }
→ Wird v.a. für Methoden, Kontrollfluss (Selektion, Iteration) und z.T. Spezialkonstrukte verwendet

```
fun myFunc() { // Start der Methoden-Definition "myFunc"
  F()          // Anweisung (Aufruf der Methode "F")
  G()          // Anweisung (Aufruf der Methode "G")
  if (true)    // Selektion
  {            // Start eines Blocks
    H()        // Anweisung (Aufruf der Methode "H")
    I()        // Anweisung (Aufruf der Methode "I")
  }            // Ende des Blocks
}              // Ende der Methode
```

Aufgabe 3.1



TEKO-OOP-Aufgabe-3-1.pdf

Aufgabe 3.2



TEKO-OOP-Aufgabe-3-2.pdf

Variablen-Gültigkeit

« Scope »

Variablen sind nur ab dem Zeitpunkt ihrer Deklaration und nur innerhalb des deklarierenden Code-Blocks gültig (“Scope”).

- In der Initialisierung einer Schleife deklarierte (Zähl-) Variablen sind nur innerhalb des Schleifenkörpers gültig (“Scope”).

```
for (i in 1..10)
```

```
var i : Int  
for (i in 1..10)
```

- Instanz-Variablen sind nur innerhalb des Objekts (der Instanz der Klasse gültig)
- **Lokale Variablen sind nur innerhalb des Code-Blocks bzw. der Methode gültig**

Parameterübergabe: Wert- und Referenzsemantik

Zwei Typarten mit grundsätzlich unterschiedlicher Verhaltensweise

Datentypen mit Wertsemantik (value types)

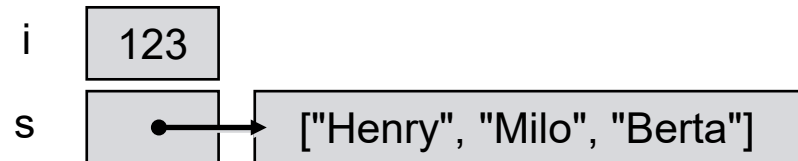
- enthalten direkt Daten / Werte → Zuweisung oder Parameterübergabe kopiert Daten / Werte
- z.B. Zahlen (Int, Long, Float, Double), Zeichen (Char), Bool'sche Werte (Boolean)

Datentypen mit Referenzsemantik (reference types)

- enthalten Referenz (Speicheradresse, „Pointer“) auf Objekte → Zuweisung kopiert Referenz
- z.B. Klassen (class), Listen (Arrays, Collections) etc.

```
var i = 123
```

```
var l = listOf("Henry", "Milo", "Berta")
```





Fragen?