

# Einführung Methoden

Programmiermethodik

Lukas Kaltenbrunner, Simon Priller

Universität Innsbruck

# Unterprogramme

- Die Teile eines Programms können auch an einer anderen Stelle als Unterprogramm formuliert werden.
- Unterprogramme können in einer Anweisungsfolge aufgerufen werden.
  - Verzweigung zum Unterprogramm
  - Abarbeitung des Unterprogramms
  - Nach der Abarbeitung Rückkehr zur Aufrufstelle
  - Eventuell Ergebnis zurückgeben

# Unterprogramme in C bzw. Java

- In C spricht man nur von Funktionen.
- In Java spricht man nur von Methoden.
  - Statische Methoden
    - Gehören zur Klasse
    - Gekennzeichnet mit dem Schlüsselwort `static`
  - Objektbezogene Methoden
    - Werden bei der Einheit zu Objektorientierung besprochen!

## Methode

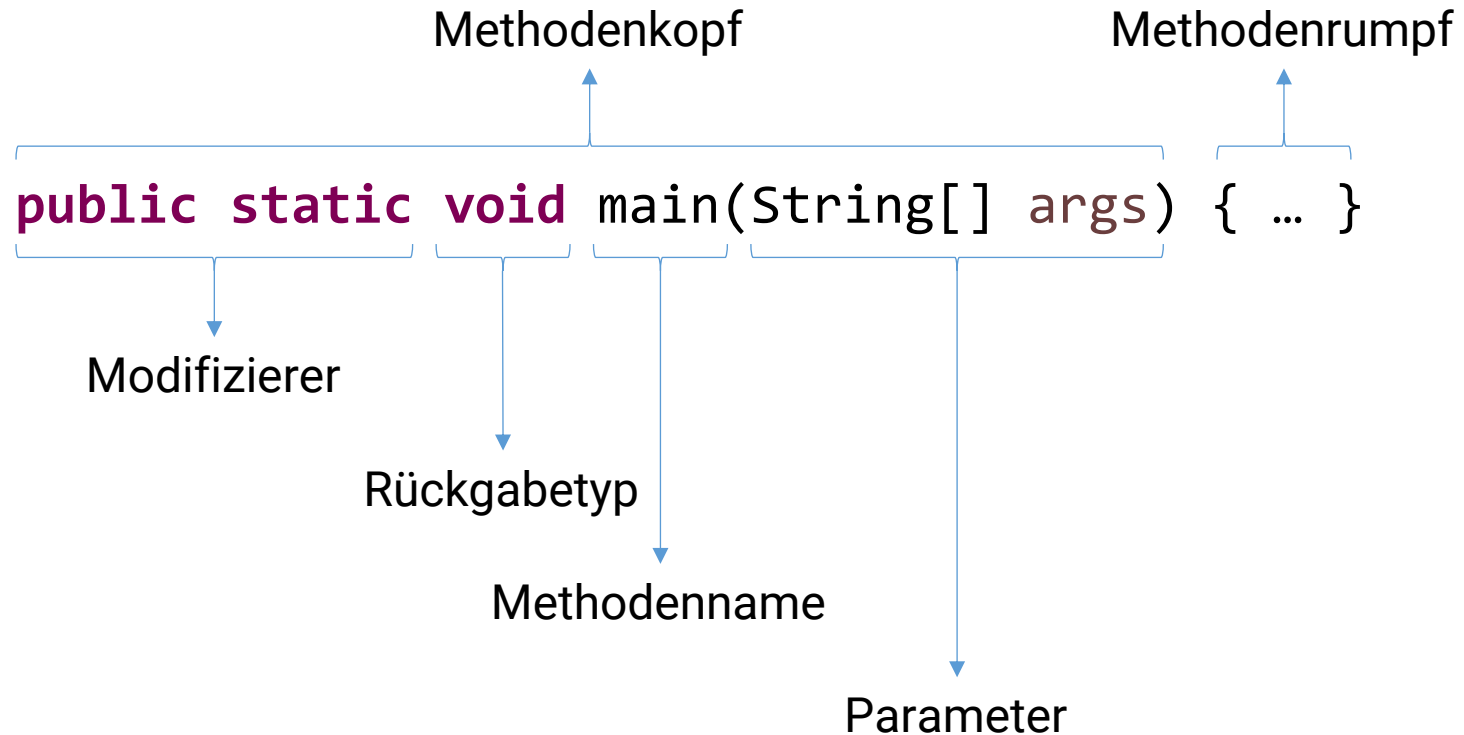
### Methodenkopf

- Rückgabetyp, Methodenname und eventuell Parameter

### Methodenrumpf

- ist ein Block
- Enthält Variablendeklarationen und Anweisungen
- Kann auch leer sein (wird noch erklärt)

# Methoden (2)



# Rückgabewert

- Typ des Rückgabewerts steht vor dem Methodennamen.
- Methoden mit Rückgabewert
  - Rückgabewert wird durch eine return-Anweisung zurückgegeben.
  - Eine return-Anweisung muss in jedem Programmpfad in der Methode vorhanden sein.
  - Der Typ des Ausdrucks, welcher in der return-Anweisung verwendet wird, muss zuweisungskompatibel mit dem Typ des Rückgabewerts sein.
- Methoden ohne Rückgabewert (Rückgabotyp `void`)
  - Benötigen keine return-Anweisung.
  - Durch eine leere return-Anweisung ( `return;` ) kann die Methode vorzeitig verlassen werden.

# Methodenname

- Der Name einer Methode sollte ausdrücken, was die Methode leistet.
- Er sollte mit einem Verb beginnen und klein geschrieben werden.
  - Methoden, die den Wert eines Felds abfragen bzw. setzten, sollten mit dem Wort `get` bzw. `set` beginnen.
  - Methoden, die einen `boolean`-Wert zurückgeben, sollten mit dem Wort `is` beginnen.
  - Methoden, die ein Objekt in einen anderen Typ bzw. ein anderes Format `F` umwandeln, sollten als `toF` bezeichnet werden.
- Falls der Name aus mehreren Worten besteht, sollte der Anfangsbuchstabe jedes Folgewortes groß geschrieben werden.
- Beispiele
  - `add`, `append`
  - `getArea`, `setArea`
  - `isEmpty`, `isBlank`
  - `toString`, `toArray`, `toLowerCase`

## Formale Parameter

- Werden im Methodenkopf deklariert
- Werden innerhalb der Methode wie lokale Variablen behandelt

## Aktuelle Parameter

- Beim Aufruf der Methode wird jeder formale Parameter durch je einen aktuellen Parameter initialisiert



# Parameterübergabe-Mechanismen

- Der Parameterübergabe-Mechanismus bestimmt wie die Parameter vom Methodenaufruf an die aufgerufene Methode übergeben werden.
- Beispiele für Parameterübergabe-Mechanismen
  - Call-by-value
    - Formaler Parameter wird mit dem Wert des aktuellen Parameter initialisiert.
    - Änderungen am formalen Parameter ändern den aktuellen Parameter nicht.
  - Call-by-reference
    - Eine Referenz wird implizit übergeben.
    - Änderungen am formalen Parameter ändern den aktuellen Parameter.
    - Z.B in C simuliert durch das Übergeben von Zeigern.

# Parameterübergabe in Java

- In Java wird nur call-by-value verwendet:
  - Für primitive Datentypen wird eine Kopie des Wertes übergeben.
  - Für Referenztypen wird eine Kopie der Referenz auf das Objekt übergeben.
- Beim Aufruf einer Methode werden die aktuellen an die formalen Parameter durch folgende Aktionen übergeben:
  - Die Ausdrücke der aktuellen Parameter werden berechnet.
  - Der Wert dieser Ausdrücke wird den entsprechenden formalen Parametern zugewiesen.
    - Die Typkompatibilität für den Parameterkontext muss eingehalten werden!
  - Durch die Zuweisung wird eine **Kopie** erzeugt!

# Beispiel (Primitiver Typ)

```
public class PrimitiveTypesMethodApplication {  
    public static int pow2(int base) {  
        base = base * base;  
        return base;  
    }  
  
    public static void main(String[] args) {  
        int value = 4;  
        System.out.println(value);  
        System.out.println(pow2(value));  
        System.out.println(value);  
    }  
}
```

formaler Parameter base

aktueller Parameter value

Ausgabe:

4

16

4

# Beispiel (Referenztyp)

```
public class ReferenceTypeMethodApplication {  
    public static void pow2(int[] bases) {  
        for (int i = 0; i < bases.length; ++i) {  
            bases[i] = bases[i] * bases[i];  
        }  
    }  
  
    public static void main(String[] args) {  
        int[] values = {2, 4, 10};  
        System.out.println(values[1]);  
        pow2(values);  
        System.out.println(values[1]);  
    }  
}
```

Ausgabe:

4

16

# Lokale Variablen

- Methoden können neben Anweisungen auch lokale Deklarationen enthalten.

```
public static void method() {  
    int x;  
    short y;  
    ...  
}
```

- Deklariert die zwei lokalen Variablen x und y
  - Dürfen nur in dieser Methode verwendet werden
  - Beim Aufruf wird der Speicherplatz angelegt
  - Am Ende der Methode wird der Speicherplatz wieder freigegeben.
- Jede lokale Variable muss initialisiert werden, bevor auf den Wert der Variable zugegriffen wird (definite assignment).

# Klassenvariablen

- Variablen können auch außerhalb von Methoden deklariert werden.
- Werden diese Variablen mit dem Keyword `static` gekennzeichnet, wird von Klassenvariablen gesprochen.
- Grundregel: Variablen immer möglichst lokal deklarieren!

```
public class ClassVariables {  
    static int a;  
    static double b;  
    public static void method() {...}  
    ...  
}
```

- `a` und `b` sind Klassenvariablen
  - Können in allen Methoden der Klasse `ClassVariables` benutzt werden
  - Werte existieren über Methodenaufrufe hinweg.
  - Näheres dazu in VO zu Objektorientierung.

# Lebensdauer

- Zeitintervall, in dem die Variable zur Laufzeit existiert.
- Lokale Variablen
  - werden angelegt, wenn ihr Block betreten wird
  - werden freigegeben, wenn ihr Block verlassen wird
- Klassenvariablen
  - werden angelegt, wenn die Klasse geladen wird
  - werden freigegeben, wenn die Klasse entladen wird

# Sichtbarkeit

- Programmstück, in dem auf die Bezeichner zugegriffen werden kann.
- Bezeichner in einer Klasse (z.B. Klassenvariablen, Methoden)
  - Ist in der ganzen Klasse gültig
  - Variablen können in Methoden neu deklariert werden. Durch die neue Deklaration wird die ursprüngliche Variable überdeckt.
- Bezeichner in einem Block (z.B. lokale Variablen)
  - Gültig von der Deklaration bis zum Ende des Blocks, in dem er deklariert wurde.
  - Blöcke können verschachtelt werden
  - In einem geschachtelten Block kann aber keine erneute Deklaration erfolgen!



# Überladen von Methoden

- In Java können Methoden überladen (engl. overloading) werden.
- Eine Methode ist überladen, wenn Methoden einer Klasse mit unterschiedlichen Parameterlisten den selben Namen haben.
  - Unterschiedliche Typen
  - Unterschiedliche Reihenfolge der Typen
  - Unterschiedliche Anzahl der Typen
- Basierend auf den Typen der Parameter wird beim Aufruf einer überladenen Methode vom Compiler die richtige Methode ausgewählt.
- Überladen gibt es auch bei Operatoren:
  - + hat unterschiedliche Bedeutung für int, double, String ....

# Signatur einer Methode in Java



- Alle Methoden einer Klasse müssen unterschiedliche Signaturen haben.
- Methodenname und Liste der Typen der formalen Parameter (wenn vorhanden) bilden die **Signatur** einer Methode!
  - **Rückgabotyp gehört nicht dazu!**
  - Die Namen der Parameter spielen keine Rolle, nur die Typen und ihre Reihenfolge!
  - Beispiele:

```
public static int method1(int param1) {...}
Signatur: method1(int)
public static void method2(double param1, char param2) {...}
Signatur: method2(double, char)
```

# Beispiel (Überladen)

```
public class OverloadingApplication {  
    public static void myPrint(int i) {  
        System.out.println("myPrint(int): " + i);  
    }  
    public static void myPrint(double d) {  
        System.out.println("myPrint(double): " + d);  
    }  
    public static void main(String[] args) {  
        myPrint(10);  
        myPrint(10.0);  
    }  
}
```

**Ausgabe:**

**myPrint(int): 10**

**myPrint(double): 10.0**

# Auflösen eines Methodenaufrufs

- Für das Bestimmen der Methode, welche durch einen Methodenaufruf aufgerufen wird, werden durch den Compiler drei Schritte durchgeführt.
  1. Basierend auf dem Methodenaufruf muss bestimmt werden in welchem Typ nach dieser Methode gesucht wird.
  2. Auswahl der Methode, welche verfügbar und anwendbar ist.
    - Sofern mehrere überladene Methoden anwendbar sind, wird durch den Compiler die spezifischste Methode ausgewählt.
    - Spezifischste Methode
      - Intuition: Eine Methode m1 ist spezifischer als die Methode m2, wenn jeder Aufruf der Methode m1 durch m2 behandelt werden könnte.
    - Wird durch den Compiler keine spezifischste Methode gefunden, ist der Aufruf mehrdeutig und es kommt zu einem Kompilierfehler.
  3. Überprüfung, ob die ausgewählte Methode passend ist.


# Beispiel (Auswahl überladener Methoden)

```
public class MostSpecificMethodApplication {  
    public static void myPrint(short s) {  
        System.out.println("myPrint(short): " + s);  
    }  
    public static void myPrint(int i) {  
        System.out.println("myPrint(int): " + i);  
    }  
    public static void myPrint(double d) {  
        System.out.println("myPrint(double): " + d);  
    }  
    public static void main(String[] args) {  
        byte b = 5;  
        myPrint(b);  
        myPrint('A');  
    }  
}
```

**Ausgabe:**  
myPrint(short): 5  
myPrint(int): 65

# Beispiel (Mehrdeutiger Methodenaufruf)

```
public class AmbiguousMethodsApplication {  
    public static void myPrint(float f, double d) {  
        System.out.println("myPrint(float, double): " + f + ", " + d);  
    }  
    public static void myPrint(double d, float f) {  
        System.out.println("myPrint(double, float): " + d + ", " + f);  
    }  
    public static void main(String[] args) {  
        myPrint(4.2f, 5.5f);  
    }  
}
```



## Ausgabe des Compilers:

java: reference to myPrint is ambiguous  
both method myPrint(float,double) in AmbiguousMethodsApplication and  
method myPrint(double,float) in AmbiguousMethodsApplication match

# Beispiel (Rückgabebetyp)

```
public class IncompatibleTypesApplication {  
    public static String method(int i) {  
        return i + "";  
    }  
    public static int method(long l) {  
        return l;  
    }  
    public static void main(String[] args) {  
        int result = method(5);  
    }  
}
```



**Ausgabe des Compilers:**

java: incompatible types: java.lang.String cannot be converted to int

# Vordefinierte statische Methoden

- Java bietet etliche statische Methoden (Klassenmethoden) in speziellen Klassen an.
- Typischer Aufruf
  - `ClassName.MethodName(Parameter)`
  - Beispiel  
`Math.sqrt(10.0);`
- Manchmal auch in der Form
  - `ClassName.FieldName.MethodName(Parameter)`
  - Beispiel  
`System.out.println(x);`



# Beispiele für Klassen

- **System**

- `System.out.println()` etc.

- **Math**

- `Math.random()`, `Math.sqrt(10.0)` etc.

- **Arrays**

- Hilfsklasse für Arrays
- Nützliche überladene Methoden (hier für `int`):
  - `binarySearch(int[] a, int key)`
  - `equals(int[] a, int[] a2)`
  - `deepEquals(Object[] a1, Object[] a2)`
  - `fill(int[] a, int val)`
  - `sort(int[] a)`

# Verwendung von Methoden anderer Klassen

- Bestimmte Klassen bzw. Pakete werden automatisch geladen.
  - Die Klassen aus `java.lang` werden automatisch importiert (z.B. Klassen `Math` und `String`)
- Andere Klassen bzw. Pakete müssen importiert oder mit dem voll qualifizierten Namen angesprochen werden.
  - Voll qualifizierter Name (bei Verwendung)  
`java.util.Arrays.sort(x);`

- Import-Deklaration (am Anfang der Datei)

```
import java.util.Arrays;  
import java.util.*;           // all classes from java.util  
import static java.lang.Math.min; // static import  
import static java.lang.Math.*;  // all methods
```

# Dokumentation der Klassen

- Startpunkt (Java 17)
  - <https://docs.oracle.com/en/java/javase/17/docs/api/index.html>
- Bereitgestellte Informationen (Beispiele)
  - In welchem Paket befindet sich eine Klasse?
    - Die einzelnen Klassen eines Pakets können ausgewählt werden.
  - Für eine Klasse
    - Beschreibung der Klasse
    - Liste der Methoden
  - Für jede Methode
    - Kurzbeschreibung
    - Längere Beschreibung (Parameter, Ausnahmen etc.)