LMU LUDWIG-MAXIMILIANS-UNIVERSITÄT MÜNCHEN

Java Grundlagen 3

Bioinformatik Bachelor

Propädeutikum WS 2019/2020

5. November 2019

Christian Hoffmann

Outline

- Methoden
- Standard Library Methods
- Scopes

Methoden

Def: Eine Methode ist ein Code-Block, der nur ausgeführt wird wenn man ihn aufruft.

Methoden werden benutzt, um gewisse Aktionen auszuführen, und werden auch Funktionen genannt

Warum benutzt man Methoden? Wiederverwendbarer Code: Schreibe den Code einmal, benutze ihn öfter

Erstellen einer Methode:

```
static void myMethod() {
  // code to be executed
}
```

Erklärung:

- `myMethod()` ist der Name der Methode
- `void` heißt, dass die Methode keine Rückgabewert besitzt
- `static` nötig, um die Methode über die Klasse aufrufen zu können (einfach mal so hinnehmen, kommt erst in ein paar Wochen genauer)

Parameter

Einer Methode können Informationen in Form von sog. „Parametern“ übergeben werden.

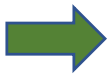
Parameter fungieren als Variablen innerhalb der Methode und werden nach dem Methodennamen innerhalb der Klammer spezifiziert. Man kann mehrere – durch Komma getrennte – Parameter spezifizieren.

Beispiel:

```
public class MyClass {

  public static void main(String[] args) {
    myMethod("DNA","Sequenz");
    myMethod("RNA","Reihe");
    myMethod("Aminosäuren","Abfolge");
  }

  static void myMethod(String para1, String para2) {
    System.out.println(para + "-" + para2);
  }
}
```



```
DNA-Sequenz
RNA-Reihe
Aminosäuren-Abfolge
```

Return Value

Wenn gewollt wird, dass eine Methode einen Rückgabe-Wert liefert, kann man Datentypen (wie int, String, int[], etc.) anstatt von **void** benutzen. Wichtig: „return“ Key-Wort nicht vergessen!

```
public class MyClass {

  public static void main(String[] args) {
    System.out.println(myMethod(3));
  }

  static int myMethod(int x) {
    return 5 + x;
  }
}
```

```
8
```

Beispiel: Return Value in Variable

```
public class MyClass {
  static int myMethod(int x, int y) {
    return x + y;
  }

  public static void main(String[] args) {
    int z = myMethod(5, 3);
    System.out.println(z);
  }
}
```

```
8
```

Beispiel mit Verzweigung

```
public class MyClass {

  static void checkAge(int age) {
    if (age < 18) {
      System.out.println("Unter 18");
    } else {
      System.out.println("Über 18");
    }
  }
  public static void main(String[] args) {
    checkAge(20);
  }
}
```

Standard Library Methods

Die Standard-Bibliotheksmethoden sind bereits implementierte Methoden in Java, welche immer verfügbar sind. (Manchmal ist zuvor der Import von gewissen Librarys nötig!)

Beispiele:

- System.out.print()
- Math.sqrt()

```
public class Numbers {

  public static void main(String[] args) {
    System.out.println("Wurzel von 4 ist: " + Math.sqrt(4));
  }
}
```

```
Wurzel von 4 ist: 2
```

Standard Library Methods

Es gibt sehr, sehr viele predefined methods:

Examples String Methods:

String str1 = “Binfo“; String str2 = “Tutorium“

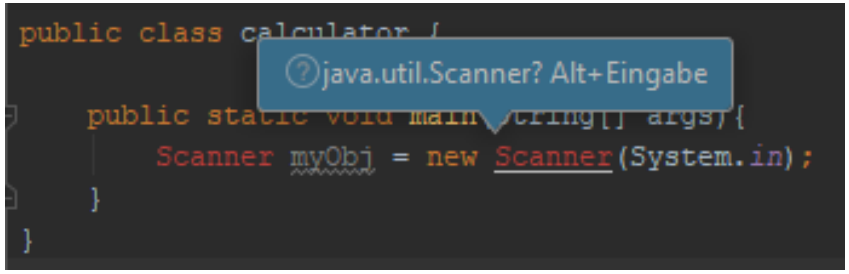
- charAt(): returns a character value by index: str1.charAt(0) → “B“
- length(): returns the length of a string: str1.length() → 5
- concat(): concatenates two string: str1.concat(str2) → “BinfoTutorium“
- endsWith(): str1.endsWith(“fo“) → true

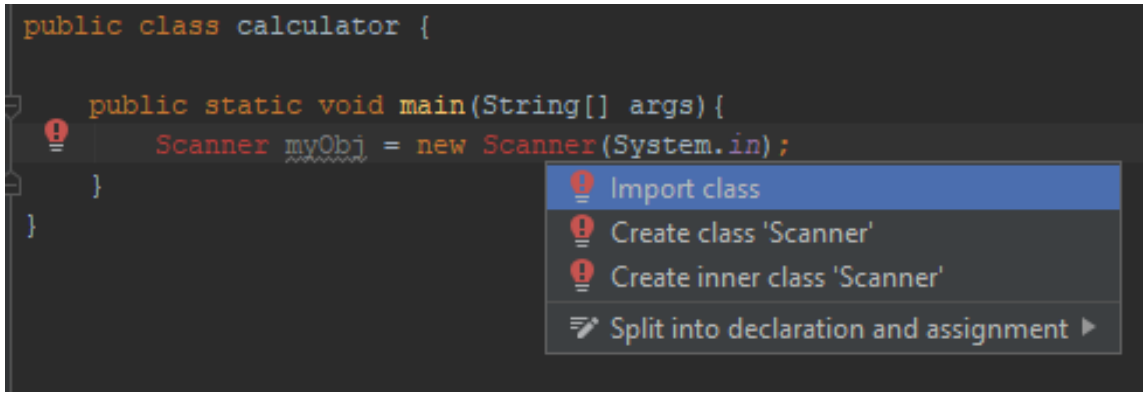
Examples Number Methods:

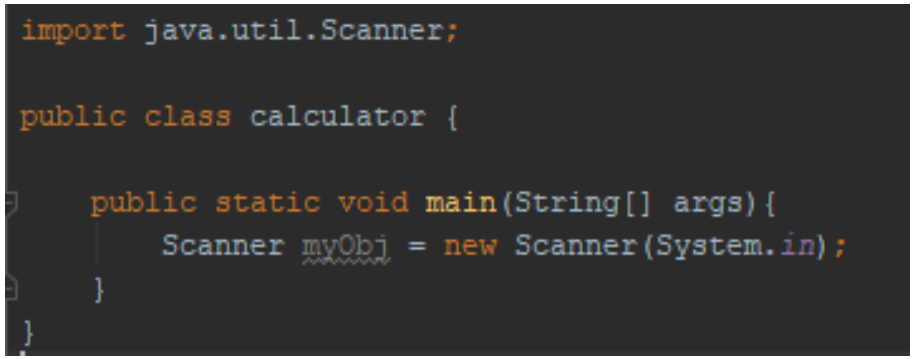
double wert = 4.156;

- round: rounds the value to the nearest Integer: Math.round(wert) → 4
- random(): returns a value between 0.0 and 1.0: Math.random() → 0.561834
- parseInt(String x): parses the parameter into a int: Integer.parseInt("42") → 42

Import von Libraries







Scopes

„Scope“ einer Variable = „Gültigkeitsbereich“ einer Variable → Teil des Programms in dem auf die Variable zugegriffen werden kann.

Es gibt folgende Kategorien:

- Member Variables (Class Level Scope)
- Local Variables (Method Level Scope)
- Loop Variables (Block Scope)

Class Level Scope

Variablen mit dieser Reichweite müssen in der Klasse (außerhalb einer Methode) deklariert werden.

Auf sie kann direkt überall in der Klasse zugegriffen werden!

```
public class Test {
  static int a = 5;

  public static void main(String[] args) {
    System.out.println(a);
    myMethod();
  }

  public static void myMethod() {
    a=3;
    System.out.println(a);
  }
}
```

```
5
3
```

Method Level Scope

In einer Methode deklarierte Variablen haben method level scope und auf sie kann nicht von außerhalb der Methode zugegriffen werden!

```
public class Test {
  void method1() {
    int a=1;
  }

  void method2() {
    System.out.println(a);                    → nicht möglich!
  }

}
```

Diese Variablen existieren nicht mehr, wenn die Methode durchgelaufen ist!

Block Scope

Eine Variable deklariert innerhalb von geschweiften Klammern { } in einer Methode zählt nur innerhalb dieser Klammern!

```
public class Test {
  public static void main(String[] args) {
    {
      int x=10;
    }
    System.out.println(x);     → nicht möglich!
  }
}
```

Block Scope

Das gleiche gilt für Variablen innerhalb einer Loop!

```
public class Test {
  public static void main(String[] args) {

    for (int i=0; i<3; i++) {
      System.out.println(i + ". Durchlauf");
    }

    System.out.println(i);   → nicht möglich!
}
```

Zusammenfassung Scopes

- Generell definiert ein Set geschweifter Klammern {} einen Gültigkeitsbereich
- In Java kann man normalerweise auf eine Variable zugreifen, solange diese innerhalb des gleichen Klammerpaares definiert wurde, in dem wir unseren Code schreiben
- Jede Variable einer Klasse, die außerhalb einer Methode definiert wurde kann normalerweise von allen Methoden der Klasse benutzt werden