RWTH AACHEN UNIVERSITY

„Good code is its own best documentation.“
(Steve McConnell)

Java – Cheatsheet

Java Grundlagen

Aufbau einer Klasse

*Textdatei „**MeineKlasse**.java“*
(muss dem Namen der Klasse entsprechen!)

Name der Klasse

```
public class MeineKlasse {

    public static void main(String[] args) {

        String uni = "RWTH";
        System.out.println(uni);

    }

}
```

*main()-**Prozedur***

*Rumpf (**body**) der Prozedur*

Kommentare

Mehrzeiliger Kommentar (beginnt mit / und endet mit */)*

```
...
/*
 * Main-Prozedur: Einstiegspunkt (Die Ausführung des Programms startet hier)
 */

public static void main(String[] args) {
        // String mit dem Inhalt "RWTH" erstellen
        String uni = "RWTH";
        System.out.println(uni); // Gibt den Wert der Variable (= RWTH) aus
}
...
```

Zeilenkommentar (beginnt mit //, gilt bis zum Zeilenende)

Datentypen

Datentyp	Was speichert der Typ?	Code-Beispiel
`String`	Zeichenketten (Text)	`String titel = "Vorkurs Informatik";`
`char`	Ein einzelnes Zeichen (z.B. 'a' oder '?')	`char geschlecht = 'w';`
`int`	Ganze Zahlen (z.B. 98, -172 oder 0)	`int alter = 21;`
`double`	Eine Kommazahl (z.B. 1.25 oder -177.3895) Achtung: In Java muss ein Punkt anstelle des Kommas verwendet werden!	`double hoehe = 175.6;`
`boolean`	Wahrheitswert `true` (wahr) oder `false` (falsch)	`boolean student = true;`

Variablen

Was will man tun?	Beispielhafte Umsetzung in Java

Deklaration und Initialisierung
Hier: Für eine `String`-Variable
	`String uni = "RWTH";`
	`Alternativ:`
	`String uni; // Deklarieren.`
	`uni = "RWTH"; // Wert zuweisen.`

Deklaration von mehreren Variablen Hier: Für Variablen vom Datentyp `int`	`int k, l, n;`
Deklaration von mehreren Variablen mit einzelner Initialisierung Hier: Initialisierung mehrerer Variablen (`k`, `l`, `n`) vom Datentyp `int`, wobei nur `k` ein Wert zugewiesen wird	`int k=7, l, n;`
Zuweisung Hier: Zuweisung des Werts `5` an die Variable `n`	`n = 5;`
Zuweisung des Ergebnisses einer Rechnung Hier: Das Ergebnis von `n+k` wird der Variable `l` zugewiesen	`l = n + k;`
Zuweisung des Ergebnisses einer Rechnung mit Klammersetzung (Addition `+`, Subtraktion `-`, Multiplikation `*`, Division `/`)	`n = l*2 - (k + 3 / n);`

Arrays (Felder)

Was will man tun?	Beispielhafte Umsetzung in Java
Deklaration eines Arrays Hier: Array vom Datentyp `double`	`double[] daten;` `// Der Wert von daten ist null.`
Deklaration und Initialisierung eines Arrays Hier: Array vom Datentyp `double` mit der Länge `5`	`double[] daten = new double[5];`
Zuweisung eines Werts an eine Stelle im Array Hier: Zuweisung an die erste und letzte Stelle des Arrays `daten`	`daten[0] = 3.0;` `daten[4] = 7.0;`
Zuweisung von einer Stelle im Array an eine Variable Hier: Zuweisung von der zweiten Stelle im Array `daten` an eine Variable `m` vom Typ `double`	`double m = daten[1];`
Länge eines Arrays	`int datenLaenge = daten.length;`

Ausgabe

Was will man tun?	Beispielhafte Umsetzung in Java
Ausgabe einer Zeichenkette Hier: Ausgabe von „Hallo Welt“	`System.out.print("Hallo ");` `System.out.print("Welt");`
Ausgabe einer Zeichenkette mit anschließendem Zeilenumbruch Hier: Ausgabe von „Hallo“ und (in einer neuen Zeile) „Welt“	`System.out.println("Hallo");` `System.out.println("Welt");`
Ausgabe eines Variablenwertes	`System.out.println(n);`
Zeichenketten verbinden Hier (für `uni="RWTH"` und `n=5`): (1) Willkommen an der RWTH! (2) Das Ergebnis ist 8	`"Willkommen an der " + uni + "!"` `"Das Ergebnis ist: " + (n+3)`

Bedingungen

Was will man tun?	Beispielhafte Umsetzung in Java

Bedingte Ausführung
Hier: Ausführung von *etwas*, wenn der Wert von n größer oder gleich 5 ist.
	`if (n >= 5) {`
	`  // Etwas`
	`}`

Bedingte Ausführung mit alternativer Ausführung bei nicht erfüllter Bedingung	`if (n >= 5) {`
	`  // Etwas`
	`} else {`

Hier: Ausführung von *etwas*, wenn der Wert von n größer oder gleich 5 ist, sonst wird *etwas anderes* ausgeführt.	`  // Etwas anderes` `}`

Werte Vergleichen
(`<` kleiner, `<=` kleiner gleich, `==` gleich, `!=` ungleich `>` größer, `>=` größer gleich)
Hier: (1) Ist n gleich `0`? (2) Ist c ungleich 'a'? (3) Ist d kleiner oder gleich 5,2?
	`n == 0`
	`c != 'a'`
	`d <= 5.2`

Strings Vergleichen Hier: Ist s gleich "Hallo Welt"?	`s.equals("Hallo Welt");`
Negation einer Bedingung Hier: Ist n nicht größer als 5?	`!(n > 5)`
Verknüpfung von Bedingungen (`&&` logisches „und“, `||` logisches „oder“) Hier: Ist n zwischen `0` und 5?	`n >= 0 && n <= 5`

Klammersetzung in einer komplexeren Bedingung	`if (!(n < 4) || (l != n + k && daten[0] == m)) {`
	`  // Etwas`
	`}`

Schleifen

While-Schleife

```
...
while ( n < MAX_N ) {
    n = n*2;
}
...
```

***Laufbedingung**: Schleife wird nur wiederholt, solange diese erfüllt (wahr) ist*
Hier: Schleife wird solange ausgeführt, wie n kleiner ist als MAX_N ist
*Rumpf (**body**) der Schleife: Das was wiederholt wird*

For-Schleife

```
...
for ( int i=0 ; i<10||i<n ; i=i+1 ) {
    System.out.println(i + " zum Quadrat ist " + i*i);
}
...
```

***Initialisierung**: Festlegung der Startsituation (i.d.R. Deklaration und Initialisierung der Zählvariablen)*
***Schrittweite**: Veränderung der Zählvariablen nach jedem Schritt*
(Hier gleichbedeutend zu `i=i+1` sind `i++` sowie `i+=1`)
***Laufbedingung**: Schleife wird nur wiederholt, solange diese erfüllt (wahr) ist*
*Rumpf (**body**) der Schleife*

Effekt: Es werden die Quadratzahlen für 0 bis 9 ausgeben. Also „0 zum Quadrat ist 0“, „1 zum Quadrat ist 1“, „2 zum Quadrat ist 4“ usw. bis „9 zum Quadrat ist 81“.

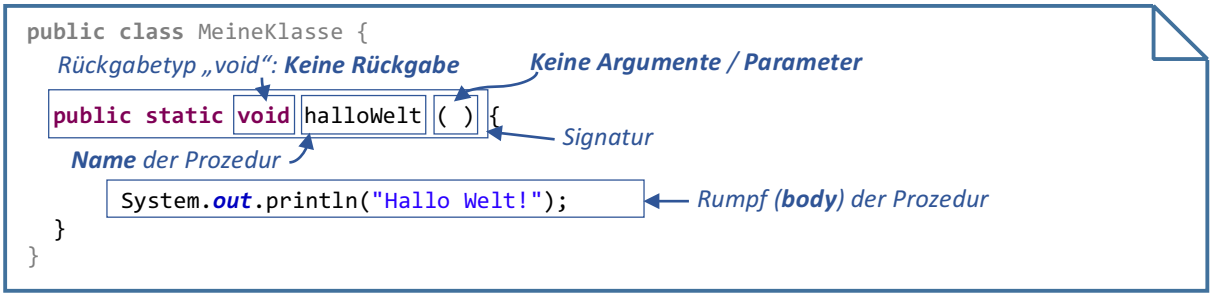
```
...
for ( int i=0 ; i<daten.length ; i=i+2 ) {
    daten[i] = daten[i] / 2;
}
...
```

***Initialisierung**: Starte bei der ersten Stelle (Index = 0!)*
***Schrittweite**: Immer zwei Stellen weiter*
***Laufbedingung**: Solange Zählvariable kleiner als Anzahl der Stellen in `daten`*
*Rumpf (**body**): Wert der Stelle durch zwei teilen und der Stelle wieder zuweisen*

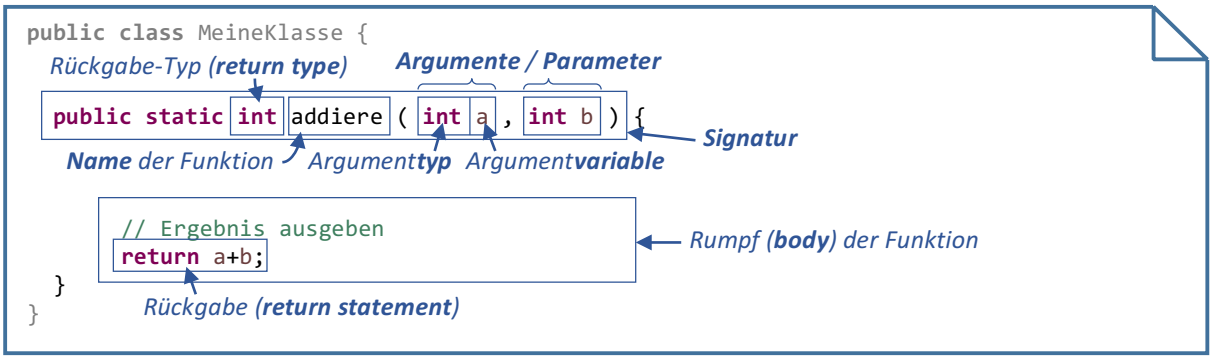
Effekt: Der Wert an jeder einzelnen Stelle des Arrays `daten` wird durch zwei geteilt.

Prozeduren und Funktionen

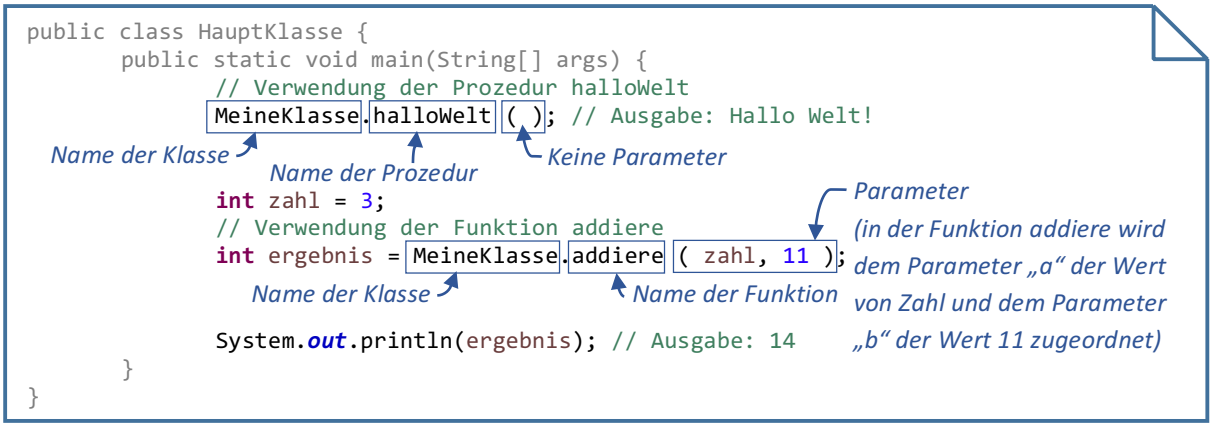
Einfache Prozedur ohne Parameter und Rückgabe



Prozedur mit Parametern und Rückgabe (Funktion)



Verwendung von Prozeduren / Funktionen



Klassen, Objekte und Methoden

Einfache Klasse

Fahrzeug.java

```
public class Fahrzeug {

        public String hersteller;
        public String modell;

        public Fahrzeug () {
                hersteller = "RWTH Motors";
                modell ="SuperA";
        }

        public void fahrzeugAusgeben() {
                System.out.println(hersteller + " " + modell);
        }
}
```

Klassenname

Attribute

Standard-Konstruktor *(keine Parameter)*

Bezeichner der Attribute

Methode

MeineKlasse1.java

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

                Fahrzeug f = new Fahrzeug();

                f.fahrzeugAusgeben(); // Ausgabe: RWTH Motors SuperA

                f.modell = "Karman";

                f.fahrzeugAusgeben(); // Ausgabe: RWTH Motors Karman
        }
}
```

new + Konstruktor*, um Objekt zu erzeugen*

Objekt *vom Typ Fahrzeug* ***erstellen***

Variable deklarieren

Methode *des Objekts der Klasse Fahrzeug* ***aufrufen***

Variable *des Objekts* ***ändern***

Vererbung mit abstrakter Oberklasse

Person.java

```
public abstract class Person {
```
Abstrakte Klasse — *Klassenname*

```
    public String name;
    public int alter;
```
Attribute

```
    public Person() {
        name = "Max Mustermann";
        alter = 38;
    }
```
Konstruktor

```
    public void personenBeschreibungAusgeben(boolean nurNamenNennen) {
        System.out.println("Name: " + name);
        if(!nurNamenNennen) {
            System.out.println("Alter: " + alter);
            System.out.println("Beruf: " + beruf());
        }
    }
```
Methode — *Bezeichner der Attribute*

```
    public abstract String beruf();
}
```
Deklaration einer ***abstrakten Methode***

Student.java

```
public class Student extends Person {
```
Klasse Student als ***Unterklasse*** *der Klasse Person*

```
    public Student(String name, int alter) {
        this.name = name;
        this.alter = alter;
    }
```
Konstruktor *mit Parametern* — *Bezeichner der Attribute*

```
    public String beruf() {
        return "Student";
    }
}
```
Definition der ***abstrakten Methode***

MeineKlasse2.java

```
public class HauptKlasse {
    public static void main(String[] args) {
        Student s = new Student("Peter", 24);

        System.out.println(s.personenBeschreibungAusgeben(true));
        // Ausgabe: Name: Peter
    }
}
```
new + Konstruktor*, um Objekt zu erzeugen* — ***Objekt*** *vom Typ Student* ***erstellen*** — ***Variable deklarieren***

Platz für eigene Notizen

Was will man tun?	Beispielhafte Umsetzung in Java

Bereitgestellte Bibliotheken

Eingabe

`int`	**`Eingabe.intEinlesen`**`()`
	Liest einen **`int`**-Wert von der Konsole ein und gibt diesen zurück. Im Falle einer fehlerhaften Eingabe wird die Zahl **`0`** zurückgeliefert und eine Fehlermeldung in der Konsole ausgegeben.
`double`	**`Eingabe.doubleEinlesen`**`()`
	Liest einen **`double`**-Wert von der Konsole ein und gibt diesen zurück. Im Falle einer fehlerhaften Eingabe wird die Zahl **`0.0`** zurückgeliefert und eine Fehlermeldung in der Konsole ausgegeben.
`String`	**`Eingabe.stringEinlesen`**`()`
	Liest einen **`String`**-Wert von der Konsole ein und gibt diesen zurück. Im Falle einer fehlerhaften Eingabe wird die leere Zeichenkette **`""`** zurückgeliefert und eine Fehlermeldung in der Konsole ausgegeben.
`char`	**`Eingabe.charEinlesen`**`()`
	Liest einen **`char`**-Wert von der Konsole ein und gibt diesen zurück. Wird mehr als ein Zeichen eingegeben, so wird nur das erste Zeichen berücksichtigt. Im Falle einer fehlerhaften Eingabe wird das leere Zeichen **`'\u0000'`** zurückgeliefert und eine Fehlermeldung in der Konsole ausgegeben.
`String[]`	**`Eingabe.dateiInhaltEinlesen`**`()`
	Öffnet den Datei-Öffnen-Dialog, welcher den Benutzer genau eine Datei auswählen lässt. Nach dem Öffnen wird der Inhalt der Datei in ein **`String[]`**-Array gespeichert und zurückgegeben. Ist die Datei leer, nicht lesbar oder erfolgt ein Abbruch wird ein Array der Länge **`0`** zurückgeliefert.
`String[]`	**`Eingabe.dateiInhaltEinlesen`**`(String dateipfad)`
	Wie **`dateiInhaltEinlesen()`**, nur ohne Datei-Öffnen-Dialog. Hier muss mit **`dateipfad`** ein Pfad zur Datei angegeben werden. Windows-Nutzer müssen darauf achten, dass `\` (Backslash) innerhalb eines Strings durch `\\` dargestellt werden muss.

Ausgabe

`void`	**`Ausgabe.arrayAusgeben`**`(int[] array)`
	Gibt das **`int`**-Array **`array`** auf der Konsole aus.
`void`	**`Ausgabe.arrayAusgeben`**`(double[] array)`
	Gibt das **`double`**-Array **`array`** auf der Konsole aus.
`void`	**`Ausgabe.arrayAusgeben`**`(String[] array)`
	Gibt das **`String`** -Array **`array`** auf der Konsole aus.
`void`	**`Ausgabe.arrayAusgeben`**`(char[] array)`
	Gibt das **`char`**-Array **`array`** auf der Konsole aus.
`boolean`	**`Ausgabe.inDateiSchreiben`**`(String[] zeilen)`
	Speichern der Zeilen **`zeilen`** in eine neue oder existierende TXT-Datei. Es wird ein Datei-Speichern-Dialog geöffnet. Gibt **`true`** zurück, wenn das Schreiben erfolgreich war; andernfalls **`false`**.
`boolean`	**`Ausgabe.inDateiSchreiben`**`(String[] zeilen, String dateipfad)`
	Wie **`inDateiSchreiben(String[])`**, nur ohne Datei-Speichern-Dialog. Hier muss mit **`dateipfad`** ein Pfad zur neuen oder existierenden Datei angegeben werden.

Zufall

`int`	**`Zufall.zufaelligenIntWertGenerieren`**`(int min, int max)`
	Liefert einen zufälligen **`int`**-Wert der zwischen **`min`** und **`max`** liegt.
`double`	**`Zufall.zufaelligenDoubleWertGenerieren`**`(double min, double max, int nkstellen)`
	Liefert einen zufälligen **`double`**-Wert der zwischen **`min`** und **`max`** mit Anzahl an Nachkommstellen **`nkstellen`**.
`int[]`	**`Zufall.zufaelligesIntArrayGenerieren`**`(int length)`
	Liefert ein **`int[]`**-Array der Länge **`length`** (**`length`** muss vom Typ **`int`** sein) zurück, welches zufällige **`int`**-Werte von **`0`** bis **`99`** enthält.
`double[]`	**`Zufall.zufaelligesDoubleArrayGenerieren`**`(int length)`
	Liefert ein **`double[]`**-Array der Länge **`length`** (**`length`** muss vom Typ **`int`** sein) zurück, welches zufällige **`double`**-Werte von **`0.0`** bis kleiner **`100`** enthält.