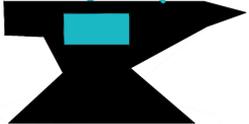


# The GAME Engineers

## #5 – Kommentare, Konstanten, Variablen Scopes und Methoden



The forging engineers



# Inhalt

- Kommentare
- Konstanten
- Gültigkeits- und Sichtbarkeitsbereich von Variablen
- Prozeduren, Funktionen und Methoden



The forging engineers



# Kommentare in Java

#NotMadeByMarvin :(



The forging engineers



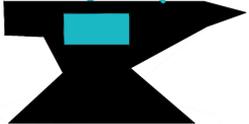
# Kommentare in Java

```
public static void main(String[] args)
{
    // einzeliger Kommentar
    System.out.println("Hallo Welt!");
}
```

```
public static void main(String[] args)
{
    /*
     * mehrzeiliger
     * Kommentar
     */
    System.out.println("Hallo Welt!");
}
```



The forging engineers



# Konstanten in Java

- Spezielle „Variablen“, die nicht verändert werden können
- “Eine Konstante ist ein Name für einen Wert, während eine Variable ein Name für einen Speicherplatz ist.”
- Schlüsselwort *final*

*final* int constant = value;

*final* String constant = value;

...

*final* double PI = 3.1415926;

*final* String ZUTAT = "Mehl";

...



The forging engineers



# Gültigkeits- und Sichtbarkeitsbereich von Variablen / Scope

- Variablen (und Konstanten) sind nur in einem bestimmten Bereich gültig
  - d.h. man kann auf diese nur in diesem Bereich zugreifen und
  - der Name ist nur in diesem Bereich “vergeben”
- Allgemeine Merkregel: “Variablen sind nur gültig und sichtbar innerhalb der geschweiften Klammern ( { } ), in denen diese angelegt wurden.”  
(Hier gibt es Ausnahmen, die allerdings erst später relevant werden.)



The forging engineers



# Gültigkeits- und Sichtbarkeitsbereich von Variablen / Scope

- { } bilden Blöcke. Blöcke werden an verschiedenen Stellen verpflichtend vorausgesetzt (z.B. nach *class* oder bei Schleifen (Mit Ausnahmen!))
- Blöcke dürfen aber auch selbst beliebig gesetzt werden

```
int x = 2;

if (true)
{
    System.out.println(x);
}

System.out.println(x);
```



```
if (true)
{
    int x = 2;

    System.out.println(x);
}

System.out.println(x);
```



The forging engineers



# Gültigkeits- und Sichtbarkeitsbereich von Variablen / Scope

```
public static void main(String[] args)
{
    int x = 1;

    {
        int y = 2;
    }

    System.out.println(x);
    System.out.println(y);
}
```



```
public static void main(String[] args)
{
    for (int i = 0; i < 10; i++)
    {
        System.out.println(i);
    }

    for (int i = 0; i < 10; i++)
    {
        System.out.println(i);
    }
}
```



```
public static void main(String[] args)
{
    int x = 1;

    {
        int x = 2;
    }

    System.out.println(x);
}
```



```
public static void main(String[] args)
{
    {
        int x = 3;

        System.out.println(x);
    }

    int x = 4;
    System.out.println(x);
}
```



The forging engineers



# Unterprogramme

- Unterprogramme sind Programmteile, die eine bestimmte Funktionalität zur Verfügung stellen.
- Das Programm ruft ein Unterprogramm auf. Das Unterprogramm wird abgehandelt und der Fluss kehrt zum Programm zurück

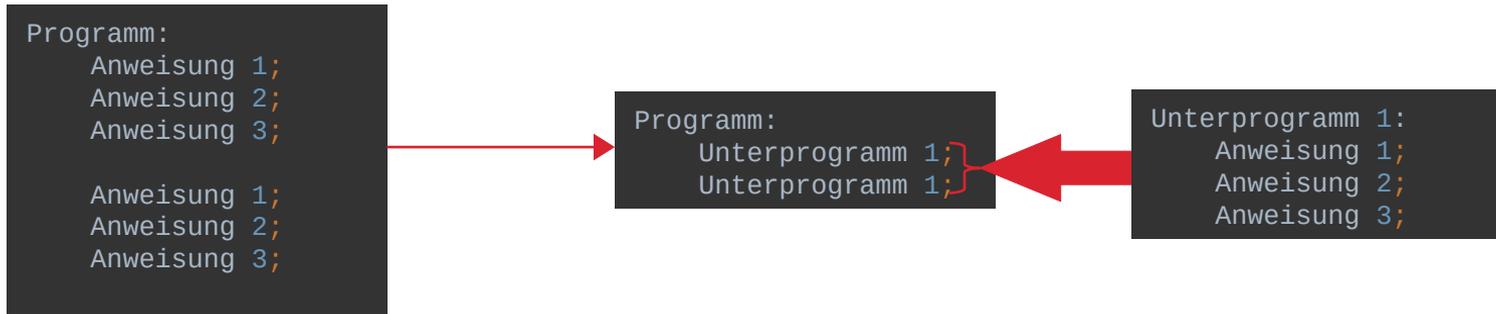


The forging engineers



# Unterprogramme

- Unterprogramme können beliebig oft aufgerufen werden.
- Vorteil: Weniger mehrfacher Code!



The forging engineers



# Unterprogramme

- Es gibt verschiedene Arten von Unterprogrammen, z.B.:
  - Prozeduren
  - Funktionen
  - Methoden



The forging engineers



# Prozedur

- Prozeduren bzw. Unterprogramme, können Werte entgegennehmen und diese verarbeiten

```
Programm:  
Anweisung 1;  
Anweisung 2;  
Anweisung 3;  
  
Unterprogramm 1 ();  
  
Anweisung 7;  
Anweisung 8;
```

```
Unterprogramm 1 ():  
Anweisung 4;  
Anweisung 5;  
Anweisung 6;
```

```
Programm:  
Anweisung 1;  
Anweisung 2;  
Anweisung 3;  
  
Unterprogramm 1 (Wert1, Wert2);  
  
Anweisung 7;  
Anweisung 8;
```

```
Unterprogramm 1 (Wert1, Wert2):  
Wert1 + 1;  
Wert3 = Wert2 + Wert1;  
Ausgabe(Wert3);
```



The forging engineers



# Funktion

- Funktionen sind Prozeduren, die Werte entgegennehmen, diese verarbeiten und auch Werte zurückgeben können:

```
Programm:  
Anweisung 1;  
Anweisung 2;  
Anweisung 3;  
  
Wert1 = Unterprogramm 1 ();  
  
Anweisung 7;  
Anweisung 8;
```

```
Unterprogramm 1 ():  
Anweisung 4;  
Anweisung 5;  
return Wert1;
```



```
Programm:  
Anweisung 1;  
Anweisung 2;  
Anweisung 3;  
  
Wert3 = Unterprogramm 1 (Wert1, Wert2);  
  
Anweisung 7;  
Anweisung 8;
```

```
Unterprogramm 1 (Wert1, Wert2):  
Wert1 + 1;  
Wert3 = Wert2 + Wert1;  
return Wert1;
```



# Methoden

- Methoden sind einfach objektorientierte Funktionen. Was das heißt? Das erörtern wir, wenn wir OOP machen :)
- Aber in Java sprechen wir von Methoden statt Funktionen (ist im Grunde aber erstmal das Gleiche wie Funktionen)



The forging engineers



# Methoden

```
public static void prozedur()  
{  
    System.out.println("Hallo Welt!");  
}
```

```
public static void prozedur(int wert1)  
{  
    System.out.println(wert1);  
}
```

```
public static int funktion()  
{  
    return 0;  
}
```

```
public static double funktion(int wert1, double wert2)  
{  
    return wert1 * wert2;  
}
```



The forging engineers



# Methoden

```
public static void main(String[] args)
{
    double ergebnis = funktion(5, 3.141);
}
```

```
public static double funktion(int wert1, double wert2)
{
    return wert1 * wert2;
}
```



The forging engineers



# Methoden

- Du hast schon Methoden benutzt. Weißt du wo?

```
public static void main(String[] args)
{
    Scanner scanner = new Scanner(System.in);
    int wert1 = scanner.nextInt();

    Random random = new Random();
    int wert2 = random.nextInt(5);

    System.out.println(wert1 + ", " + wert2);
}
```



The forging engineers



**Ende**  
**Fragen?**



The forging engineers

