Einführung in die Rechnerbedienung

Andreas Honecker
Institut für Theoretische Physik, Georg-August-Universität Göttingen

Organisatorisches

- Montag, 5.3.2007 – Donnerstag, 15.3.2007
- Vorlesung: 10:15 – 12:00 in HS 4 (mit Pause)
- Übungen: 13:00 – 16:00 im Multimediaraum (Account beantragen !)
- Materialien / Informationen:

http://www.theorie.physik.uni-goettingen.de/~honecker/rb07

Worum geht es ?

- Programmierkenntnisse für „*Computergestütztes wissenschaftliches Rechnen*“

Programmiersprachen – Typen

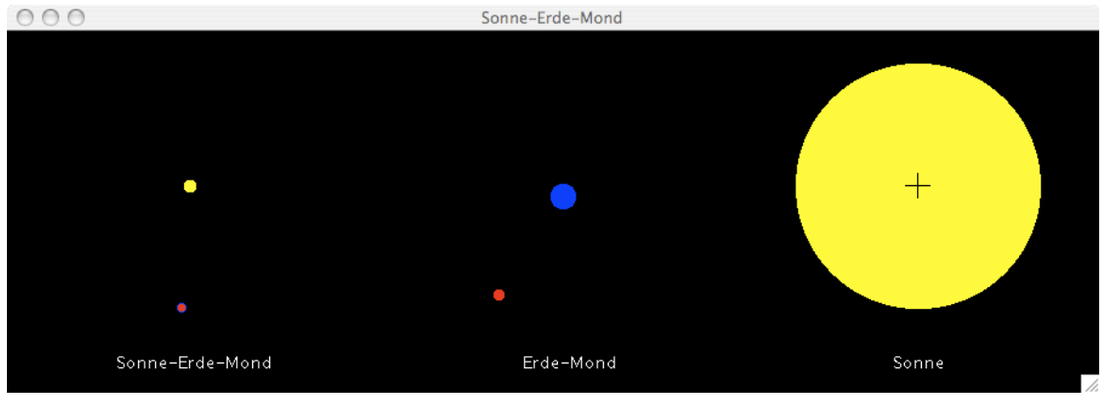
- **Interpreter** (z.B. BASIC):

  Quelltextanweisungen werden **während** Programmausführung „interpretiert“

- **Compiler** (z.B. Fortran, C, C++):

  Aus Quelltext werden **vorab** Maschinen-Instruktionen erzeugt („compiliert“)

Warum Java ?



- Einfacher Zugriff auf Grafik
- Vollwertige Programmiersprache
- „Soft-Skill“: Nachfrage aus Wirtschaft
- Ähnlichkeit zu C bzw. C++

  ⇒ leichte Konvertierung für rechenintensive Anwendungen

Literatur

1. D. Abts, *Grundkurs JAVA*, 4. Auflage, Vieweg-Verlag, Wiesbaden (2004)

   ★ kompaktes & erschwingliches Lehrbuch

2. C. Ullenboom, *Java ist auch eine Insel*, 5. Auflage, Galileo-Computing, Bonn (2006)

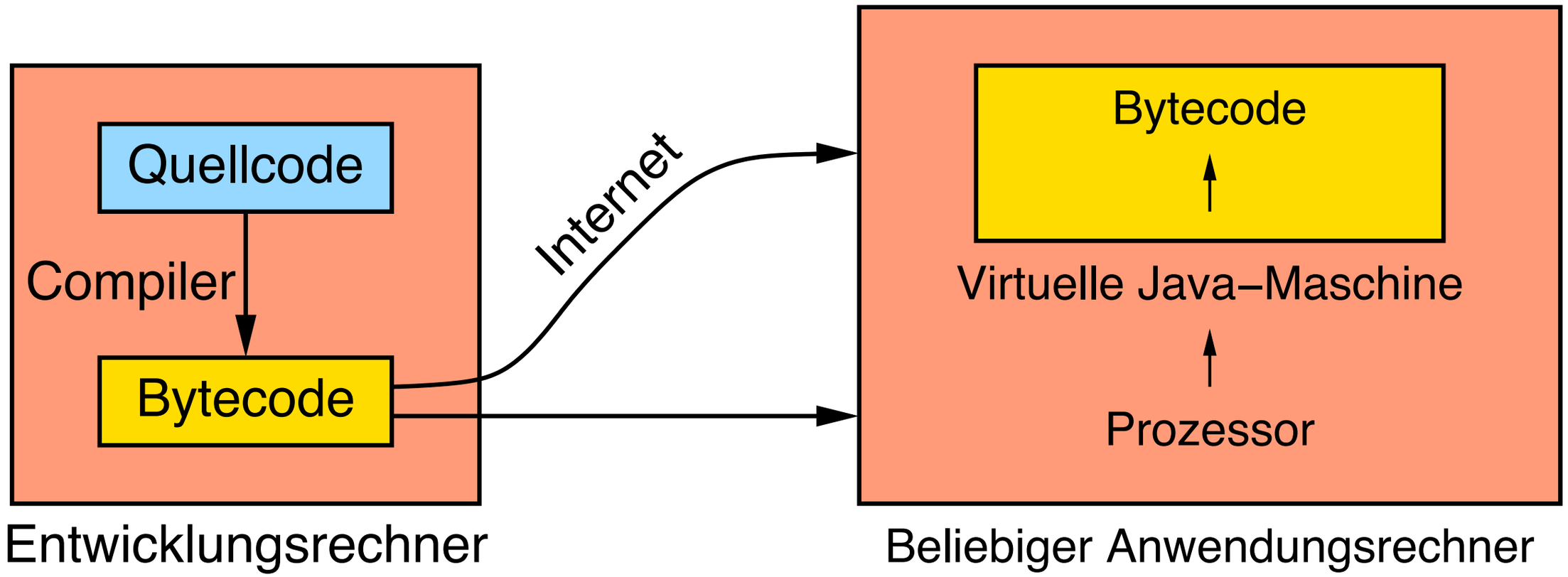
   ★ umfangreiche Referenz

   ★ online zugänglich

Inhalt

- Einführung
- Grundlagen: Quelltext, Compilieren, Datentypen, elementare Operationen
- Ablaufsteuerung (`for`, `if`, ...)
- „Methoden“ (Funktionen)
- Objektorientierte Programmierung: Klassen
- Arrays
- Zeichenketten
- Ein- / Ausgabe
- Grafik & Fenster in Java-Programmen
- Applets

Java: Konzept



⊕ Sicherheit: Eingeschränkter Zugriff ➟ Internet

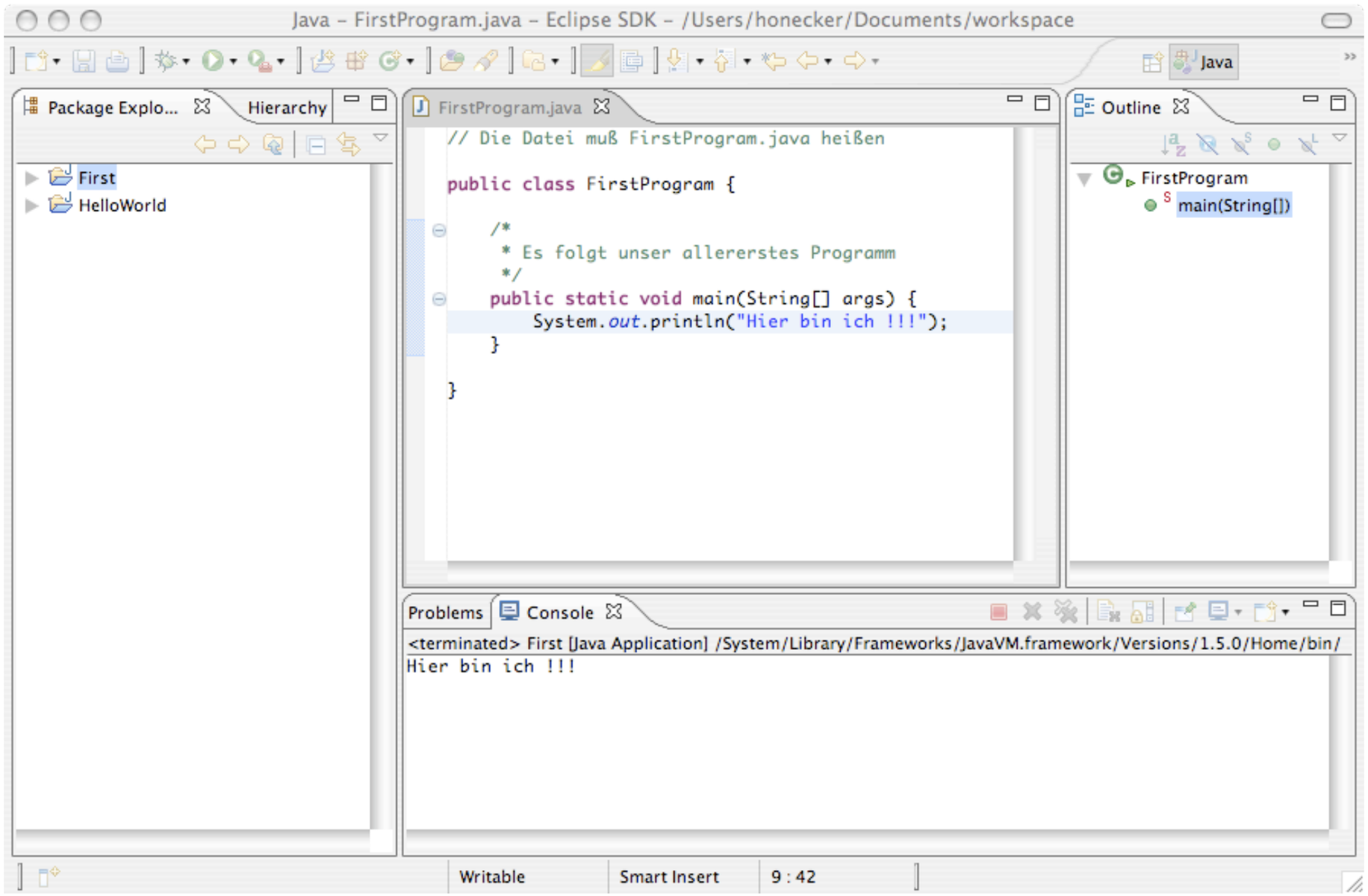
⊕ Portabilität: Plattform-unabhängiges Verhalten

Erstes Beispiel: Empfohlener Weg

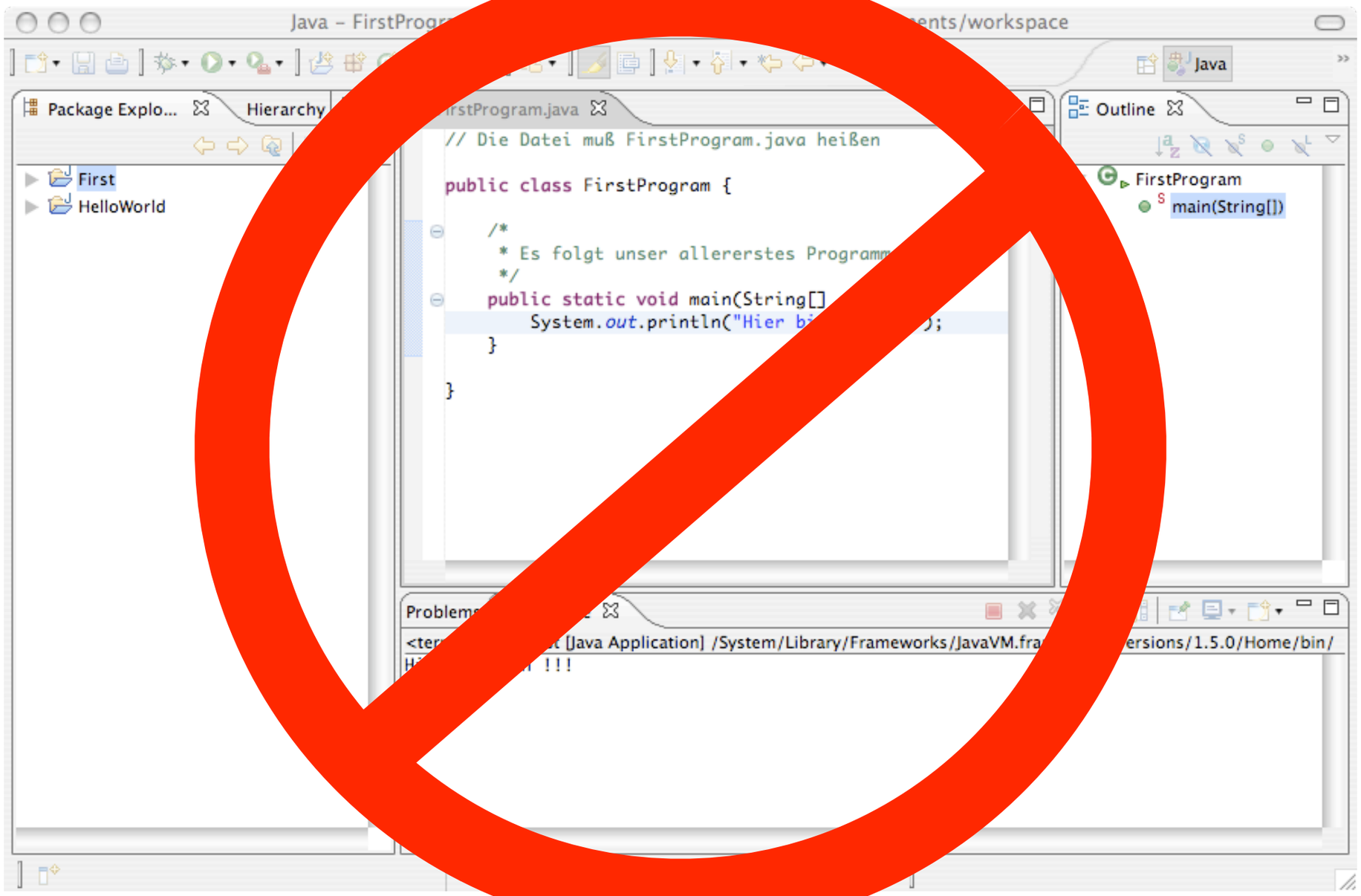
```
// Die Datei muss FirstProgram.java heissen

public class FirstProgram {

    /*
     * Es folgt unser allererstes Programm
     */
    public static void main(String[] args) {
        System.out.println("Hier bin ich !!!");
    }

}
```

- Compilieren mit `javac FirstProgram.java`
- Ausführen mit `java FirstProgram`

Erstes Beispiel: Eclipse



Erstes Beispiel: Eclipse



Wird von Tutoren nicht unterstützt !

Erstes Beispiel: Alternativer Weg

```
// Die Datei muss FirstProgram.java heissen

public class FirstProgram {

    /*
     * Es folgt unser allererstes Programm
     */
    public static void main(String[] args) {
        System.out.println("Hier bin ich !!!");
    }

}
```

- Compilieren mit

  `gcj FirstProgram.java --main=FirstProgram -o FirstProgram`

  ☆ erzeugt Maschinen-Instruktionen

  ☆ unterstützt *nicht* alle Java-Merkmale

- Ausführen mit `./FirstProgram`

Grundlegende Syntax

```
// Die Datei muss FirstProgram.java heissen

public class FirstProgram {

    /*
     * Es folgt unser allererstes Programm
     */
    public static void main(String[] args) {
        System.out.println("Hier bin ich !!!");
    }

}
```

Grundlegende Syntax

```
// Die Datei muss FirstProgram.java heissen

public class FirstProgram {

    /*
     * Es folgt unser allererstes Programm
     */
    public static void main(String[] args) {
        System.out.println("Hier bin ich !!!");
    }

}
```

Grundlegende Syntax

```
// Die Datei muss FirstProgram.java heissen

public class FirstProgram {

    /*
     * Es folgt unser allererstes Programm
     */
    public static void main(String[] args) {
        System.out.println("Hier bin ich !!!");
    }

}
```

Jede Anweisung endet mit einem Semikolon „;“

Grundlegende Syntax

```
// Die Datei muss FirstProgram.java heissen

public class FirstProgram {

    /*
     * Es folgt unser allererstes Programm
     */
    public static void main(String[] args) {
        System.out.println("Hier bin ich !!!");
    }

}
```

Jede Anweisung endet mit einem Semikolon „;“

Grundlegende Syntax

```
// Die Datei muss FirstProgram.java heissen

public class FirstProgram {

    /*
     * Es folgt unser allererstes Programm
     */
    public static void main(String[] args) {
        System.out.println("Hier bin ich !!! ");
    }

}
```

Jede Anweisung endet mit einem Semikolon „;“

Grundlegende Syntax

```
// Die Datei muss FirstProgram.java heissen

public class FirstProgram {

    /*
     * Es folgt unser allererstes Programm
     */
    public static void main(String[] args) {
        System.out.println("Hier bin ich !!! ");
    }

}
```

Jede Anweisung endet mit einem Semikolon „;“

Geschweifte Klammern „{ ... }“ fassen Blöcke zusammen

Grundlegende Syntax

```
// Die Datei muss FirstProgram.java heissen

public class FirstProgram {

    /*
     * Es folgt unser allererstes Programm
     */
    public static void main(String[] args) {
        System.out.println("Hier bin ich !!!");
    }

}
```

Jede Anweisung endet mit einem Semikolon „;“

Geschweifte Klammern „{ ... }“ fassen Blöcke zusammen

Grundlegende Syntax

```
// Die Datei muss FirstProgram.java heissen

public class FirstProgram {

    /*
     * Es folgt unser allererstes Programm
     */
    public static void main(String[] args) {
        System.out.println("Hier bin ich !!!");
    }

}
```

Jede Anweisung endet mit einem Semikolon „;“

Grundlegende Syntax

```
// Die Datei muss FirstProgram.java heissen

public class FirstProgram {

    /*
     * Es folgt unser allererstes Programm
     */
    public static void main(String[] args) {
        System.out.println("Hier bin ich !!!");
    }

}
```

Grundlegende Syntax

```
// Die Datei muss FirstProgram.java heissen

public class FirstProgram {

    /*
     * Es folgt unser allererstes Programm
     */
    public static void main(String[] args) {
        System.out.println("Hier bin ich !!!");
    }

}
```

Grundlegende Syntax

```
// Die Datei muss FirstProgram.java heissen

public class FirstProgram {

    /*
     * Es folgt unser allererstes Programm
     */
    public static void main(String[] args) {
        System.out.println("Hier bin ich !!!");
    }

}
```

Einzeiliger Kommentar: Text nach „//“ wird ignoriert

Grundlegende Syntax

```
// Die Datei muss FirstProgram.java heissen

public class FirstProgram {

    /*
     * Es folgt unser allererstes Programm
     */
    public static void main(String[] args) {
        System.out.println("Hier bin ich !!!");
    }

}
```

Einzeiliger Kommentar: Text nach „//“ wird ignoriert

Grundlegende Syntax

```
// Die Datei muss FirstProgram.java heissen

public class FirstProgram {

   /*
    * Es folgt unser allererstes Programm
    */
   public static void main(String[] args) {
      System.out.println("Hier bin ich !!!");
   }

}
```

Einzeiliger Kommentar: Text nach „//“ wird ignoriert

Grundlegende Syntax

```
// Die Datei muss FirstProgram.java heissen

public class FirstProgram {

   /*
    * Es folgt unser allererstes Programm
    */
   public static void main(String[] args) {
      System.out.println("Hier bin ich !!!");
   }

}
```

Einzeiliger Kommentar: Text nach „//“ wird ignoriert

Mehrzeiliger Kommentar:
Text zwischen „/*... */“ wird ignoriert

Datentypen

Datentyp	Bit	Wertebereich
`boolean`		`true`, `false`
`char`	16	0 ... 2^{16}-1 = 65 535
`byte`	8	-128 ... 127
`short`	16	-32 768 ... 32 767
`int`	32	-2^{31} ... 2^{31}-1
`long`	64	-2^{63} ... 2^{63}-1
`float`	32	ca. 1.4 10^{-45} ... 3.4 10^{38}
`double`	64	ca. 4.9 10^{-324} ... 1.8 10^{308}

Logischer Datentyp

```
boolean a;
a = false;
```

- „`boolean a;`“ deklariert „a“ als Wahrheitswert
- „`a = false;`“ weist „a“ einen von zwei möglichen Werten zu: „`true`“ oder „`false`“
- kompakter:

```
boolean a = false;
```

Zeichen-Datentyp

```
char a;
a = '!';
```

- „`char a;`“ deklariert „a“ als Zeichen
- „`a = '!';`“ weist „a“ den Code des Zeichens „!“ zu
- Wichtige Sonderzeichen:
  Tabulator „`\t`“, Zeilenumbruch „`\n`“
- *`"Zeichenketten"`* stehen in doppelten Anführungszeichen

Ganzzahlige Datentypen

```
byte a = -2;
short b = 32767;
int c = -65536;
long d = 2147483648L;
```

- „a“, „b“, „c“ und „d“ werden als ganze Zahlen mit 8, 16, 32, bzw. 64 Bit deklariert
- „a“, „b“, „c“ und „d“ werden die Werte -2, 32767, -65536 bzw. 2147483648 zugewiesen
- Eine Ziffernfolge (ggfs. mit vorgestelltem „–“) hat den Datentyp `int`; ein nachgestelltes „`L`“ (oder „`l`“) erzwingt den Datentyp `long`

Fließkomma-Datentypen

```
float a = 2.035f;
double b = 1e100;
```

- „a“ wird als einfach, „b“ als doppelt genaue Fließkommazahl deklariert

- Allgemeine Syntax: $A.BeC = A.B \cdot 10^C$
- Fließkommazahlen müssen mindestens einen Punkt „`.`“ oder ein „`e`“ enthalten
  Beispiele: „`1.`“, „`.5`“, „`1e10`“
- Der Datentyp ist `double`; ein nachgestelltes „`f`“ (oder „`F`“) erzwingt den Datentyp `float`

Variablen-Deklarationen

```
Typname Variablenname;
```

- reserviert Speicherplatz für den Datentyp *Typname*
- Zugriff über *Variablenname*
- Gleichzeitige Deklaration und Initialisierung mehrerer Variablen des gleichen Typs
  Beispiel:

```
int v1=123, nummer=-2;
```

Arithmetische Ausdrücke I

bestehen aus Operatoren & numerischen Operanden

Operator	Bezeichnung	Beispiel	Wert
`++`	Inkrementierung	`++a`	`a+1`
`--`	Dekrementierung	`b--`	`b`
`*`	Multiplikation	`2*3`	6
`/`	Division	`6/2`	3
`%`	Rest nach Division	`17%6`	5
`+`	Addition	`3+4`	7
`-`	Subtraktion	`8-5`	3

Arithmetische Ausdrücke 2

- „`++a`“ und „`a++`“ **erhöhen** `a` um `1`, “`--a`“ und „`a--`“ **erniedrigen** `a` um `1`
- „`++a`“ und „`--a`“ verändern `a` **vor**, “`a++`“ und „`a--`“ **nach** dem Auslesen
- Runde Klammern „`(` ... `)`“ erzwingen eine Auswertungs-Reihenfolge
  Beispiel:

```
(a+3.0)/(4-b)
```

Zuweisungen

- einfache Zuweisung:
  Variable = *Ausdruck*;
- Zuweisung mit Operator:
  Variable *op*= *Ausdruck*;
  entspricht
  Variable = *Variable* *op* *Ausdruck*;
- *op* kann einer der Operatoren „*“, „/“, „%“, „+“, „–“ sein

Beispielprogramm

```
// Die Datei muss Beispiel1.java heissen

public class Beispiel1 {
        public static void main(String[] args) {
        int a = 5;
        double b = (a++ - 3.0)/17;
        a /= 2;
        System.out.println(4/3);
        System.out.println("a = " + a);
        System.out.println("b = " + b);
    }

}
```

- „`System.out.println(`*X*`)`“ akzeptiert unterschiedliche Datentypen „*X*“ als Argument
- Ausgaben „*X*“, „*Y*“, ... können mit „+“ aneinander gereiht werden