Java-Technologie

HOCHSCHULE DER MEDIEN

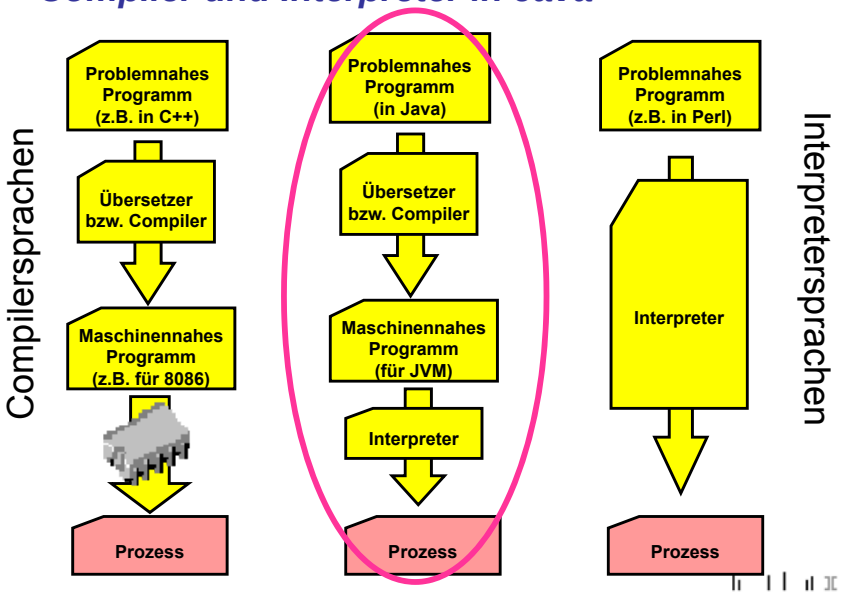
Prof. Dr. Christian Rathke
Prof. Dr. Peter Thies
Hochschule der Medien (HdM)

{thies|rathke}@hdm-stuttgart.de
http://www.hdm-stuttgart.de/~rathke
http://www.prof-thies.de/

Die Java-Technologie

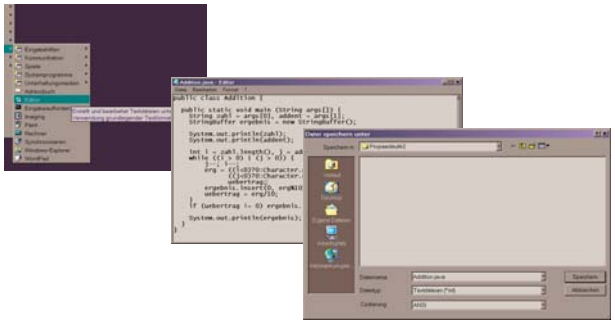
- Die Java-Technologie besteht aus der Java-*Programmiersprache* und der Java-*Plattform*
- Die Java-Programmiersprache ist eine *problemorientierte* Programmiersprache. Sie ist
  - (relativ) einfach
  - objektorientiert
  - verteilt
  - robust
  - sicher
  - architektur-neutral
  - portabel
  - performant
  - nebenläufig
  - dynamisch

Compiler und Interpreter in Java



Erstellen eines Java-Programms

- Java-Programme sind normale Texte und können mit einfachen Texteditoren erstellt werden.
- Es ist darauf zu achten, dass reine Texte ohne Formatierungen abgespeichert werden (Vorsicht bei MSWord!).



Exemplarische Sprachelemente – Das Additionsprogramm in Java

```
public class Addition {

  public static void main (String args[]) {
    String zahl1 = args[0], zahl2 = args[1];

    int[][] werteTabelle = {{0,1,2,3,4,5,6,7,8,9},
                            {1,2,3,4,5,6,7,8,9,0},
                            {2,3,4,5,6,7,8,9,0,1},
                            {3,4,5,6,7,8,9,0,1,2},
                            {4,5,6,7,8,9,0,1,2,3},
                            {5,6,7,8,9,0,1,2,3,4},
                            {6,7,8,9,0,1,2,3,4,5},
                            {7,8,9,0,1,2,3,4,5,6},
                            {8,9,0,1,2,3,4,5,6,7},
                            {9,0,1,2,3,4,5,6,7,8}};
    int[][] uebertragTabelle = {{0,0,0,0,0,0,0,0,0,0},
                                {0,0,0,0,0,0,0,0,0,1},
                                {0,0,0,0,0,0,0,0,1,1},
                                {0,0,0,0,0,0,0,1,1,1},
                                {0,0,0,0,0,0,1,1,1,1},
                                {0,0,0,0,0,1,1,1,1,1},
                                {0,0,0,0,1,1,1,1,1,1},
                                {0,0,0,1,1,1,1,1,1,1},
                                {0,0,1,1,1,1,1,1,1,1},
                                {0,1,1,1,1,1,1,1,1,1}};
    int[] nachfolgerTabelle = {1,2,3,4,5,6,7,8,9,0};

    StringBuffer ergebnis = new StringBuffer();
    System.out.println(zahl1);
    System.out.println(zahl2);

    int i = zahl1.length(), j = zahl2.length();
    int letzterUebertrag = 0;
    …
```

Schlüsselwörter

Symbole

Exemplarische Sprachelemente – Das Additionsprogramm in Java (fortg.)

```
…
    while ((i > 0) | (j > 0)) {
      int ziffer1 = ((i<0)?0:Character.getNumericValue(zahl1.charAt(i-1)));
      int ziffer2 = ((j<0)?0:Character.getNumericValue(zahl2.charAt(j-1)));

      int neuerWert = werteTabelle[ziffer1][ziffer2];

      int neuerUebertrag;
      if (letzterUebertrag == 1) {
        neuerWert = nachfolgerTabelle[neuerWert];
        if (neuerWert == 0) neuerUebertrag = 1;
        else neuerUebertrag = uebertragTabelle[ziffer1][ziffer2];
      }
      else neuerUebertrag = uebertragTabelle[ziffer1][ziffer2];

      letzterUebertrag = neuerUebertrag;
      ergebnis.insert(0, neuerWert);
      j--; i--;

    };
    if (letzterUebertrag == 1) ergebnis.insert(0, letzterUebertrag);

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

Vordefinierte Programmteile

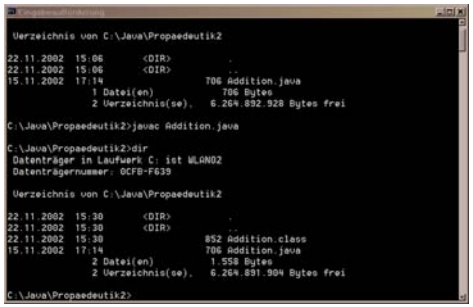
Vordefinierte Elemente zur Steuerung der Abfolge und Wiederholung von Programmteilen

Sprachelemente

- Vordefinierte *Schlüsselwörter* mit festgelegter Bedeutung wie z.B. `class`, `int`, `public`,
- Benutzerdefinierte Bezeichner und Symbole,
- Vordefinierte Programmteile, die über ihre Namen aktiviert werden wie z.B. die Textausgabe auf den Bildschirm,
- Vordefinierte Elemente zur Steuerung der Abfolge und Wiederholung von Programmteilen,
- Genau festgelegte Syntax und Semantik für die Verwendung aller Sprachelemente.

Die Übersetzung des Java-Programms

- Das Java-Programm wird durch Aufruf des Java-Compilers für die Java-Datei übersetzt.
- Es entsteht dabei eine neue Datei gleichen Namens mit der Erweiterung `.class`, die eine maschinennahe Form des Programms enthält.
- Die maschinennahe Form von Java nennt man „Bytecode“.



Exemplarische Sprachelemente – Das Additionsprogramm in Bytecode

```
Compiled from Addition.java
public class Addition extends java.lang.Object {
   public Addition();
   public static void main(java.lang.String[]);
}

Method Addition()
   0 aload_0
   1 invokespecial #1 <Method java.lang.Object()>
   4 return

Method void main(java.lang.String[])
   0 aload_0
   1 iconst_0
   2 aaload
   3 astore_1
   4 aload_0
   5 iconst_1
   6 aaload
   7 astore_2
   8 new #2 <Class java.lang.StringBuffer>
  11 dup
  12 invokespecial #3 <Method java.lang.StringBuffer()>
  15 astore_3
  16 getstatic #4 <Field java.io.PrintStream out>
  19 aload_1
  20 invokevirtual #5 <Method void println(java.lang.String)>
  23 getstatic #4 <Field java.io.PrintStream out>
  26 aload_2
  27 invokevirtual #5 <Method void println(java.lang.String)>
  30 aload_1
  31 invokevirtual #6 <Method int length()>
  34 istore 4
  36 aload_2
  37 invokevirtual #6 <Method int length()>
  40 istore 5
  42 iconst_0
  43 istore 6
  45 goto 114
  48 iinc 5 -1
  51 iinc 4 -1
  54 iload 4
  56 ifge 63
  59 iconst_0
  60 goto 72
  63 aload_1
  64 iload 4
  66 invokevirtual #7 <Method char charAt(int)>
  69 invokestatic #8 <Method int getNumericValue(char)>
  72 iload 5
  74 ifge 81
  77 iconst_0
  78 goto 90
  81 aload_2
  82 iload 5
  84 invokevirtual #7 <Method char charAt(int)>
  87 invokestatic #8 <Method int getNumericValue(char)>
  90 iadd
  91 iload 6
  93 iadd
  94 istore 7
  96 aload_3
  97 iconst_0
  98 iload 7
 100 bipush 10
 102 irem
 103 invokevirtual #9 <Method java.lang.StringBuffer
insert(int, int)>
 106 pop
 107 iload 7
 109 bipush 10
 111 idiv
 112 istore 6
 114 iload 4
 116 ifle 123
 119 iconst_1
 120 goto 124
 123 iconst_0
 124 iload 5
 126 ifle 133
 129 iconst_1
 130 goto 134
 133 iconst_0
 134 ior
 135 ifne 48
 138 iload 6
 140 ifeq 151
 143 aload_3
 144 iconst_0
 145 iload 6
 147 invokevirtual #9 <Method java.lang.StringBuffer
insert(int, int)>
 150 pop
 151 getstatic #4 <Field java.io.PrintStream out>
 154 aload_3
 155 invokevirtual #10 <Method void
println(java.lang.Object)>
 158 return
```

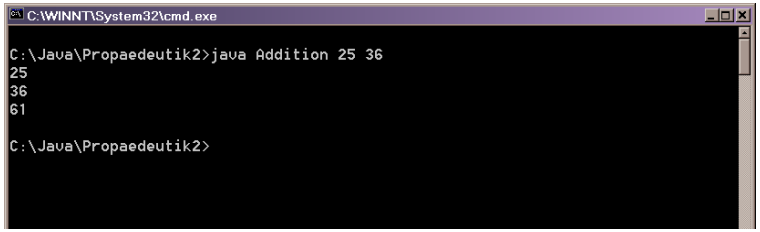
Symbolische Speicheradressen

Sprachelemente

- Vordefinierte Maschinenbefehle mit festgelegter Bedeutung, wie z.B. *aload, return, astore, goto, pop.*
- Symbolische Speicheradressen, wie z.B. *_0, _1, _2*
- Vordefinierte Programmteile, die über spezielle Maschinenbefehle und ihre Namen aufgerufen werden, wie z.B. *invokevirtual #9 <Method java.lang.StringBuffer insert(int, int)>*
- Vordefinierte Elemente zur Steuerung der Abfolge und Wiederholung von Programmteilen, wie z.B. *goto, ifle*
- Genau festgelegte Syntax und Semantik für die Verwendung aller Sprachelemente.

Die Ausführung des Java-Programms

- Die Ausführung erfolgt in der maschinennahen Form, d.h. im Falle von Java, als Bytecode.
- Benutzten wir anstelle eines Java-Prozessors z.B. einen Intel-Prozessor, könnten wir das Programm durch Doppelklick starten.
- Stattdessen starten wir den Java-Interpreter und lassen das Programm durch ihn ausführen.



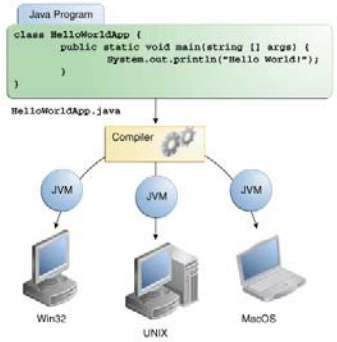
Die Java-Programmiersprache

- sowohl compiliert als auch interpretiert;
  - compiliert in eine maschinennahe Zwischensprache, genannt *Bytecode;*
  - interpretiert durch den Interpreter, genannt die *Java Virtual Machine (JVM);*

- compiliert wird das Programm nur einmal, um den Bytecode zu erzeugen;
- interpretiert wird das Programm jedes Mal, wenn es ausgeführt werden soll.

Der Java-Bytecode

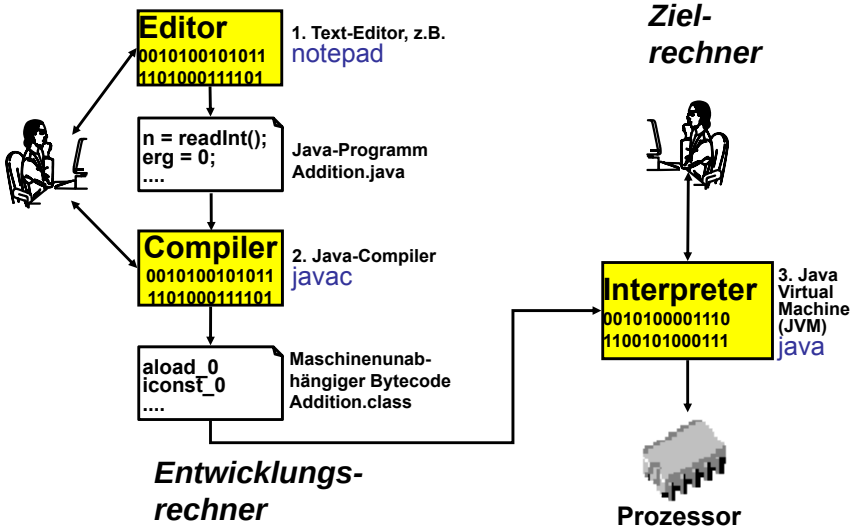
- Bytecodes ermöglichen die Erfüllung des Anspruchs "Einmal geschrieben, überall ablauffähig".
- Das Compilieren des Programms in Bytecode ist auf jeder beliebigen Plattform mit Hilfe des Java-Compilers möglich.
- Die Interpretation des Bytecodes kann durch jede Realisierung der JVM geleistet werden.



Java-Programmierung

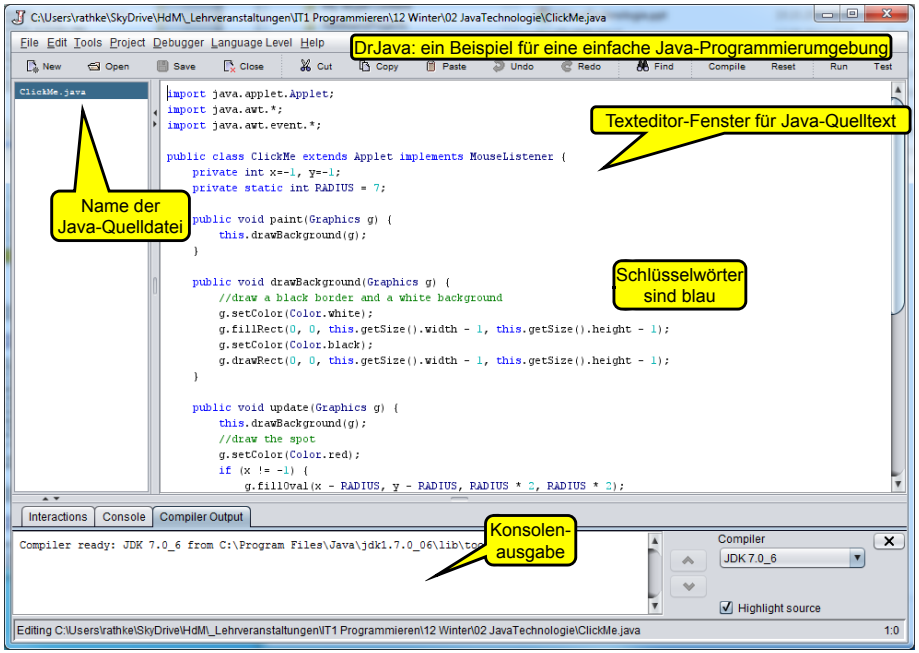
1. **Erzeugen einer Java-Quelldatei.** Eine Java-Quelldatei enthält das Java-Programm als reinen Text (ohne Formatierungsinformationen), geschrieben in der Programmiersprache Java. Diesen Text kann man mit Hilfe einer beliebigen Textanwendung erstellen.
2. **Aus der Java-Quelldatei wird durch einen Übersetzungsvorgang („Compilieren“) eine sog. Bytecode-Datei erzeugt.** Der *Java-Compiler* – ein Programm mit Namen "javac" – erzeugt aus der Java-Quelldatei die Bytecode-Datei. Diese enthält für uns unleserlichen Text, der aber durch den virtuellen Java-Prozessor (*Java Virtual Machine*), ausgeführt werden kann.
3. **Das Programm in der Bytecode-Datei ist ablauffähig**. Die Java Virtual Machine wird durch ein weiteres Programm – den Java *Interpreter* (Name: "java") – realisiert. Das Programm führt den Inhalt der Bytecode-Datei auf unserem Rechner aus.

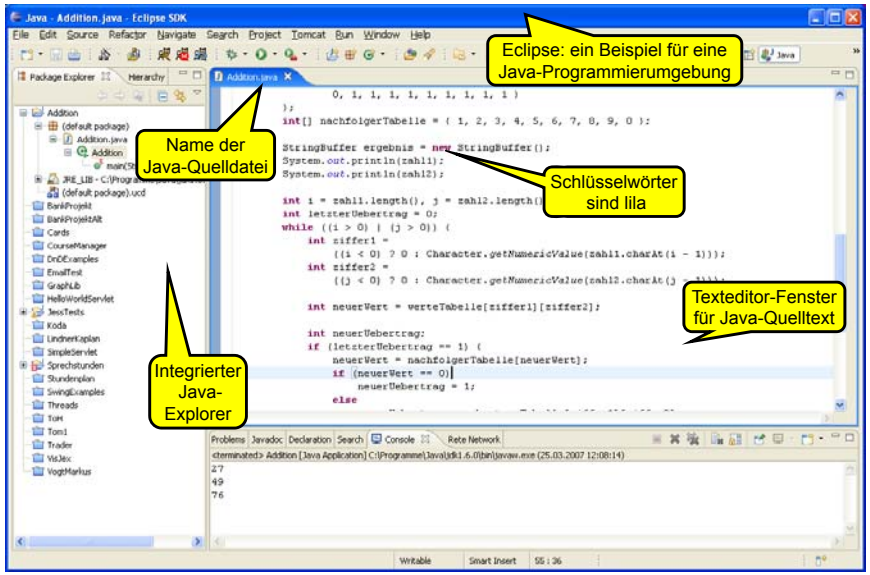
Java-Programmierung (II)



Programmierumgebungen

- Programmierumgebungen oder Entwicklungsumgebungen erleichtern das Programmieren.
- Sie stellen eine Integration aller zur Entwicklung und Ausführung von Programmen notwendigen Werkzeuge dar.
- Engl.: Integrated Development Environments (IDEs).
- Alle Aktivitäten der Programmerstellung werden über fenster- und menübasierte Schnittstellen ermöglicht.
- Häufig vorkommende Aktionen haben eigene Menüeinträge oder sind eigenen Tasten hinterlegt.
- Spezielle Texteditoren nutzen die syntaktische Merkmale der Programmiersprache aus, um
  - Teile mit Farbe oder Stil zu kennzeichnen,
  - Strukturen durch Einrückung oder Markierung hervorzuheben.





Programmierumgebungen erleichtern den Java-Entwicklungs-Zyklus

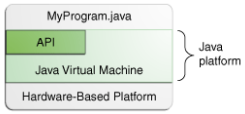
1. Das Java-Programm in der Programmierumgebung laden/eingeben/modifizieren.
   - Syntaktische Hilfen durch Farben und Markierungen
   - Vordefinierte Code-Segmente
   - Direkter Zugang zur Dokumentation
2. Das Programm in eine Datei abspeichern!
   - Vorgabe des Dateinamens
   - Kontrolle syntaktischer Eigenschaften
3. Das Programm übersetzen.
   - Taste bzw. Menüauswahl.
   - Automatisches Abspeichern bei Veränderungen.
4. Beim Auftreten von Fehlern zum Überarbeiten des Programms zurück zum Editor.
   - Separater Anzeigebereich,
   - Implizite Fehlerlokalisation

Programmierumgebungen erleichtern den Java-Entwicklungs-Zyklus

5. Das Programm ausführen.
   - Taste bzw. Menüauswahl
   - Fehlersuchmodus (Debugging Mode) mit schrittweiser Ausführung.
6. Beim Auftreten von Fehlern unmittelbares Überarbeiten des Programms im Editor.
7. Das Programm macht seine Ausgaben
   - Ausgaben in eigenem Fenster oder separatem Bereich

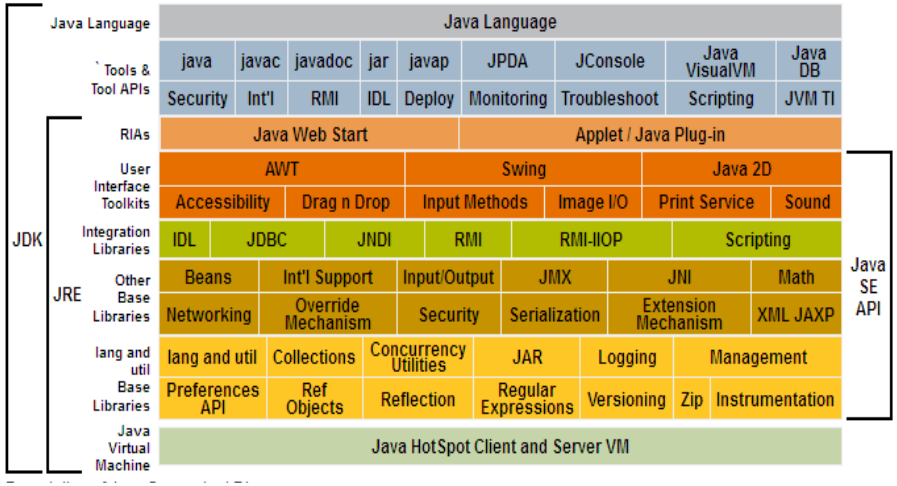
Die Java-Plattform

- Eine *Plattform* ist die Hard- und Software-Umgebung, in der ein Programm abläuft.
- Beispiele: Windows 7, Linux, Solaris, MacOS zusammen mit dem jeweiligen Rechnertyp
- Die Java-Plattform basiert ausschließlich auf Software und läuft auf anderen, Hardware-basierten Plattformen.
- Sie besteht aus 2 Komponenten:
  - Java Virtual Machine (Java VM)
  - Java Application Programming Interface (Java API)



- Java API
  - große Ansammlung direkt verwendbarer Software-Komponenten z.B. für GUIs oder Datenbankzugriffe,
  - aufgeteilt in Bibliotheken von zusammengehörenden Klassen und Schnittstellen,
  - diese Java-Software-Bibliotheken heißen "Pakete" (packages)
- Das Java-API und die Java-VM machen Programme unabhängig von der Hardware.

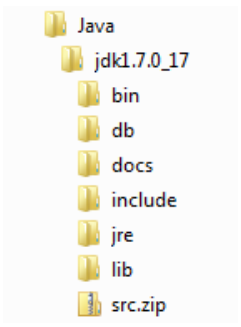
Java Standard Edition 7 Plattform

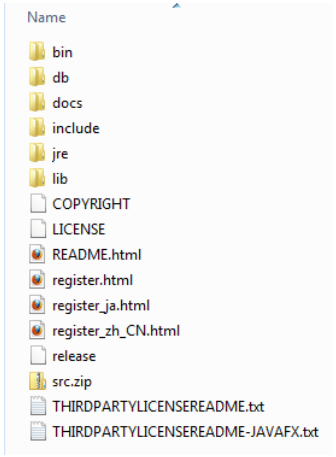


Bezug und Installation der Java-Plattform

- Die Firma Oracle stellt eine kostenfrei nutzbare Java-Entwicklungsumgebung bereit, das Java® Development Kit (JDK™).
- Ein Installationsprogramm (ca. 77MB) und Dokumentation (ca. 56MB) kann aus dem Web heruntergeladen werden von http://www.oracle.com/technetwork/java/javase/downloads/index.html
- Beim Start des Installationsprogramms (Doppelklick im Explorer) wird die Java-Software in ein Zielverzeichnis (z.B. C:\Program Files\Java) entpackt.
- Die Java-Dienstprogramme (Compiler, Interpreter usw.) liegen auf dem Unterverzeichnis \bin des Zielverzeichnisses.

Installation der Java-Plattform (fortg.)





Bezug und Installation von Java-Entwicklungsumgebungen

- DrJava:
  http://drjava.sourceforge.net
- Eclipse:
  http://www.eclipse.org

Die Java Technologie – Programmtypen

- **Client basiert**
  - **Anwendungen**: alleinstehende (stand alone), ablauffähige Programme. Diese sind auf der Java-Plattform ablauffähig.
  - **Applets**: im Browser ablauffähige Programme
- **Server basiert**
  - **Dienste** (Servers): Anwendungen, die von mehreren anderen Programmen (Clients) über eine Netzverbindung genutzt werden (z.B. Web-Servers, Mail-Servers, Print-Servers)
  - **Servlets**: Java Programme auf einem Web-Server, die es erlauben, HTML-Seiten dynamisch zu erzeugen (Alternative zu CGI-Scripts oder PHP).

Die erste Java Anwendung

```
/**
 * The HelloWorldApp class implements an application that
 * simply displays "Hello World!" to the standard output.
 */
class HelloWorldApp {
   public static void main(String[] args) {
        System.out.println("Hello World!"); //Display the string.
   }
}
```

- Kommentare werden in /** und */ eingeschlossen bzw. mit // eingeleitet
- Jede Operation (in Java: Methode) und jedes Datum existiert in einer Klasse oder einem Objekt – der Instanz einer Klasse
- Die Methode `main` ist der Startpunkt einer Java-Anwendung
- alles andere besteht aus Objekten, Klassen, Methoden und Java-Sprachelementen

Kommentare

```
/**
 * The HelloWorldApp class implements an application that
 * simply displays "Hello World!" to the standard output.
 */
class HelloWorldApp {
   public static void main(String[] args) {
      System.out.println("Hello World!"); //Display the string.
   }
}
```

Java unterstützt drei Arten von Kommentaren:

1. */* *Text* */* — wird vom Compiler ignoriert
2. */** *Dokumentation* */* — wird von javadoc verwendet, um Dokumentation zu generieren
3. *// Text* — wird bis Zeilenende vom Compiler ingnoriert

Klassendefinitionen

```
/**
 * The HelloWorldApp class implements an application that
 * simply displays "Hello World!" to the standard output.
 */
class HelloWorldApp {
  public static void main(String[] args) {
    System.out.println("Hello World!"); //Display the string.
  }
}
```

- Jedes Java-Programm besteht aus mindestens einer oder mehreren so genannten Klassen.
- Hinter dem Schlüsselwort `class` folgt der Klassenname.
- Was zur Klasse dazugehört, wird in geschweiften Klammern eingeschlossen.

Methodendefinitionen

```
/**
 * The HelloWorldApp class implements an application that
 * simply displays "Hello World!" to the standard output.
 */
class HelloWorldApp {
  public static void main(String[] args) {
    System.out.println("Hello World!"); //Display the string.
  }
}
```

- Jede Klasse kann mehrere Operationen, in Java heißen sie *Methoden*, enthalten.
- Was zur Methode dazugehört, steht in den geschweiften Klammern.
- Jede allein stehende Java-Anwendung muss die Methode **`main`** enthalten, deren „Signatur“ genau so aussieht:
  `public static void main(String[] args)`

Methodendefinitionen (fortg.)

```
/**
 * The HelloWorldApp class implements an application that
 * simply displays "Hello World!" to the standard output.
 */
class HelloWorldApp {
  public static void main(String[] args) {
    System.out.println("Hello World!"); //Display the string.
  }
}
```

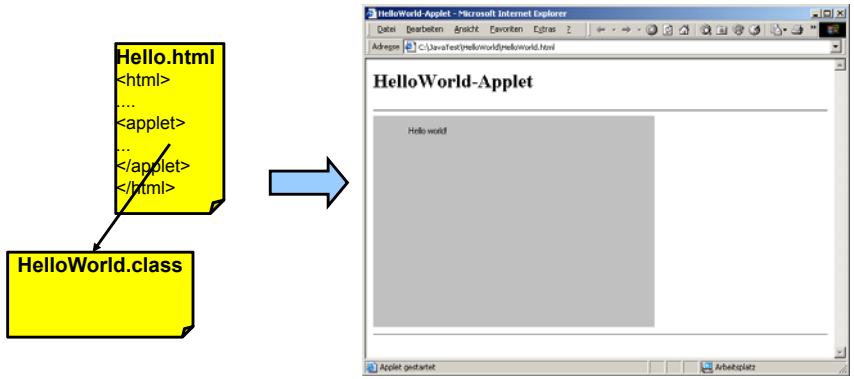
- Dem Methodennamen sind drei „Modifikatoren“ vorangestellt:
  - **`public`** bedeutet, dass die Methode öffentlich zugänglich ist und von jedem „fremden“ Programmteil aufgerufen werden kann.
  - **`static`** bedeutet, dass die Methode eine sog. Klassenmethode ist.
  - **`void`** bedeutet, dass die Methode kein Ergebnis berechnet.
- Die Methode hat einen sog. Parameter, d.h. bei Aufruf können zusätzliche Angaben in Form sog. Argumente an die Methode übertragen werden.

Die Definition der Main-Methode

```
/**
 * The HelloWorldApp class implements an application that
 * simply displays "Hello World!" to the standard output.
 */
class HelloWorldApp {
  public static void main(String[] args) {
    System.out.println("Hello World!"); //Display the string.
  }
}
```

- Die "Hello World"-Anwendung ist eine der einfachsten Anwendungen, die tatsächlich etwas Sichtbares bewirken.
- Sie enthält nur eine einzige Klasse: `HelloWorldApp`.
- Sie enthält nur die (erforderliche) Methode `main`.
- Diese gibt mit Hilfe der Anweisung
  `System.out.println("Hello World!");`
  eine Textzeile auf dem Bildschirm aus.

Java-Programme im Browser: Java-Applets



Applets = Java-Programme in Bytcode,
auf die innerhalb einer Web-Seite verwiesen wird.

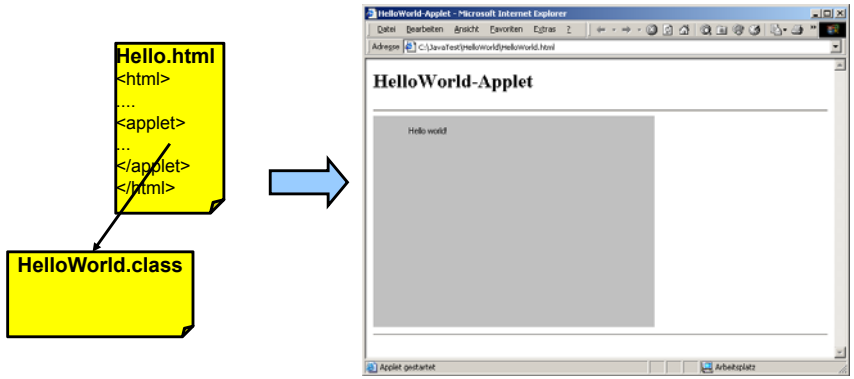
Das erste Java-Applet

```
import java.applet.Applet;
import java.awt.Graphics;

/**
 * The HelloWorld class implements an applet that
 * simply displays "Hello World!".
 */
public class HelloWorld extends Applet {
    public void paint(Graphics g) {
        // Display "Hello World!"
        g.drawString("Hello World!", 50, 25);
    }
}
```

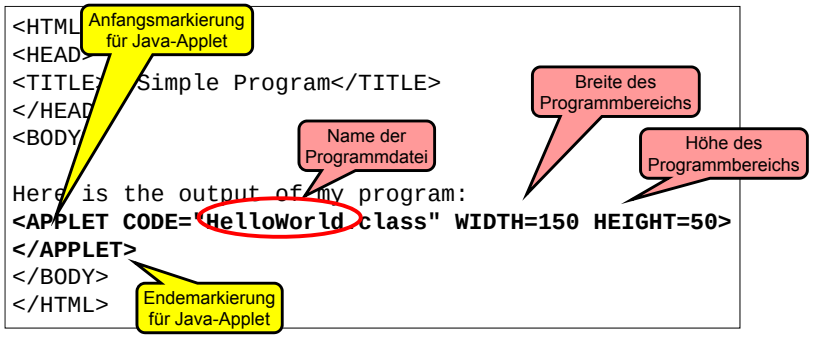
- Java-Applets verwenden die Klasse `Applet` aus dem Applet-API.
- Java-Applets können mit graphischen Operation auf der Applet-Fläche einer Web-Seite ausgeben.

Java-Programme im Browser: Java-Applets



Applets: Java-Programme in Bytcode,
auf die innerhalb einer Web-Seite verwiesen wird.

Das Applet ablaufen lassen



- Die Markierung <APPLET> legt fest, dass der Browser die Klasse laden soll, deren compilierter Code in der Datei namens HelloWorld.class ist.
- Der Browser sucht nach dieser Datei im selben Verzeichnis wie das der HTML-Datei.

Das Applet ablaufen lassen (II)

- Wenn der Browser die Klassendatei gefunden hat, lädt er sie über das Netz (falls erforderlich).
- Dann erzeugt der Browser eine Instanz dieser Klasse. Falls das Applet zweimal auf der Seite auftaucht, wird die Datei nur einmal geladen, aber es werden zwei Instanzen der Klasse erzeugt.
- Die Attribute `WIDTH` und `HEIGHT` legen die Größe des Ausgabebereichs des Applets in Pixel fest.
- Diese Dimensionen können vom Applet nicht überschritten werden.

Java-Anwendungen und Java-Applets

	Anwendungen (Applications)	**Applets**
Entwicklungs-Umgebung	ein (Text-) Editor	ein (Text-) Editor
Programm-Code	Java	Java
Übersetzung	mit dem Java-Compiler "javac"	mit dem Java-Compiler "javac"
Ablaufumgebung	**ein Rechner mit der Java-Laufzeitumgebung**	**ein Java-fähiger Browser**
Aktivierung	**Aufruf des Java-Interpreters**	**Anzeigen der Web-Seite**