Inhalt

2 Einstieg in das Programmieren mit Java

2.1 Geschichte

1990 Oak Object Application Kernel
Sun ↷ **Steuerung von Haushaltsgeräten**

Java wurde ab 1990 von einer Forschungsgruppe „Green Project“ unter Leitung von Billy Joy und James Gosling der Firma *Sun Microsystems* entwickelt, ursprünglich als einheitliche Hochsprache zur Steuerung von Haushaltsgeräten. Zunächst trug sie den Namen **Oak** (***O**bject **A**pplication **K**ernel*).

1993 Oak *Sun* ↷ Internetfähigkeit

Um 1993 erkannte *Sun*, dass sich die Sprache **Oak** für Animationen des sich damals stürmisch entwickelnden **Internets** eignet.

1995 Java *Netscape* ↷ Java-Anwendungen auf Web-Seiten

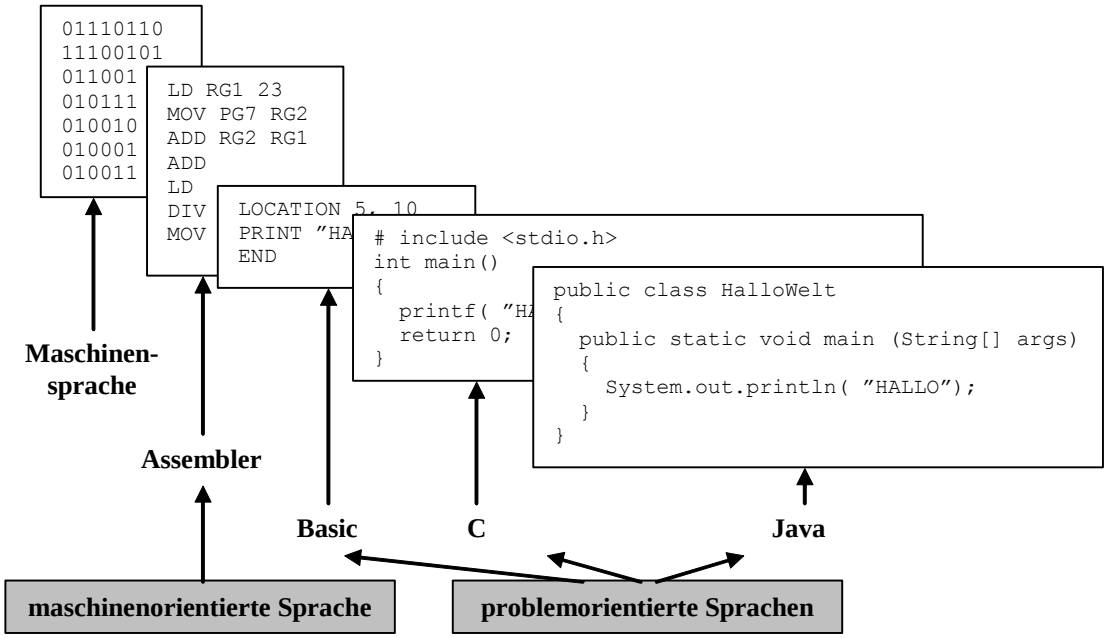
Den Durchbruch erlebte **Java**, als die Firma *Netscape* Anfang 1995 bekannt gab, dass ihr *Navigator 2.0* Java-Programme in Web-Seiten ausführen kann.

Eigenschaften der Sprache

- Java hat eine an **C / C++** angelehnte Syntax, *ohne* deren systemnahen Konstrukte.
- Sie ist *keine* reine OOP, denn sie hat sowohl *imperative* als auch *objektorientierte* Bestandteile. Java ist somit eine **Hybridsprache**.
- Mit Java entstand eine Sprache zur Entwicklung von umfangreichen Anwendungen, die über das globale Computernetzwerk transportiert werden können (**Applet**), aber auch solche, die nichts mit Netzwerken zu tun haben (**Application**).

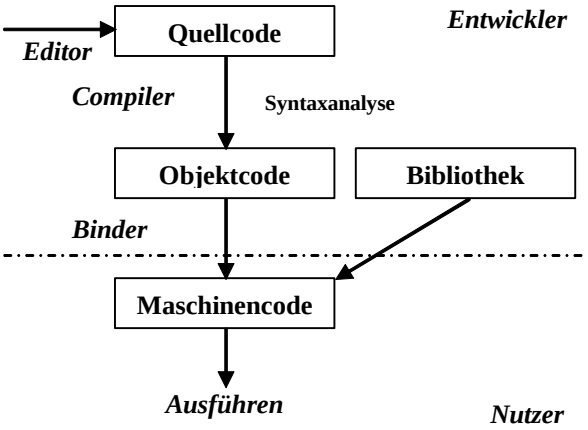
2.2 Konzept

2.2.1 Compiler und Interpreter



Jedes problemorientierte Programm muss vor der Ausführung in die Maschinensprache des Computers übersetzt werden. Das geschieht durch spezielle Programme. Dazu gibt es zwei Ansätze:

Compiler
Die Textdatei des Programms, der **Quellcode**, wird durch einen **Compiler** auf korrekte Syntax überprüft. Werden keine Fehler gefunden, so erzeugt der Compiler einen neuen Code, den **Objektcode**. Dieser Code wird anschließend vom **Binder** mit benötigten Bibliotheksprogrammen zum ausführbaren **Maschinencode** zusammengefügt. *Die Übersetzung liegt beim Entwickler.* Der Anwender braucht nur das Programm im Maschinencode für seinen Computer. In welcher Sprache das Programm ursprünglich entwickelt wurde, interessiert ihn nicht.

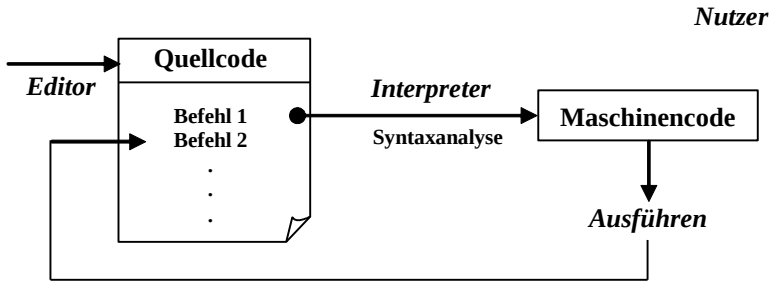


Übersetzte Programme sind im Allgemeinen sehr *schnell*, da die Übersetzung bereits vorliegt. Dafür sind sie aber nur auf der Systemplattform (Hardware und Betriebssystem) lauffähig, für

die sie übersetzt wurden. Deshalb muss man beim Installieren von Software stets das vorhandene Betriebssystem angeben.
Für den Gebrauch im Rahmen des Internets ist das ein Nachteil, da hier eine Vielzahl von verschiedenen Systemplattformen untereinander vernetzt sind.

Beispiele: Algol, Pascal, C

Interpreter
Der Quellcode des Programms bleibt bis zur Ausführung unbearbeitet. *Die Übersetzung liegt beim Nutzer*. Zum Starten braucht man außer diesem einen ***Interpreter***. Der Interpreter liest zur Laufzeit einen Befehl des Quellcodes ein, wandelt ihn in **Maschinencode** um und führt ihn aus. Erst dann wird der nächste Befehl des Quellcodes vom Interpreter behandelt.



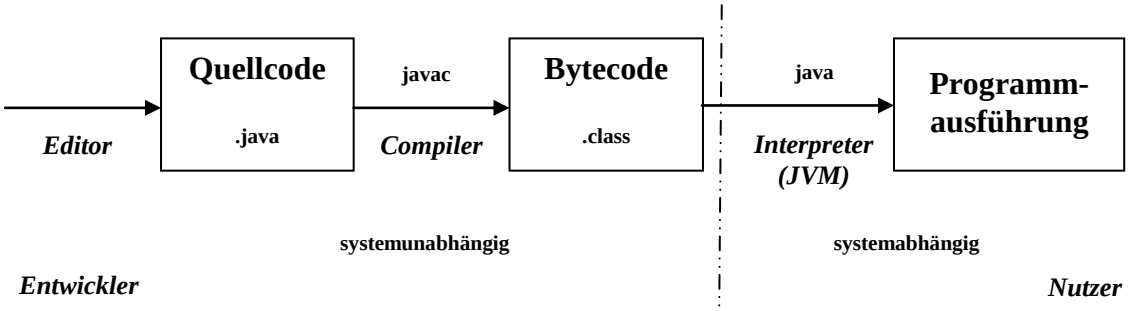
Da der Quellcode bei jedem Programmaufruf immer wieder neu übersetzt werden muss, sind zu interpretierende Programme *langsamer*. Syntaktische Fehler machen sich erst zur Laufzeit bemerkbar und führen zu Programmabbrüchen. Die Entwicklung selbst geht etwas schneller, da die Fehleranalyse solcher Programme leichter ist.

Beispiele: Basic, Kommandosprachen

2.2.2 Bytecode und virtuelle Maschinen

Das Prinzip von **Java** basiert auf eine *Kombination beider Übersetzungstechniken*. Ein Java-**Quellcode** wird vom *Entwickler compiliert*. Dabei entsteht ein *Zwischencode*, der sogenannte **Bytecode**. Dieser ist *nicht* ausführbar im obigen Sinn, dafür aber vollkommen *unabhängig* von der Hardware und dem Betriebssystem und kann über das *Internet* an andere Nutzer verschickt werden.
Ein *Interpreter* (**JVM ..** *Java Virtual Machine*) des *Nutzers* übernimmt die weitere Verarbeitung und muss deshalb für die konkrete Plattform vorhanden sein, auf der der Bytecodes ausgeführt werden soll.
Virtuelle Maschinen sind in **Browsern** integriert. Für eigenständige Anwendungen werden diese von verschiedenen Softwarefirmen angeboten. Die kostenlose Entwicklungsumgebung **J2SDK** der Firma Oracle stellt das Programm `java` als Interpreter bereit.
Um eine Steigerung der Geschwindigkeit (bis zu 20 %) zu erreichen, steht ein *Just in Time Compiler* (**JIT**) zur Verfügung. Dieser übersetzt das Programm beim ersten Aufruf zur Laufzeit und speichert den lauffähigen Code ab. Beim nächsten Aufruf wird ein erneutes Übersetzen überflüssig.

Schritte zum Erstellen und Ausführen eines Java-Programms



1. **Quellcode**: Mit einem beliebigen **Editor** wird das Java-Programm geschrieben und erhält einen Namen mit der Endung `.java`.

2. **Bytecode**: Anschließend wird er von einem **Compiler** `javac` übersetzt. Der entstandene **Bytecode** hat Endung `.class`.

3. **Ausführung**: Der Bytecode kann nun durch einen speziellen **Interpreter**, der sogenannten **Java Virtual Machin** (*JVM*), in die plattformabhängige Maschinensprache übersetzt und ausgeführt werden.

Ein *Vorteil* dieser Vorgehensweise ist die *Plattformunabhängigkeit* des Bytecodes, ein *Nachteil* der *Geschwindigkeitsverlust* durch den Interpreter bei der Ausführung des Bytecodes.

2.3 Grundlagen der Java-Programmierung

2.3.1 Installation von Java

Das *Java 2 Software Development Kit* **J2SDK** umfasst alle notwendigen Programme und Tools, die Standard-Klassenbibliothek und einige Demos. Zusätzlich wird noch ein Texteditor benötigt.

Installationsschritte:

1. **Java SE Development Kit - JDK 7** ist eine im uninstallierten Zustand ca. 36 MB große ausführbare Datei. Das Herunterladen (*SDK*) erfolgt direkt von der Oracle-Webseite **http://www.oracle.com/technetwork/java/javase/downloads/**.
2. Das heruntergeladene Installationsprogramm wird aktiviert, die Installation erfolgt problemlos und weitestgehend automatisch.
3. Anschließend sollte man die Systemvariable **PATH** um den Eintrag auf das zum Java gehörige *bin*-Verzeichnis erweitern. Dort befinden sich Compiler, Interpreter und andere nützliche Tools.
4. Unter **http://download.oracle.com/javase/7/docs/api/** finden Sie die Dokumentation der Klassen.

Weitere, umfangreichere Entwicklungswerkzeuge werden von Borland *JBuilder*, IBM *VisualAge* und anderen vertrieben. IBM bietet außerdem eine Public-Domain-Oberfläche *Eclipse*[1] an.

2.3.2 Java Application

Eine **Application** (Anwendung) ist ein *eigenständiges* Programm. Sie kann *unabhängig* vom Internet durch einen *Java-Interpreter* ausgeführt werden.

1. **Quellcode**:

HalloWeltApplication.java

```
// HalloWeltApplication.java                          MM 2014
/*                Mein erstes Programm                      */

/**
 * Dieses Programm gibt den Schriftzug "Hallo Welt!"
 * auf der Konsole aus.
 */
public class HalloWeltApplication
{
/**
 * Hauptmethode, erzeugt Bildschirmausschrift
 */
  public static void main( String[] args)
  {
    System.out.println( "Hallo Welt!");  // Konsolenausgabe
  }
}
```

[1] http://www.eclipse.org/

Ein Java-Programm ist in **Blöcke** strukturiert. Blöcke sind Anweisungen, die in **{** und **}** eingeschlossen wurden. Im Beispiel sind zwei Blöcke *ineinander geschachtelt*:

`class`
Eine Klasse `HalloWeltApplication` ist die **oberste Struktureinheit** des Programms. Deren Name *muss* mit dem Namen des Programms übereinstimmen, `HalloWeltApplication.java`.
Das Schlüsselwort `public` sagt aus, dass die Klasse `HalloWeltApplication` eine **öffentlich** zugängliche Klasse ist.

`main`
Innerhalb einer Klasse gibt es **untergeordnete Struktureinheiten, wie Attribute und Methoden**. Jede Klasse, die wie diese ein ausführbares Programm darstellen soll, besitzt die **Hauptmethode** `main`.

2. **Bytecode**:

```
$> javac HalloWeltApplication.java        => HalloWeltApplication.class
```

3. **Ausführung**:

```
$> java HalloWeltApplication
Hallo Welt!
```

Man beachte, dass Groß- und Kleinschreibung signifikant sind!

2.3.3 Java Applet

Ein **Applet** ist eine kleines Programm, welches *zusammen* mit einer **HTML-Seitenbeschreibung** (*HTML HyperText Markup Language*) auf einem **Web-Server** als *Bytecode* abgelegt und bei Abruf an einem **Web-Client** gesendet wird. Auf einem **Web-Client** kann der empfangene Bytecode von einem im **Browser** integrierten *Java-Interpreter* ausgeführt werden. Damit ist es möglich, nicht nur statische Inhalte, sondern auch *Animationen* auf Web-Seiten unterzubringen.

1. **Quellcode**:

HalloWeltApplet.java

```
// HalloWeltApplet.java                                    MM 2014

import java.applet.*;                                    // Applet
import java.awt.*;                                      // Graphics

/**
 * Dieses Applet gibt den Schriftzug "Hallo Welt!" aus.
 */
public class HalloWeltApplet extends Applet
{
/**
```

```
   * Hauptmethode, Darstellung des Applet.
   */
    public void paint( Graphics g)
    {
      g.drawString( "Hallo Welt!", 50, 50);
    }
}
```

`extends Applet`

Das Programm stellt ein Applet für eine Internetanwendung dar.

`paint`

Die Methode `paint` wird für den Bildschirmaufbau in einem Applet aufgerufen. Sie entspricht der Methode `main` in einer Applikation. Der Darstellungsbereich `Graphics g` wird ihr vom Browser übergeben.

Ein Applet kann *nicht* eigenständig ausgeführt werden. Um das Applet in einem Browser darstellen zu können, muss es in einer HTML-Seite[2] eingebunden werden.

HalloWeltApplet.html

```
<html>
<!-- Diese Seite bindet das HalloWelt - Applet ein. -->
  <head>
    <title>HalloWeltApplet
    </title>
  </head>

  <body>
<!-- Applet -->
    <applet
      code=HalloWeltApplet.class width=170 height=100>
    </applet>
  </body>
</html>
```

2. **Bytecode**:

```
$> javac HalloWeltApplet.java          => HalloWeltApplet.class
```

3. **Ausführung**:

[2] HTML-Kurzreferenz http://de.selfhtml.org/
HTML-Gerippe http://werbach.com/barebones/BBG-form.html

Es gibt mehrere Möglichkeiten zur Darstellung des Applets:

- **Aufruf des Appletviewer**

```
$> appletviewer HalloWeltApplet.html
```

- **Öffnen der HTML-Seite im Browser**

Web-Server (Java Compiler)

.html
.class

Web-Server (Java Interpreter)

Alle für das Applet benötigten Klassen müssen übersetzt auf dem Web-Server im Verzeichnis der HTML-Seite stehen!

2.3.4 Java Script

Java Script erweitert die Anwendungsbereiche eines Applets, wie zum Beispiel Interaktionen mit Nutzern und *hat wenig mit Java zu tun*. Diese Sprache erweitert die Möglichkeiten von HTML, der Quelltext wird dort direkt eingebunden. Die Syntax ist an C/C++ bzw. Java angelehnt. *Die Verarbeitungsschritte entsprechen denen eines Applets.* Die Ausführung ist aber *nur* in einem Browser, *nicht* mit dem Appletviewer, möglich!

Ampelstellung: 1 .. rot, 2 .. gelb, 3 .. grün, 0 .. defekt

AmpelApplet.java

```
// AmpelApplet.java                                    MM 2014

import java.applet.*;                                  // Applet
import java.awt.*;                                  // Graphics, Color

/**
 * Dieses Applet stellt eine Ampelsteuerung dar.
 * JavaScript ruft Methoden des Applets auf und
```

```
 * steuert dadurch diese von aussen.
 */
public class AmpelApplet extends Applet
{
/**
 * Ampelstellung:
 * 1 .. rot, 2 .. gelb, 3 .. gruen, 0 .. defekt
 */
 int lampe = 0;

/**
 * Hauptmethode, Ampel zeichnen.
 */
  public void paint( Graphics g)
  {
    setBackground( Color.gray);
    switch( lampe)
    {
      case 1:  g.setColor( Color.red);
               g.fillOval( 5, 15, 40, 40);
               g.setColor( Color.white);
               g.fillOval( 5, 65, 40, 40);
               g.fillOval( 5, 115, 40, 40);
               break;
      case 2:  g.setColor( Color.yellow);
               g.fillOval( 5, 65, 40, 40);
               g.setColor( Color.white);
               g.fillOval( 5, 15, 40, 40);
               g.fillOval( 5, 115, 40, 40);
               break;
      case 3:  g.setColor( Color.green);
               g.fillOval( 5, 115, 40, 40);
               g.setColor( Color.white);
               g.fillOval( 5, 15, 40, 40);
               g.fillOval( 5, 65, 40, 40);
               break;
      default: g.setColor( Color.red);
               g.fillOval( 5, 15, 40, 40);
               g.setColor( Color.white);
               g.fillOval( 5, 65, 40, 40);
               g.fillOval( 5, 115, 40, 40);
    }
  }

/**
 * Ampelstellung aendern, wird von Javascript aufgerufen.
 */
  public void setLampe( int nr)
  {
    lampe = nr; repaint( 100L);
  }
}
```

Die Methode `repaint` veranlasst das Aktualisieren des Appletobjekts mittels `paint` innerhalb von 0.1 sec.

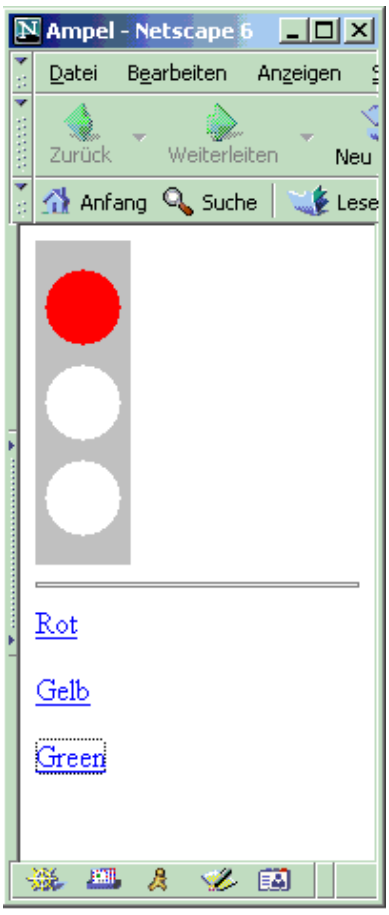
AmpelApplet.html

```
<html>
<!--   Diese Seite bindet das Ampel - Applet ein.   -->
  <head>
    <title>Ampel
    </title>
  </head>

  <body>
<!--   Applet   -->
    <applet code=AmpelApplet.class name=A
            width=50 height=170>
    </applet>

    <hr>
<!--   Javascript   -->
    <a onMouseOver="document.A.setLampe( 1)">Rot
    </a><p>
    <a onMouseOver="document.A.setLampe( 2)">Gelb
    </a><p>
    <a onMouseOver="document.A.setLampe( 3)">Green
    </a>
  </body>

</html>
```



Das Applet ist hier ein Objekt der Klasse `AmpelApplet` mit dem Namen *A*. Über `document.A.setLampe()` greift das aktuelle Dokument `document`, diese Web-Seite, auf die Methode `setLampe()` des Objektes *A* zu.

2.4 Ein- und Ausgaben

Getreu dem **EVA-Prinzip** (*Eingabe* - *Verarbeitung* - *Ausgabe*) der elektronischen Datenverarbeitung ist es notwendig, geeignete Ein- und Ausgaberoutinen für Anwendungen zur Verfügung zu stellen.

2.4.1 Ausgabe

Textausgabe

Eine Methode zur **Textausgabe** wurde schon in ***HalloWeltApplication.java*** verwendet.

- Bildschirmausgabe mit anschließendem Zeilenvorschub:

```
System.out.println( "Hallo Welt!");          // Hallo Welt!
```

- Bildschirmausgabe ohne anschließendem Zeilenvorschub
  (+ mit drei verschiedenen Interpretationen):

```
int a = 3, b = 5;
System.out.print                    // Ich rechne: 3 + 5 = 8
("Ich rechne: " + a + " + " + b + " = " + (a + b));
```

`System.out.println` und `System.out.print` sind **Methoden** aus der **Java-Standardbibliothek** mit denen man *Text* und *Zahlen* in einem Anwendungsfenster, standardmäßig einer **Konsole**, ausgeben kann. Der auszugebende Text steht in der Methode als Argument in Anführungsstrichen. Der *Operator* **+** wird in Abhängigkeit der *Operanden* mit drei verschiedenen Interpretationen verwendet. Mit ihm können *Texte* verknüpft und *Zahlen* addiert werden. **+** kann aber auch als *Zeichen* in einem Text stehen.

Fehlertextausgabe

Analog verwendet man die Methoden `System.err.println` und `System.err.print` zur **Fehlertextausgabe** standardmäßig auf einer **Konsole**.

```
System.err.print( "Fehler! ");
System.err.println( "Bitte nur ganze Zahlen eingeben!");
```

2.4.2 Eingabe

Etwas komplizierter sind Methoden zur Eingabe über eine Tastatur zu realisieren. Eine Eingabe einer ganzen Zahl erfordert den folgenden Quelltext:

IntegerEingabe.java

```
// IntegerEingabe.java                                  MM 2014

import java.io.*;                                  // Datenstroeme

/**
 * Integereingabe ohne Klasse IOTools.
 */
```

```
public class IntegerEingabe
{
 public static void main( String[] args)
  {
    int eingabe;          // Vereinbarung einer ganzen Zahl
                                        // Tastatureingabestrom
    BufferedReader in = new BufferedReader
                      ( new InputStreamReader( System.in));
    while( true)
    {
      System.out.print
      ( "Bitte eine ganze Zahl eingeben! ");
      try
      {
                       // Tastatureingabe einer ganzen Zahl
        int eingabe = Integer.parseInt( in.readLine());
        break;
      }
      catch( NumberFormatException e)
      {                     // Verletzung des Zahlenformats
        System.err.println( "Eingabefehler " + e);
        System.err.println
        ( "Bitte Eingabe wiederholen...!");
      }
      catch( Exception e)
      {                                    // andere Fehler
        System.err.println( "Fehler! " + e);
      }
    }
                  // Konsolenausgabe der eingelesenen Zahl
    System.out.println( "Eingegebene Zahl: " + eingabe);
  }
}
```

Zunächst wird ein Datenstrom zum Zeichenlesen von der Tastatur aufgebaut. Eine Zeichenkette wird von der Tastatur gelesen und in eine ganze Zahl umgewandelt. Treten dabei Fehler auf, so wird mit einem entsprechenden Fehlerhinweis reagiert. Die Eingabe wird solange wiederholt, bis sie korrekt ist. Die eingegebene Zahl wird auf der Konsole ausgegeben.

Um dieses Eingabekonzept verstehen und anwenden zu können, sind umfangreiche Kenntnisse über objektorientiertes Programmieren und Datenströme notwendig. Deshalb ist es zweckmäßig, dem Anfänger eine Klasse zur Verfügung zu stellen, welche gängige Eingaberoutinen zusammenfasst. Da eine solche innerhalb eines Pakets[3] bereits existiert, nutzen wir diese und noch einige weitere Klassen im Sinne der Wiederverwendung. Außerdem haben wir eine Klasse `Euklid` und eine Klasse `MVC` hinzugefügt. Ein so zusammengefasstes Paket `Tools` kann jederzeit erweitert werden. Ein Link befindet sich auf der Homepage zu diesem Kurs.

[3] Dietmar Ratz, Jens Scheffler, Detlef Seese: *Grundkurs Programmieren in Java, Bd 1*, Hanser Verlag

2.4.3 Installation des Pakets `Tools`

Paket `Tools` **`Tools.zip`**

`Tools.IO.IOTools`:	Klasse für Tastatureingaberoutinen
`Tools.Euklid.Euklid`:	Klasse für ggT, kgV mittels Algorithmus von Euklid
`Tools.MVC.*`:	Klassen für MVC-Architektur (später)
`Tools.Game.*`:	Klassen für Brettspiele als Applet und Application

Installationsschritte:

1. In ein Verzeichnis (hier: `Java.dir`) wird das komprimierte Paket **`Tools.zip`** heruntergeladen, ohne es auszupacken.
2. Zum Einbinden des Pakets `Tools` in andere Programme muss die Umgebungsvariable `CLASSPATH` auf `Tools.zip` gesetzt werden.
   Windows: `CLASSPATH=.;C:\...\Java.dir\Tools.zip`
   Unix(Linux): `export CLASSPATH=.:/.../Java.dir/Tools.zip`
3. Jedes Java-Programms, welches Klassen des Pakets `Tools` verwendet, muss diese am Anfang des Programms einbinden, für die Eingaberoutinen mit der Anweisung
   `import Tools.IO.IOTools;`
   oder auch `import Tools.IO.*;`

Dokumentation: **Doc**
Beispiele: **Summe.java**, **GgTKgV.java**

2.4.4 Methoden der Klasse `IOTools`

Nach diesen Vorbereitungen kann die Klasse `IOTools` für Eingaben verwendet werden.

Lesen eines Zeichen:	`readChar()`
mit Eingabeaufforderung:	`readChar( String prompt)`
Lesen eines Wortes:	`readString()`
mit Eingabeaufforderung:	`readString( String prompt)`
Lesen einer Zeile:	`readLine()`
mit Eingabeaufforderung:	`readLine( String prompt)`
Lesen einer kurzen ganzen Zahl:	`readShort()`
mit Eingabeaufforderung:	`readShort( String prompt)`
Lesen einer ganzen Zahl:	`readInteger()`
mit Eingabeaufforderung:	`readInteger( String prompt)`
Lesen einer langen ganzen Zahl:	`readLong()`
mit Eingabeaufforderung:	`readLong( String prompt)`
Lesen einer gebrochenen Zahl:	`readFloat()`
mit Eingabeaufforderung:	`readFloat( String prompt)`

Lesen einer gebrochenen Zahl
doppelter Genauigkeit: `readDouble()`
mit Eingabeaufforderung: `readDouble( String prompt)`

Lesen eines Wahrheitswertes `readBoolean()`
mit Eingabeaufforderung: `readBoolean( String prompt)`

Die folgenden Beispiele soll den Umgang mit der Klasse `IOTools` demonstrieren. Es wird noch einmal die Eingabe eines Integer programmiert, aber mit der Methode `readInteger` der Klasse `IOTools`.

IOToolsIntegerEingabe.java

```
// IOToolsIntegerEingabe.java                          MM 2014

import Tools.IO.*;                                  // IOTools

/**
 * Integereingabe unter Verwendung der Klasse IOTools.
 */
public class IOToolsIntegerEingabe
{
  public static void main( String[] args)
  {
    int eingabe = IOTools.readInteger
                  ( "Bitte eine ganze Zahl eingeben! ");
    System.out.println( "Eingegebene Zahl: " + eingabe);
  }
}
```

Summe.java

```
// Summe.java                                          MM 2014

import Tools.IO.*;                                  // IOTools

/**
 * Berechnung der Summe von Double-Zahlen,
 * Anwendung des Pakets Tools.
 */
public class Summe
{
  public static void main( String[] args)
  {
    double summe = 0, zahl;
    int count = 0;
    char weiter;

    do
    {
      zahl = IOTools.readDouble( ++count + ". Zahl: ");
```

```
        summe += zahl;

        weiter = IOTools.readChar( "Weiter(j/n)? ");
      } while( weiter == 'j');

      System.out.println
      ( "Summe: " + summe + " Anzahl: " + count);
    }
  }
```