Informationstechniken

Programmierung interaktiver Systeme

(Skript im Aufbau, auch bereits vorhandene Seiten werden unter Umständen noch verändert)

Prof. Dr. Wolf-Fritz Riekert
Hochschule für Bibliotheks- und Informationswesen (HBI) Stuttgart

mailto:riekert@hbi-stuttgart.de
http://v.hbi-stuttgart.de/~riekert

Literatur (1)

Ken Arnold und James Gosling: *The Java Programming Language*. 2nd edition. Addison-Wesley, 1999.

Dietrich Boles: *Programmieren spielend gelernt* *(mit dem Java-Hamster-Modell).* B.G. Teubner Stuttgart-Leipzig, 1999. Die ersten Kapitel, die auch im Web abrufbar sind, geben eine sehr gute und kompakte Einführung in das Programmieren. (http://www.is-informatik.uni-oldenburg.de/~dibo/hamster/).

Java-Tutorial der Firma Sun. Online-Nutzung und Download: http://java.sun.com/docs/books/tutorial/

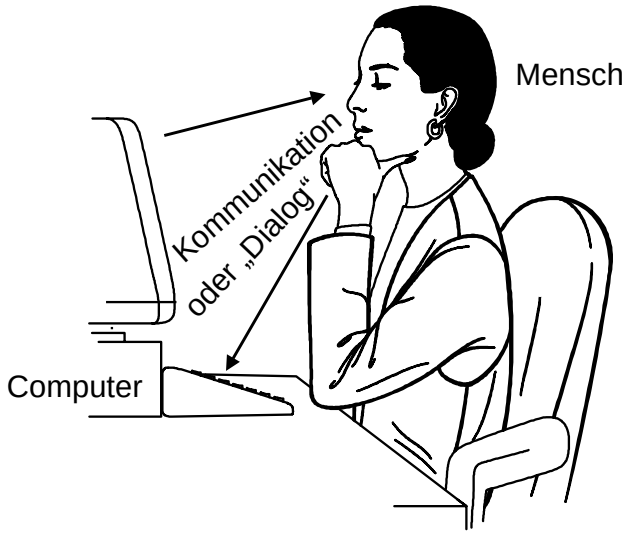
Reinhard Schiedermeier: *Programmieren in Java*. Vorlesungsskript, Fachhochschule München, 2000. Online-Nutzung und Download über: http://www.informatik.fh-muenchen.de/~schieder/

Literatur (2)

Dirk Louis, Peter Müller: *Jetzt lerne ich Java - der einfache Einstieg in die Internetprogrammierung*. Markt und Technik Verlag, Stuttgart, 2000. Preisgünstiges Buch, für Anfänger geeignet, umfasst Java 2.

Christian Wolff, Einführung in Java. *Objektorientiertes Programmieren mit der Java 2-Plattform*. B.G. Teubner Stuttgart-Leipzig, 1999. Sehr detailreiches und sorgfältig ausgearbeitetes Buch über Java und objektorientierte Programmierung für die intensive Beschäftigung mit dem Thema.

Computerbenutzung als Mensch-Computer-Kommunikation



Mensch-Computer-Kommunikation: Äußerungsformen

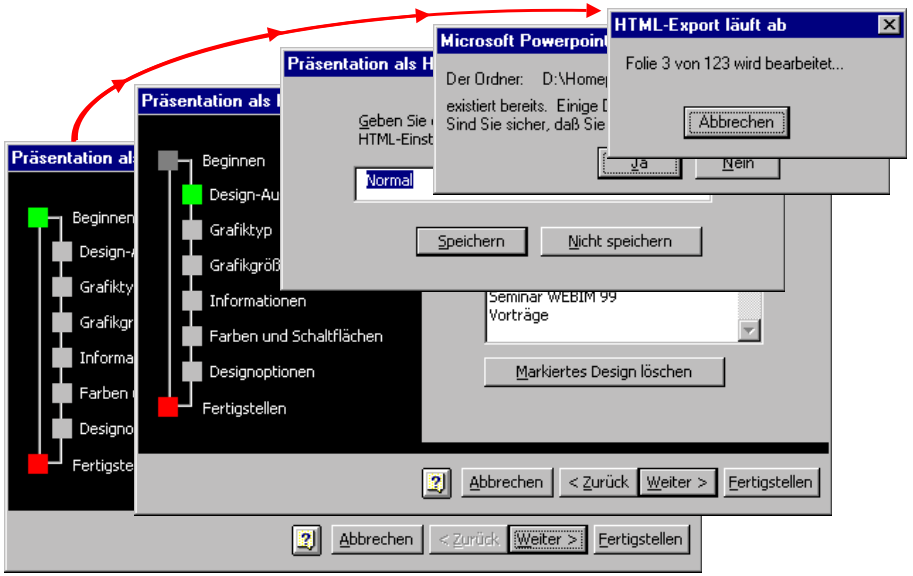
Äußerungsformen des Menschen

- Text über Tastatur
- Zeigeaktionen über die Maus
- (Sprache über Mikrophon)
- (Mimik über Videokamera)

Äußerungsformen des Computers

- Text,
- Graphik,
- Video über Bildschirm;
- Audio über Lautsprecher

Computergesteuerter Dialog Beispiel: HTML-Export aus Powerpoint



Computergesteuerter Dialog

Kennzeichen des computergesteuerten Dialogs:

- Computer stellt Fragen an Benutzer(in)*
- Benutzer beantwortet Fragen
- Dasselbe von vorn, bis Computer „zufrieden“

Vorteil:

- Benutzer wird geführt

Nachteil:

- Freiheit des Benutzers wird eingeengt

Worin besteht die Einengung des Benutzers?

** Der Einfachheit halber nachfolgend nur noch „der Benutzer“, Benutzerinnen gelten als implizit eingeschlossen.*

Das Nievergeltsche Dialogmodell

Laut Nievergelt (1980) ist der Dialog durch drei Dimensionen bestimmt

- **Site**: Die im Moment ansprechbaren Daten
- **Mode**: Die im Moment ausführbaren Funktionen
- **Trail**: Der zeitliche Fortschritt des Dialogs

Gute Dialogsysteme

- machen alle drei Dimensionen (Site, Mode und Trail) sichtbar
- erlauben eine einfache Fortbewegung in Site, Mode und Trail

Einengung der Benutzer durch computergesteuerte Dialoge

Streng computergesteuerte Dialoge schränken alle drei Dimensionen ein:

- Es ist unmöglich, andere Daten als die aktuell angebotenen zu bearbeiten
- Es steht nur eine kleine Auswahl von Funktionen zur Verfügung (oft nur eine, die mit ja/nein ausgewählt wird)
- Einmal getroffene Entscheidungen können nicht mehr rückgängig gemacht werden

Trend in Richtung benutzergesteuerter Dialoge

Gewünscht wird, in Mode, Site und Trail zu navigieren:

- Zwischen Daten muss frei gewechselt werden können (z.B. zwischen einem Brieftext und einer Adressdatei)
- Die verfügbaren Funktionen müssen vor dem Benutzer „ausgebreitet“ sein, z.B. in einer Menüleiste
- Die Dialoghistorie soll nachvollziehbar aufgezeichnet werden. Arbeitsschritte müssen rückgängig gemacht werden können.

Konsequenz:

- Seit den 80-er Jahren zunehmender Trend in Richtung benutzergesteuerter Dialoge

Benutzergesteuerter Dialog Beispiel: Windows Desktop

Benutzergesteuerter Dialog

Kennzeichen des benutzergesteuerten Dialogs

- Benutzer gibt einen Befehl an den Computer
- Computer führt Befehl aus
- Dasselbe von vorn, bis Benutzer zufrieden

Errungenschaften des benutzergesteuerten Dialogs:

- Kommandointerpreter (z.B. DOS, UNIX Shell)
- Menüs, Formulare
- Direkte Manipulation: direktes Übertippen von Textfeldern, „Anfassen“ und Verändern von Objekten durch Mauseinsatz
- Fenstersysteme, graphische Symbole (Icons)
- sogenannte objektorientierte Benutzungsoberflächen

Mischformen

Wenn bestimmte Abfolgen eingehalten werden müssen, können computergesteuerte Dialogformen prinzipiell angebracht sein:

- Beispiel: Softwareinstallation

Doch auch hier kommen zunehmend Elemente der Benutzersteuerung zum Zuge:

- Beispiel: sogenannte Assistenten
  - ⇨ Zwar wird eine Abfolge von Abfragefenstern angeboten
  - ⇨ Aber man kann meist zu früheren Fenstern zurückspringen

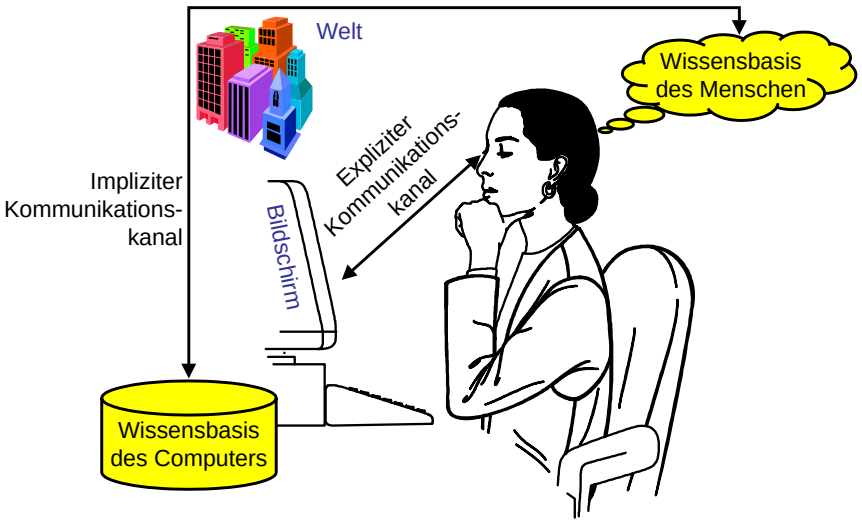
Software-Ergonomie

- Mitte der 80-er Jahre wurde **Software-Ergonomie** als Ziel benutzerfreundlicher Systemgestaltung proklamiert
- Ziel der Software-Ergonomie ist die **Verringerung mentaler Belastungen** bei der Benutzung von Computerprogrammen
- Gegensatz: klassische (Hardware-)Ergonomie, die die Verringerung körperlicher Belastungen zum Ziel hat, etwa durch körpergerechte Bürostühle, Tische, Tastaturen oder Bildschirmpositionen, augenschonende Bildröhren usw.
- Beispiel: Grundsätze der Dialoggestaltung nach DIN 66234 Teil 8 und ISO 9241 Part 10

Grundsätze der Dialoggestaltung angelehnt an DIN 66234 Teil 8

- **Aufgabenangemessenheit**
  - ⇨ Erledigung von Aufgaben ohne unnötige Belastung durch Eigenschaften des Dialogsystems
- **Selbstbeschreibungsfähigkeit**
  - ⇨ Erläuterungen durch das System, Hilfefunktion
- **Steuerbarkeit**
  - ⇨ Dialog soll durch Benutzer beeinflussbar sein
- **Erwartungskonformität**
  - ⇨ Dialog soll Erwartungen entsprechen, die aus bereits gemachten Erfahrungen resultieren
- **Fehlerrobustheit**
  - ⇨ verständliche Fehlermeldungen, leichte Behebbarkeit von Bedienungsfehlern

Wissensbasierte Mensch-Computer-Kommunikation



Was ist besonders an der Objektorientiertheit?

Vor allem die Betrachtungsweise:

Klassisch:

- Computer enthält **Programme** und **Daten**
- Computerbenutzung = Aufruf von Programmen
- Programme erzeugen, verändern und löschen die Daten

Objektorientiert:

- Computer enthält **Objekte**
- Objekte werden durch **Botschaften** aktiviert
- Objekte besitzen **Methoden**: Sie können
  - ⇨ ihren Zustand ändern und
  - ⇨ Botschaften an andere Objekte schicken

Objekte

- gehören einer **Klasse** an (z.B. Buch)
- haben eine eindeutige **Identität** (z.B. ISBN, Inventarnr.)
- besitzen **Merkmale** und **Merkmalswerte**, z.B.:
  - ⇨ Titel = „Objektorientierte Analyse und Design“
  - ⇨ Autor = „Booch, G.“
  - ⇨ ...

  (Merkmalswerte können Zahlen, Texte oder andere Objekte sein)
- zeigen „Verhalten“, d.h. sie besitzen **Methoden**, z.B.
  - ⇨ bestellen
  - ⇨ entleihen
  - ⇨ ...

Klassen

- Objekte gehören **Klassen** an. Man sagt, sie sind „Instanzen“, der Klassen, denen sie angehören.
- Klassen sind **Abstraktionen** ihrer Instanzen. Sie legen die Merkmale und Methoden ihrer Instanzen fest.
- Die Klassen bilden eine **Hierarchie**
  - ⇨ Die Hierarchie bedeutet **Spezialisierung** oder **Generalisierung** (je nach Blickrichtung).
  - ⇨ Die Festlegungen von Merkmalen und Methoden werden entlang der Klassenhierarchie **vererbt**.
- Die Klassen stellen „**Fabriken**“ dar zur Erzeugung ihrer Instanzen.

Objektorientierte Werkzeuge

- Werkzeuge zur objektorientierten **Analyse** und **Modellierung**
- Objektorientierte **Programmiersprachen** zur Definition von Objektklassen, -merkmalen und -methoden (z.B. C++, Java, Visual Basic)
- Objektorientierte und objektrelationale **Datenbanken** zum Speichern und Abfragen von Objekten
- Objektorientierte **Netzwerksoftware** (z.B. CORBA: „Common Object Request Broker Architecture“ oder Java RMI: „Remote Method Invocation“)
- Werkzeuge zur Konstruktion von objektorientierten **Benutzungsoberflächen**

Objektorientierte Benutzungsoberflächen

Interaktionsobjekte dienen zur Darstellung der **Anwendungsobjekte** (im Fall einer bibliothekarischen Datenbank z.B.: Publikationen, Kataloge, Katalogeinträge) auf dem Bildschirm

- Beispiele: Fenster, Icons, Menüs, Textfelder, Grafiken
- Softwaretechnisch sind Interaktionsobjekte auch Objekte (im Sinne der objektorientierten Programmierung), die mit den Anwendungsobjekten kommunizieren
- **Direkte Manipulation** vermittels Interaktionsobjekten
  - ⇨ Aktivierung mit Maus und Tastatureingabe
  - ⇨ wirkt sich auf Anwendungsobjekte aus
  - ⇨ Änderungen sofort an Anwendungsobjekten sichtbar

Applets: Beispiele für Objekte

- Applets sind eine besondere Art von Java-Objekten, die als Interaktionsobjekte für die Programmierung von Benutzungsschnittstellen verwendet werden
- Sie liegen in der Klassenhierarchie unter der Klasse Applet
- Applets werden in HTML-Seiten eingebettet
  - ⇨ gekennzeichnet durch ein <Applet>-Tag
  - ⇨ und durch eine URL bezeichnet
- Applets besitzen Methoden, die ihr Verhalten an der Benutzungsoberfläche beschreiben, z.B.
  - ⇨ Init(), Start(), Stop(), Destroy() (*Lebenszyklus des Applets*)
  - ⇨ paint() (*Zeichenfunktion des Applets*)
  - ⇨ mouseClicked(), MouseEntered() usw. (*Reaktion auf Benutzerinteraktionen*) ...

Ein Zählerapplet: Beispiel eines Java-Objekts

Java-Code des Zählerapplets

```
public class ZaehlerApplet extends MouseListenerApplet {

   private int zaehlerstand = 0;

   public void mouseClicked(MouseEvent e){
      zaehlerstand = zaehlerstand + 1;

      // Zeichenflaeche des Applets neu zeichnen
      repaint();
   }

   public void paint(Graphics g){
      // Rahmen um das Applet zeichnen
      g.drawRect(0, 0, getSize().width-1,
                       getSize().height-1);

      // Zaehlerstand ausgeben
      g.drawString(String.valueOf(zaehlerstand),5,15);
   }
}
```

Einbettung des Applets in eine HTML-Seite

```
<HTML>
 <HEAD>
   <TITLE> ZaehlerApplet </TITLE>
 </HEAD>
 <BODY>
   <APPLET code='ZaehlerApplet.class'
           width='25'
           height='25'>
   </APPLET>
 </BODY>
</HTML>
```

Applet-Tag

URL bzw. Dateiname des Applet-codes

Räumliche Ausdehnung des Applets

Java-Code der Oberklasse MouseListenerApplet

```
public class MouseListenerApplet
          extends Applet implements MouseListener {
     public void init() {addMouseListener(this);}
     public void start(){}
     public void stop() {}
     public void destroy() {}
     // Folgende Methoden implementieren MouseListener
     public void mouseClicked(MouseEvent e) {}
     public void mouseEntered(MouseEvent e) {}
     public void mouseExited(MouseEvent e) {}
     public void mousePressed(MouseEvent e) {}
     public void mouseReleased(MouseEvent e) {}
     // Zeichenroutine des Applets
     public void paint(Graphics g) {}
}
```


Definitionen

Algorithmus: Arbeitsanleitung zum Lösen eines Problems oder einer Aufgabe, die so präzise formuliert ist, dass sie im Prinzip auch von einem Computer ausgeführt werden kann.

Programmablaufpläne (Flussdiagramme) und **Struktogramme** (Nassi-Shneidermann-Diagramme) dienen zur graphischen Darstellung von Algorithmen.

Programmiersprachen dienen zur Formulierung von Algorithmen.

Ein in einer Programmiersprache formulierter Algorithmus heißt **Programm**.

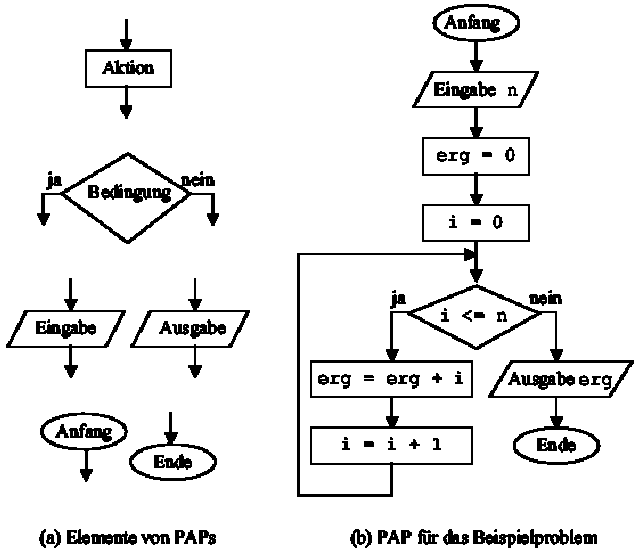
In Form von Programmen können Algorithmen durch einen **Computer** ausgeführt werden.

Ein Beispielproblem

Gegeben sei eine natürliche Zahl n

Bestimme die Summe der natürlichen Zahlen von 0 bis n

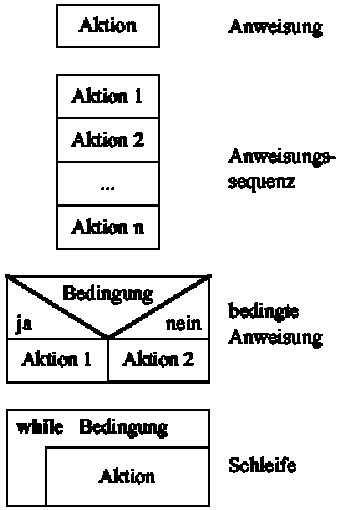
Programmablaufpläne (Flussdiagramme)

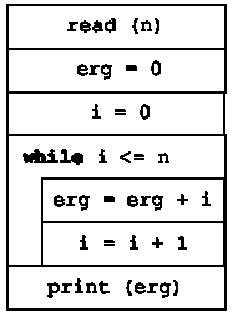


(a) Elemente von PAPs (b) PAP für das Beispielproblem

Quelle:
Dietrich Boles:
Programmieren
spielend gelernt.
B.G. Teubner,
Stuttgart - Leipzig
1999c

Struktogramme (Nassi-Shneiderman-Diagramme)





(a) Elemente von Struktogrammen (b) Struktogramm für Beispielproblem

Quelle:
Dietrich Boles:
Programmieren
spielend gelernt.
B.G. Teubner,
Stuttgart - Leipzig
1999c

Programm

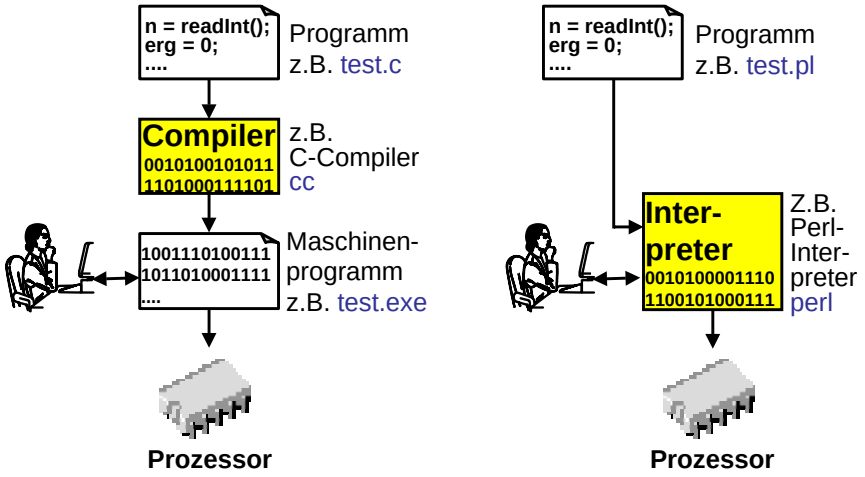
```
int n = readInt();
int erg = 0;
int i = 0;
while (i <= n) {
        erg = erg + i;
        i = i + 1;
        }
printInt(erg);
```


Art der Ausführung eines Programms durch den Computer

- Ein Prozessor (z.B. der Pentium-Prozessor) kann nur sogenannte **Maschinenprogramme** ausführen, diese bestehen aus einer Folge von Zahlencodes.
- Menschen können Maschinenprogramme mit Hilfe einer sogenannten **Assemblersprache** schreiben, dabei sind die Zahlencodes durch Namen von Maschinenbefehlen ersetzt.
- Meist werden Programme aber in **höheren Programmiersprachen** geschrieben (z.B. C, Java, Modula)
  - ⇨ Sie werden dann entweder mit einem **Compiler** in Maschinenprogramme übersetzt und anschließend durch den Prozessor ausgeführt.
  - ⇨ oder durch einen sogenannten **Interpreter** ausgeführt.

Compiler und Interpreter sind selbst Maschinenprogramme.

Compiler und Interpreter



Eigenschaften von Compilersprachen

- Der Compiler wird nur bei der Programmentwicklung gebraucht, im Betrieb läuft das übersetzte Programm ab.
- Deshalb hat der Compiler Zeit für aufwendige Programmüberprüfungen und Optimierungen.
- Compiler überprüfen Programme hinsichtlich „Vokabular“ und „Grammatik“, so dass viele Programmierfehler bereits bei der Compilierung entdeckt werden können.
- Durch Compiler übersetzte Programme sind Maschinenprogramme, die in der Regel sehr schnell und effizient ablaufen.
- Die Programmentwicklung mit Compilersprachen ist etwas mühevoll, da ein Programm nach jeder Änderung neu compiliert werden muss.
- Beispiele für Compilersprachen: C/C++, Visual Basic (VB)

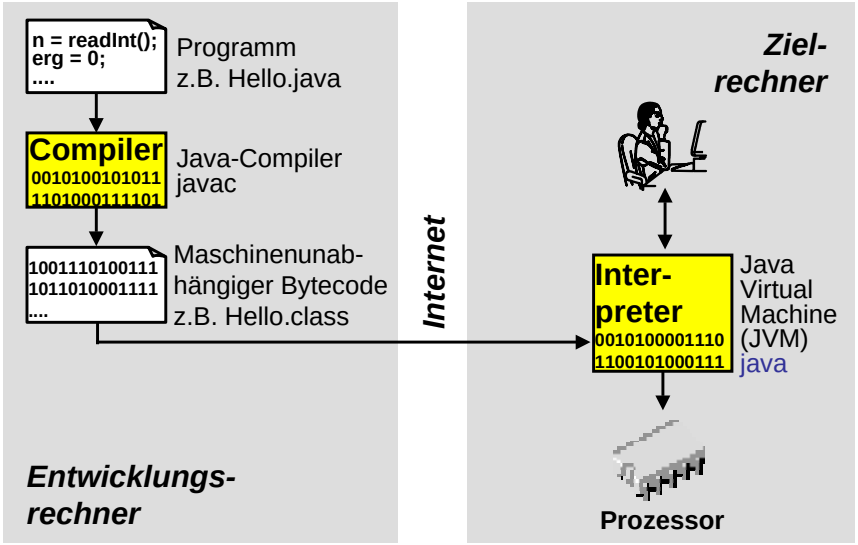
Eigenschaften von Interpretersprachen

- Interpreterprogramme funktionieren nicht für sich alleine, sie benötigen zur Ausführung einen Interpreter.
- Da der Interpreter zur Laufzeit des Programms aktiv ist, hat er wenig Zeit für aufwendige Prüfungen. Fehlerhafte Programme „stürzen“ oft mit einer kurzen Meldung „ab“.
- Es gibt jedoch Programmentwicklungsumgebungen mit speziellen Editoren, die Syntaxüberprüfungen vornehmen.
- Interpretierte Programme sind deutlich langsamer als compilierte, was mit den heutigen schnellen Computern allerdings kein großes Problem mehr darstellt.
- Die Programmentwicklung ist erleichtert, da Programme nach Änderungen sofort wieder gestartet werden können.
- Beispiele für Interpretersprachen: Perl, Visual Basic for Applications (VBA), Javascript

Ausführung von Java-Programmen

- Java benötigt sowohl einen **Compiler** als auch einen **Interpreter**.
- Der Compiler übersetzt ein Java-Programm in einen sogenannten maschinenunabhängigen **Bytecode**.
- Dieser Bytecode wird nicht von einem Prozessor ausgeführt, sondern von einem Interpreter, der **Java Virtual Machine** (JVM).
- Die JVM ist ein Maschinenprogramm, das direkt vom Prozessor ausgeführt wird.
- Vorteil des Verfahrens: Der Bytecode ist **maschinenunabhängig**. Er kann auch auf einem fremden Computer ausgeführt werden. Nur die JVM muss für jeden Prozessortyp angepasst werden.

Java-Ausführungsmodell



Die Java-Entwicklungsumgebung J2SDK

- Die Firma Sun stellt eine kostenfrei nutzbare Java-Entwicklungsumgebung bereit, den Java® 2 Software Development Kit (J2SDK, früherer Name JDK™ 1.2).
- Ein Installationsprogramm (ca. 20MB) und Dokumentation (ca. 16MB) kann im Web heruntergeladen werden (http://java.sun.com/products/jdk/1.2/).
- Beim Start des Installationsprogramms (Doppelklick im Explorer) wird die Java-Software auf ein Zielverzeichnis (z.B. D:\java) entpackt. Es entsteht kein Eintrag in die Windows-Registrierdatenbank (Registry).
- Die Java-Dienstprogramme (Compiler, Interpreter usw.) liegen auf dem Unterverzeichnis \bin des Zielverzeichnisses.

Benutzung der Java-Entwicklungsumgebung J2SDK

- Die Java-Entwicklungsumgebung J2SDK wird über die DOS-Eingabeaufforderung kommandoorientiert genutzt.
- In der HBI ist Java bereits vorinstalliert auf **`U:\user\riekert\java`**
- Da die Java-Dienstprogramme auf dem Unterverzeichnis **`\bin`** liegen, legt man nach dem Start der DOS-Eingabeaufforderung am besten einen Pfad dorthin:
  `path %path%;u:\user\riekert\java\bin`
- Wenn Sie die Entwicklungsumgebung woanders installiert haben, setzen Sie den Pfad entsprechend dorthin.
- Durch die Festlegung des Pfades können Sie die Java-Dienstprogramme (Compiler, Interpreter usw.) mit ihrem bloßen Namen (javac, java usw.) aufrufen..

Typische Java-Entwicklungssequenz

DOS-Eingabeaufforderung starten

Java-Umgebung in Pfadvariable einschließen:

```
path %path%;u:\user\riekert\java\bin
```

Java-Klasse `Hello` *im Editor erstellen:*

```
notepad Hello.java
```

Java-Compiler übersetzt `Hello.java` *, dabei entsteht der Bytecode* `Hello.class`

```
javac Hello.java
```

Java-Interpreter interpretiert den Bytecode der Klasse `Hello`*:*

```
java Hello
```

Das Java-Programm macht die Ausgabe:

```
Hello World
```

Zum Überarbeiten des Programms zurück zum Editor usw.

Das „Hello World“-Programmbeispiel

```
public class Hello{
    public static void main(String[] args) {
        System.out.println("Hello World");
    }
}
```


Java-Syntax am Beispiel Zählerapplet

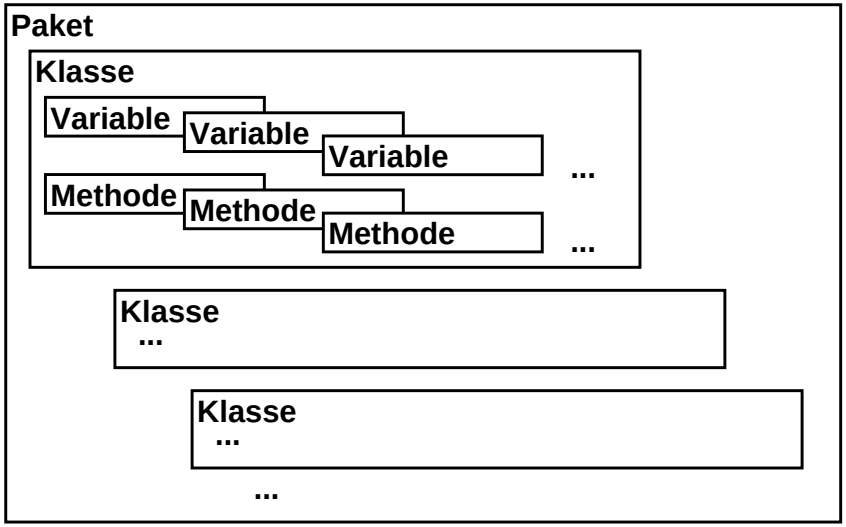
```
public class ZaehlerApplet extends MouseListenerApplet
{
   private int zaehlerstand = 0;

   public void mouseClicked(MouseEvent e)
   {
      zaehlerstand = zaehlerstand + 1;
      repaint();
   }

   public void paint(Graphics g)
   {
      g.drawRect(0, 0, getSize().width-1,
                       getSize().height-1);
      g.drawString(String.valueOf(zaehlerstand),
                       5,15);
   }
}
```

Variable

Methode

Methode

Aufbau eines Java-Programms



Klassen: Bausteine für Java-Programme

- Java-Programme bestehen aus **Klassendefinitionen**
- Üblicherweise wird jede Klassendefinition in einer eigenen **Datei** *`Klassenname`*`.java` abgelegt. Der zugehörige Bytecode liegt in der Datei *`Klassenname`*`.class`.
- Jede Klasse kann einem „**Paket**“ (engl. *package*) zugewiesen werden. Wird hierzu nichts festgelegt, ist das das „Unnamed-Package“.
- Java-Klassendefinitionen enthalten die Definitionen von Klassenbestandteilen (engl. *members*):
  - ⇨ **Variablen**, auch Datenfelder oder Merkmale genannt,
  - ⇨ **Methoden**, zur Festlegung von Programmabläufen,
  - ⇨ **Konstruktoren**, methodenähnliche Bestandteile, die beim Erzeugen einer Instanz der Klasse aktiv werden.

Variablen in Klassendefinitionen

```
public class Konto
{

  public int kontonr;

  protected int kontostand = 0;

  protected double zinssatz = 0.025;

  private String passwort = "unbekannt";

  public Person besitzer;

  public static int naechsteKontonr = 1;

}
```


Variablen

- Variablen besitzen einen **Namen**
- Variablen dienen zum Speichern von **Werten**
- Werte besitzen einen **Typ**, z.B.:
  - ⇨ int: eine ganze Zahl
  - ⇨ double: eine Gleitkommazahl
  - ⇨ eine Klasse, z.B. Person: In der Variablen darf dann ein Verweis (einen Pointer, eine Referenz) auf eine Instanz der Klasse gespeichert werden.
  - ⇨ String: ein Text (String ist eine vordefinierte Klasse)
- Variablen können bei Ihrer Definition mit einem Wert **vorbesetzt** werden, z.B. mit einem elementaren Wert wie einer Zahl oder mit einem Verweis auf ein Objekt

Instanzenvariablen

Variablen in Klassendefinitionen sind im Normalfall Instanzenvariablen, d.h. jede Instanz der Klasse erhält eine eigene Variable.

- Instanzenvariablen werden innerhalb eines Objekts (d.h. in einer Methode eines Objekts) nur mit ihrem Namen angesprochen. Beispiel:
  `kontostand = kontostand * (1 + zinssatz);`
- Ansonsten müssen Objekt **und** Variable bezeichnet werden, getrennt durch einen Punkt. Beispiel:
  `System.out.println(spendenkonto.kontonr);`

Klassenvariablen

Der Modifikator `static` wird zur Definition von Klassenvariablen verwendet. Solche Variablen gibt es für die ganze Klasse nur einmal.

- Um Klassenvariablen in Instanzen der Klasse anzusprechen, genügt der bloße Name. Beispiel:
  `naechsteKontonr = naechsteKontonr + 1;`
- Außerdem können Klassenvariablen auch von außerhalb angesprochen werden. Dann muss der Name der Klasse gefolgt von einem Punkt vorangestellt werden. Beispiel:
  `System.out.println(Konto.naechsteKontonr);`

Methoden

- Methoden haben ähnlich wie Variablen einen **Namen**
- Methoden haben einen **Ergebnistyp**
  - ⇨ Das ist der Typ des Ergebnisses, sofern ein solches nach Aufruf der Methode zurückgegeben wird.
  - ⇨ Andernfalls ist der Ergebnistyp „**void**“.
- Darüber hinaus besitzen Methoden eine **Parameterliste**
  - ⇨ Parameter werden wie lokale Variable behandelt.
  - ⇨ Die Werte der Parameter werden beim Aufruf der Methode übergeben.
- Außerdem hat jede Methode einen **Methodenrumpf**, eingeschlossen durch geschweifte Klammern {}
  - ⇨ Der Methodenrumpf enthält die **Anweisungen**, die beim Aufruf ausgeführt werden.

Ansprechbarkeit von Variablen und Methoden

- Ein Prinzip der objektorientierten Programmierung ist das **Information Hiding**, d.h. nicht alle Teile des Programms sind von überall ansprechbar.
- Klassen können mit dem Modifikator **public** für alle anderen Programmteile ansprechbar gemacht werden.
- Die Ansprechbarkeit Variablen und Methoden kann mit folgenden Modifikatoren festgelegt werden:
  - ⇨ **public**: allgemein ansprechbar
  - ⇨ **protected**: nur in Methoden der betreffenden Klasse und deren Unterklassen ansprechbar.
  - ⇨ **private**: nur in Methoden der betreffenden Klasse ansprechbar.

Einfache Datentypen in Java

`boolean`	Wahrheitswert, entweder `true` oder `false`
`char`	Zeichen (16 Bit im Unicode-Zeichensatz)
`byte`	vorzeichenbehaftete Ganzzahl (8 Bit)
`short`	vorzeichenbehaftete Ganzzahl (16 Bit)
`int`	vorzeichenbehaftete Ganzzahl (32 Bit)
`long`	vorzeichenbehaftete Ganzzahl (64 Bit)
`float`	Gleitkommazahl in 32 Bit (nach IEEE 754-1985)
`double`	Gleitkommazahl in 64 Bit (nach IEEE 754-1985)

Literale in Java

Literale werden verwendet, um in einem Programm konstante Werte eines bestimmten Datentyps verwenden zu können:

Wahrheitswerte: `true`, `false`

Ganzzahlen: `29`, `-3`

Gleitkommazahlen: `4.5`, `18.`, `1.8E-23` (bedeutet: $1{,}8 \times 10^{-23}$)

Zeichen: `'Q'` (in einfachen Apostrophen), Sonderzeichen als Escape-Sequenz, eingeleitet mit `\` (Gegenschrägstrich): `'\n'` (Zeilenvorschub), `'\t'` (Tabulator), `'\''` (Apostroph), `'\"'` (Anführungszeichen), `'\\'` (Gegenschrägstrich), `'\u`*zzzz*`'` (ein durch die Ziffern *zzzz* angegebener Unicode)

Zeichenketten (Strings): `"Programmieren"`, beliebige Zeichen in Anführungszeichen, Sonderzeichen als Escape-Sequenz, z.B.: `"C:\\programme\\netscape\n"`

Namen in Java

- Untrennbar mit einem Namen verbunden sind in Java nur Pakete, Klassen, Variablen und Methoden.
- Objekte haben nach ihrer Erzeugung keinen Namen (ebenso wenig wie Zahlen oder Zeichen).
- Um ein Objekt zu benennen, benötigt man eine **Variable**, der man eine Referenz auf das Objekt als Wert zuweist.
- Je nach Art der Variablen (Instanzenvariable, Klassenvariable, lokale Variable) kann dann das Objekt in bestimmten Kontexten über den Namen der Variablen angesprochen werden.

Definition einer Variablen

Zuweisung einer Objektreferenz

```
Person kanzler;
kanzler = new Person("Gerhard", "Schröder");
System.out.println(kanzler.vorname);
```


Definition von Variablen in Java

Variablen können an verschiedenen Stellen in einem Java-Programm definiert werden:

- Auf oberster Ebene einer Klassendefinition
  - ⇨ als Instanzenvariable (z.B. nr)
  - ⇨ oder als Klassenvariable (z.B. naechsteNr)
- In Methoden (u.ä., insbesondere auch in Konstruktoren)
  - ⇨ als Parameter der Methode (z.B. args)
  - ⇨ oder als lokale Variable (z.B. neuesKonto)

```
public class Konto {
    public int nr;
    static public int naechsteNr;
    public static void main(String[] args) {
        Konto neuesKonto = new Konto();
        ... } }
```


Anweisungen in Java

Der Rumpf von Methoden und Konstruktoren besteht aus Blöcken (s.u.), d.h. Sequenzen sogenannter Anweisungen.

Elementare Anweisungen sind:

i=i+1 **Zuweisungen**, z.B.: `i = i + 1`

meth() **Methodenaufrufe**, z.B.: `System.out.println(i)`

new **Objekterzeugungsausdrücke**, z.B.: `new Person()`

Zusammengesetzte Anweisungen sind:

Blöcke (Anweisungssequenzen), bestehen aus Anweisungen, verkettet mit „`;`“, geklammert durch „`{}`“ z.B.: `{i = i+1; s = s+i; println(s)}`

Bedingte Anweisungen, z.B.: `if`-Anweisung
`if (x>0) {y=x} else {y=-x}`

Schleifen, z.B.: `while`- und `for`-Anweisung
`while(i<10){summe = summe+i; i=i+1}`

Bedingte Anweisungen und Programmschleifen in Java

Bedingte Anweisung:

```
if (Bedingung)
    {Anweisung; Anweisung; ...}
else
    {Anweisung; Anweisung; ...}
```

Schleife

```
while (Bedingung)
    {Anweisung; Anweisung; ...}
```

Alternative Art der Schleife:

```
for (Anfangsanweisung; Bedingung; Wiederholanweisung)
    {Anweisung; Anweisung; ...}
```

Ausdrücke in Java

Variablen, **Literale** und **Ergebnisse** von Methodenaufrufen können mit Hilfe von **Operatoren** zu **Ausdrücken** verknüpft werden.

- Arithmetische Operatoren: `+`, `-`, `/`, `*` (Grundrechenarten), z.B.: `y = x*x/4 - x + 1;`
- Vergleichsoperatoren: `==` (gleich), `!=` (ungleich), `<` (kleiner), `>` (größer), `<=` (kleiner oder gleich), `>=` (größer oder gleich), z.B.: `if x != 0 System.out.println(1/x);`
  - ⇨ Achtung: Strings müssen mit speziellen Methoden verglichen werden, z.B. Gleichheit mit `equals()`: `if (eingabe.equals("ja")) {doIt()}`
- logische Operatoren: `&` (und), `|` (oder), `!` (nicht), z.B.: `while (i<n & x>0) {x = x-y; i = i+1}`

Programmablauf in Java

- Java-Anwendungen sind repräsentiert durch Java-Klassen.
- Die Ausführung einer Java-Anwendung beginnt mit der Klassenmethode `main()` der betreffenden Klasse. Aufrufparameter werden in Form des Parameters `args`, eines Arrays von Zeichenketten (Strings) übergeben.
- Die Anweisungen in der Methode `main()` werden der Reihenfolge nach durchlaufen.
- Trifft der Interpreter dabei auf einen Methodenaufruf, so werden die Anweisungen dieser Methode abgearbeitet. Danach wird an der Stelle nach dem Methodenaufruf fortgesetzt.
- Mit der letzten Anweisung der `main()`-Methode endet das Programm.

Programmablauf in Java: Beispiel

```
public class Copier {
    public OutputStream out;
    public InputStream in;
    public void start() throws IOException {
    // Hier wird vom Stream in zum Stream out kopiert
    [... Programmcode weggelassen] }
Start → public static void main(String[] args)
                          throws IOException {
        Copier keyboardEcho = new Copier();
        keyboardEcho.in = System.in;
        keyboardEcho.out = System.out;
        keyboardEcho.start();
    }
}
```

Ende