*Grundlagen – Pakete*

* **Grundlagen S.2**
  1. objektorientierte Programmierung(OOP)
  2. Begriffe
* **Einführung S.3**
  1. Java
  2. Programmentwicklung
* **Erstes Beispiel S.3**
* **Sprachelemente S.5**
  1. Lexikalische Elemente
  2. Datentypen
  3. Literale
  4. Variablen und Konstanten
  5. Arrays
  6. Operatoren
  7. Kontrollstrukturen
* **Klassen und Objekte S.11**
  1. alles rund um Objekte(Return, final, ...)
* **Unified Modeling Language (UML) S.14**
  1. Klassendiagramme
  2. Relation
* **Klassen S.16**
  1. Attribute
  2. Methoden
* **Ableitung & Vererbung S.16**
* **Pakete S.18**
  1. Erstellung von Paketen
  2. Sichtbarkeit
  3. Geltungsbereiche
  4. Zugriffsrechte
* **Einlesen S.17**
* **Zufallsgenerator S.18**
* **Sortierverfahren S.19**
  1. Selectionsort
  2. Insertionsort
* **Interfaces & abstrakte Klassen S.20**
* **Exceptions S.21**

*Grundlagen*

Die Grundidee der **objektorientierten Programmierung** ist, Daten und Funktionen, die auf diese Daten angewandt werden können, möglichst eng in einem sogenannten Objekt zusammenzufassen und nach außen hin zu kapseln, so dass Methoden fremder Objekte diese Daten nicht versehentlich manipulieren können.

→ Datenkapselung

In der objektorientierten Programmierung gibt es zwei Designs:

1. Top - Down
2. Bottom – Up

**Top – Down:**

Formulierung eines Überblicks über das System, Details werden vernachlässigt. Stattdessen wird der Algorithmus in Abschnitte unterteilt. Diese Abschnitte werden im Folgenden genauer ausformuliert, die zunächst aus genauer definierten, kleineren Abschnitten besteht, bis schließlich die komplette, detaillierte Spezifikation des Algorithmus erreicht ist. Schwerpunkt auf Planung und Verständnis des Systems.

**Bottom – Up:**

Es werden zunächst einzelne Programmbestandteile definiert und eventuell sogar schon gleich codiert. Daraus wird dann das Programm zusammengestöpselt. Dabei sollte darauf geachtet werden, dass ein Sourcecode entsteht, den man jederzeit in einem anderen Projekt verwenden kann.

In einer **Klasse** stehen die Attribute und Methoden (auch Memberfunktion genannt) eines Objekts. Die Klasse ist der Bauplan des zur Laufzeit erstellten Objekts.

Durch die **Datenkapselung** wird der direkte Zugriff auf die innere Datenstruktur unterbunden. Objekte können den internen Zustand anderer Objekte (z.B Wert von einem Attribut) nicht in unerwarteter Weise lesen oder ändern. Ein Objekt hat eine Schnittstelle, die darüber bestimmt, auf welche Weise mit dem Objekt interagiert werden kann.

Bei der **Polymorphie** können verschiedene Objekte (auch Exemplare genannt) auf die gleiche Nachricht unterschiedlich reagieren.

**Vererbung** heißt vereinfacht, dass eine abgeleitete Klasse die Methoden und Attribute (auch Member genannt) der Basisklasse ebenfalls besitzt, also „erbt“. Somit kann die abgeleitete Klasse auch darauf zugreifen.

*Einführung*

Die Java Plattform (Java Platform) besteht aus einer großen Menge vorgefertigter Klassen (Klassenbibliothek). Diese Klassen sind in sogennanten Packeten(Packages) organisiert.

Diese Pakete decken die unterschiedlichsten Bereiche ab, z.B.: Strings oder Ein-/Ausgabe.

Die Java Plattform ist in das so genannte Java **Development Kit (JDK)** integriert. Dieses JDK wird von SUN kostenlos zur Verfügung gestellt und umfasst folgende Komponenten:

Compiler (javac)

komplette Java Plattform

Virtuelle MaschineVM)

*Ein erstes Beispiel*

gibt es auf der nächsten Seite...

**class** Person // Klassenkopf

{ // Klassenrumpf

**private** String vorname; // Attribute sind durch private gekapselt und können

**private** String nachname; //nur durch Methoden in der selben Klasse geändert //werden.

**public** Person(String derVorname, String derNachname) // Konstruktor

{

vorname = derVorname;

nachname = derNachname;

}

// Methoden

**public** **void** aendereNachname(String neuerNachname)

{

//ändert das Attribut Nachname anhand des übergebenen neuen Werts.

nachname = neuerNachname;

}

**public** String gibName()

{

//Die beiden Strings Vor-/Nachname werden zusammengefügt und zurückgegeben

//um anschließend ausgegeben werden zu können.

**return** vorname + " " + nachname;

}

}

**public** **class** Main {

**public** **static** **void** main(String[] args) //ausführbare Methode

{

Person p1 = **new** Person("Andrea", "Gruber");

//ein neues Objekt der Klasse Person wird angelegt.

//Die Attribute des Objekt p1 müssen dem Konstruktor entsprechen //(2 Strings)

System.***out***.println(p1.gibName());

//Ausgabe des Objekts p1 durch die Methode gibName()

Person p2 = **new** Person("Richard", "Frech");

System.***out***.println(p2.gibName());

p1.aendereNachname("Frech");

//Für das Objekt p1 wird die Methode aendereNachname, die den Wert des //Attributes ändert, aufgerufen.

System.***out***.println(p1.gibName());

}

}

*Sprachelemente*

Foldende **lexikalische Elemente** beinhaltet Java:

White Spaces (Leerzeichen, horiz. Tab,...)

Kommentare

Bezeichner

Schlüsselwörter (int, char, class, public,...)

Literale

Interpunktionszeichen ( { } [ ] ( ) ; , .)

Operatoren (+ - \* / % < > usw.)

Kommentare gliedern sich in drei Gruppen:

Traditionelle Kommentar  /\* Kommentar\*/

Zeilenkommentar // Kommentar

Dokumentationskommentar /\*\* Schnittstelle X \*/

Unter Bezeichner versteht man die Namen in einem Programm. Mit einem Namen versehen sind : Variablen, Klassen, Methoden, Interfaces, Pakete. Sie bestehen aus einer beliebig langen Folge von Unicode-Buchstaben und -Ziffern.

Elementare **Datentypen** sind:

Boolescher Typ

Ganzzahlige Typen

Gleitpunkt Typen

Referenz **Datentypen** sind:

Felder

Klassen

**Literale**

Für ganze Zahlen **(Int)** gibt es drei verschiedene Darstellungen:

dezimal: 31

hexadezimal: 0x1f

oktal: 037

Gleitkommazahlen **(float)** können ebenfalls auf drei verschiedene Arten dargestellt werden:

23.0149

230149e-4

230149E2

Einzelne Zeichen **(char)** werden unter Hochkomma gesetzt. Dabei handelt es sich um Unicode Zeichen. (2 Byte Speicherbdarf).

' + ', ' \* '

' a ', ' ö '

' 1 '

' " , ' \' '

' \u0061 '

Zeichenketten **(String)** bestehen aus einer Folge von Unicode-Zeichen. Diese Folge kann natürlich auch leer sein. Zeichenketten werden unter Anführungszeichen gesetzt.

"Einführung in Java"

"Einführung in \"Java\""

"" //Leerstring

Wobei bei Zeichen und Zeichenketten das \ -Zeichen das Escapesymbol darstellt, um ein '- Zeichen oder ein "- Zeichen darstellen zu können.

**Variablen** haben immer einen Namen und einen veränderbaren Wert. Es gibt Referenz-Variablen und Elementare Variable gemäß des Typen (siehe Datentypen).

Konstaten sind "Variablen", deren Wert nicht verändert werden kann. Bei der Deklaration wird das Schlüsselwort final vorangestellt.

final double PI = 3.14159;

## Arrays (Felder)

Arrays sind Objekte, die aus einer bestimmten Anzahl von Variablen des selben Typs bestehen, den sogenannten Komponenten des Arrays. Die Anzahl der Komponenten ist unveränderlich. Arrays werden zur Laufzeit erzeugt. Weder Arrays noch ihre Komponenten haben einen Namen. Ist n die Anzahl der Komponenten eines Arrays, dann sind die Komponenten eines Arrays mit 0,1,...,n-1 durchnummeriert. Die Nummer einer Komponente nennt man Index. Der Zugriff auf ein Array erfolgt über eine Referenz. Der Zugriff auf die Komponenten eines Arrays erfolgt über ihren Index.

Felder werden dynamisch erzeugt. Felder kann man auf drei verschiedene Arten erzeugen. Hier werden die gelieferten Referenzen einer Variablen zugewiesen.

int [] feld1 = {7,3,77,5,10,6,55};

int [] feld2 = new int [12];

int [] feld3 = new int[] {88,66,2,99,8};

Die Anzahl n der Komponenten eines Feldes nennt man die Länge des Feldes. Die Länge eines Feldes ist unveränderlich. Jedes Feld hat ein Attribut mit dem Namen length, in dem die Länge des Feldes abgelegt ist.

Bsp.:

int[] feld = {7,3,77,4};

int laengeDesFeldes = feld.length; //In diesem Fall: 4

## Mehrdimensionalen Arrays:

Bsp.:

double[ ][ ] a1 = new double [ ] [ ] {{1.0,-4.9},{3.7,2.0}};

## Operatoren

Bei Operatoren unterscheidet man unäre, binäre und ternäre Operatoren, je nach Anzahl der Operanden.

* Arithmetische Operatoren
* Additive Operatoren:
  + Addition: +
  + Subtraktion: -
* Multiplikative Operatoren:
  + Multiplikation: \*
  + Division: /
  + Divisionsrest: %

## Vergleichs Operatoren

* Relationale Operatoren:
  + kleiner als: <
  + größer als: >
  + kleiner oder gleich: <=
  + größer oder gleich: >=
* Gleichheits-Operatoren:
  + gleich: ==
  + ungleich: !=

Die relationalen Operatoren haben die höhere Priorität. Ein Ausdruck mit Vergleich Operatoren liefert immer einen boolean Wert.

## Logische Operatoren

UND: &

EXKLUSIVE ODER: ^

INKLUSIVE ODER: |

& hat höhere Priorität als ^, und ^höhere als |. Die Operanden müssen beide boolean sein. Es werden beide Operanden ausgewertet und dann die Resultate verknüpft. Das Ergebnis ist bein boolean Wert.

## Boolesche Operatoren

UND: &&

ODER: ||

&& hat höhere Priorität als ||. && arbeitet wie &, wertet aber seinen rechten Operanden nur aus, wenn der linke Operand den Wert true hat. || arbeitet wie |, wertet seinen rechten Operanden nur aus, wenn der linke Operand den Wert false hat. Ein Ausdruck mit Booleschen Operatoren liefert immer einen boolean Wert.

## Logische Negation

NICHT: !

Nicht hat eine rechten Operanden, der vom Typ boolean sein muss.

Das Resultat ist false, wenn der Operand den Wert true hat und umgekehrt.

## Konditional Operator

Bedingung: ? :

Erster Operand links, zweiter Operand dazwischen, dritter Operand rechts. Dieser Operand stellt eine Kurzform für ein if-else dar.

Bsp.:

char ch = 'j';

system.out.println(ch == 'j' ? "ja" : "Nein");

## Bit Operatoren

Die Bit Operatoren sind &,^und |. Es dind also dieselben Operatoren wie die logischen Operatoren. Bei den Bit Operatoren müssen beide Operanden ganzzahlig sein. Es erfolgt eine bitweise Verknüpfung der Operanden. Das Resultat hat den Typ der Operanden.

Bsp.:

0x1010 & 0x1001 ergibt 0x1000

0x1010 ^ 0x1001 ergibt 0x0011

0x1010 | 0x1001 ergibt 0x1011

## Zuweisungs-Operatoren

Zuweisung: =

Bsp.: stand = stand + betrag;

Zuweisungs Operator kombiniert mit arithmetischen Operatoren:

i +=5; i=i+5;

j-=3; j=j-3;

f\*=PI; f=f\*PI;

d/=100;d=d/100;

## Inkrement- und Dekrement Operatoren

i++; i=i+1;

++i;i=i+1;

i--;i=i-1;

--i;i=i-1;

Bei i++ und i-- ist der Wer des Ausdrucks gleich dem Wert des Operanden vor dessen Modifikation. Bei ++i und --i ist der Wert des Ausdrucks gleich dem Wert des Operanden nach dessen Modifikation.

## Kontrollkonstrukte

## Verzweigungen

if und if-else und switch-case-default wie in C.

Der switch Ausdruck muss eine der Typen char, byte, short oder int haben.

## Schleifen

while, do-while und for wie in C.

Mit der Anwesiung continue; kann ein Schleifendurchgang abgebrochen werden. Mit der Anweisung break; kann eine Schleife ganz abgebrochen werden.

*Klassen & Objekte*

**Objekte** werden nicht einzeln entworfen und implementiert. Objekte mit dem gleichen Satz von Attributen und Methoden werden exemplarisch in einer so genannten Klasse beschrieben. In einer **Klasse** stehen die Attribute und Methoden (auch Memberfunktion genannt) eines Objekts. Die Klasse ist der Bauplan des zur Laufzeit erstellten Objekts.

Eine Klasse beschreibt "exemplarisch" gewisse Objekte durch Angabe von:

* Attributen (Daten),
* Konstruktoren, die bei der Erzeugung eines Objektes zur Ausführung kommen,
* Methoden (Operationen), die auf die Objekte angewendet werden können.

Ein Objekt ist ein Exemplar einer Klasse, dessen Zustand durch Werte für die Attribute bestimmt ist.

In einem Programm können Objekte erst dann erzeugt werden, wenn die entsprechenden Klassen verfügbar sind. Aus einer Klasse entstehen Objekte durch "Instanzierung", deshalb nennt man ein Objekt eine Instanz seiner Klasse.

## Methoden

Methoden sind Operationen. Methoden werden auf Objekte der Klasse angewendet, für die sie definiert sind. Methoden bestimmen das Verhalten der Objekte. Objekte derselben Klasse zeigen gleiches Verhalten. Methoden sind Funktionen.

## Konstruktoren...

haben als Namen den Namen der Klasse,

haben keinen Rückgabetyp,

werden nur bei der Konstruktion eines Objekts aufgerufen, und das automatisch.

Die **return - Anweisung** beendet die Ausführung einer Methode bzw. eines Konstruktors.

## Erzeugung von Objekten

Objekte werden zur Laufzeit mit dem Operator new erzeugt. Objekte werden im so genannten Freispeicher (heap) angelegt und haben keinen Namen. Objekte können nur indirekt über eine Referenz angesprochen werden.

## Referenzen

Die Referenz erhält man bei der Erzeugung des Objekts. Eine Referenz hat einen Wert und einen Typ. Der Wert ist die Speicheradresse des Objekts und der Typ ist der Datentyp des Objekts. Eine Variable vom Typ einer Klasse nennt man eine Referenz - Variable. Um mit einer Referenz zu arbeiten, legt man sie in einer Referenz - Variablen ab. In einer Referenz - Variablen können nur Referenzen desselben Typs abgelegt werden. Die Null - Referenz null ist Typ – Neutral.

## Der Zugriff auf die Elemente eines Objektes

Der Zugriff auf ein Element eines Objekts erfolgt mit dem Zugriffs – Operator (Punkt-Operator '.'). Der Zugriffs-Operator hat als linken Operanden die Referenz des Objekts und als rechten Operanden ein Element des Objekts (Attribut, Methode).

## Datenkapselung, Geheimnisprinzip und Schnittstelle

Auf die Attribute (Daten) eines Objektes soll kein direkter Zugriff gewährt werden. Die Attribute sollen nur über entsprechende Methoden des Objektes erreicht werden können (Datenkapselung). Auch die Datendarstellung soll nicht offen gelegt werden (Geheimnisprinzip). Die Gesamtheit der Methoden, die zugänglich zur Verfügung gestellt werden, nennt man Schnittstelle.

## Sichtbarkeit

Um ein Klassenelement von außerhalb der Klasse zugreifbar zu machen, muss es mit dem Modifizierer „public“ versehen werden. Wenn man ein Klassenelement vor dem Zugriff von außerhalb schützen (verbergen) will, versieht man es mit dem Modifizierer „ private“.

## Das aktuelle Objekt

Methoden werden immer für ein Objekt aufgerufen. Das Objekt, für das eine Methode aufgerufen wird, nennt man das aktuelle Objekt. Eine Methode operiert auf dem aktuellen Objekt. Innerhalb einer Methode erfolgt der Zugriff auf die Attribute des aktuellen Objekts durch die Nennung des Namens der Attribute. Der Aufruf einer anderen Methode für das aktuelle Objekt erfolgt auch durch die Nennung des Namens der Methode.

## Die this Referenz

Innerhalb einer Methode gibt es implizit die Referenz auf das aktuelle Objekt. Die Referenz auf das aktuelle Objekt erhält man mit dem Schlüsselwort „this“. Man kann „this“ beim Zugriff auf die Attribute des aktuellen Objekts und beim Aufruf von Methoden für das aktuelle Objekt verwenden.

## Initialisierer bei Attributen

Die Deklaration eines Attributs kann einen Initialisierer enthalten.

Bsp.: private double zinssatz = 1.5;

## final Elemente sind unveränderliche Attribute.

**Überladene Methoden**

Man kann für eine Klasse mehrere Methoden mit demselben Namen deklarieren. Diese Methoden müssen sich durch die Parameterliste unterscheiden. Ein unterschiedlicher Rückgabetyp unterscheidet Methoden nicht.

Bsp.: public double zahleAus(double betrag);

public double zahleAus();

## Die Konstruktion von Objekten

Bei der Konstruktion eines Objekts erfolgt der Reihe nach:

* Reservierung des benötigten Speicherplatzes,
* Belegung der Attribute mit Standard- bzw. den Initialisierungswerten,
* Aufruf des Konstruktors.

Hat eine Klasse mehrere Konstruktoren, dann entscheiden die spezifizierten Parameter welcher Konstruktor aufgerufen wird.

Den Aufruf des **Garbage – Collector**, der Objekte zerstört, kann man auch mit der Methode gc() erzwingen.

*Unified Modeling Language (UML)*

Die UML dient zur Beschreibung von objektorientierten Strukturen. Die UML bietet die besten Analyse und Entwurfsmethoden der Software Entwicklung zusammenfasst.

Man unterscheidet folgende Diagramme:

* Statische Klassen- und Objektdiagramme
* Interaktionsdiagramme
* Aktivitätsdiagramme
* Anwendungsfalldiagramme (Use case)

**Sichtbarkeitszeichen**

+ oder nichts: öffentliche (public) Elemente

- : private (private) Elemente

#: geschützte (protected) Elemente

**Relationen zwischen Klassen**

Vererbung

Assoziation

Aggregation und Komposition

**Kardinalität**

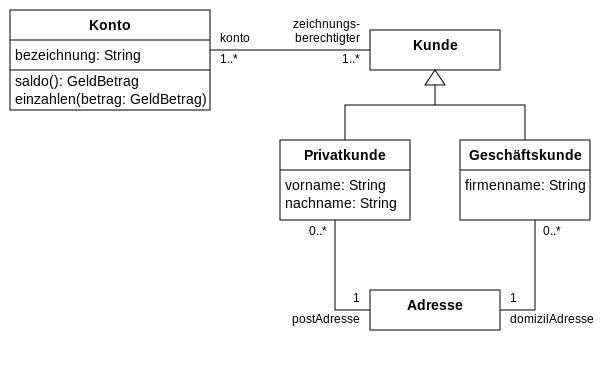
1 genau ein Objekt

\* viele, d.h. Null oder mehr

0..1 höchstens 1

m..n mindestens m aber höchstens n

**Klassendiagramm:**



*Klassenattribute/-Methode*

In einer Klasse kann man auch Attribute und Methoden definieren, die sich nicht auf die Instanzen der Klasse beziehen, sondern auf die Klasse selbst. Mit **„static“**.

Beispiel:

Die Klasse (z.B.: Konto) definiert das Attribut (z.B.: Zinssatz) zentral für alle Objekte der Klasse, d.h. das Attribut ist kein Instanz-Attribut sondern ein Klassen-Attribut.

In einer Instanz-Methode hat man **Zugriff** auf

* die Attribute und Methoden des aktuellen Objekt,
* die Attribute und Methoden von einem anderen Objekt des Typs der Klasse,
* die Klassen-Attribute und Klassen-Methoden.

In einer Klassen-Methode hat man **Zugriff** auf

* die Klassen-Attribute und Klassen-Methoden,
* die Attribute und Methoden von einem Objekt des Typs der Klasse, wenn die Methode eine Referenz auf das Objekt besitzt.

In der UML werden Klassenattribute & - Methoden mit Unterstrich gekennzeichnet.

*Vererbung*

Ziel beim Ableiten und Vererben ist es Eigenschaften einer bereits vorhandenen Klasse zu nutzen. Eine Klasse wird von einer bestehenden Klasse abgeleitet. Die neue Klasse ist Spezialisierung der alten Klasse. Programmcode-Wiederverwendung.

## Subklasse - Superklasse

Eine Klasse **Geschäftskunde** wird zur Subklasse der Klasse **Kunde**, wenn man in der Deklaration der Klasse **Geschäftskunde** die Klasse **Kunde** mit dem Schlüsselwort „extends“ angibt. Die Klasse Kunde ist dann Superklasse der Klasse Geschäftskunde. → class Geschäftskunde extends Kunde.

Class Viereck

Class Rechteck extends Viereck

Class Quadrat extends Rechteck

* Viereck ist eine direkte Superklasse von Rechteck.
* Viereck ist eine indirekte Superklasse von Quadrat.
* Rechteck ist eine direkte Subklasse von Viereck.
* Quadrat ist eine indirekte Subklasse von Viereck.
* Quadrat erbt von Viereck indirekt durch Rechteck.

Eine Subklasse erbt Schnittstellen. Zugriff nur auf „Public“ & „Protected“ Attribute.

## Möglichkeiten durch Vererbung

Eine abgeleitete Klasse kann man dadurch:

**Erweitern:** Neue Elemente hinzufügen

**Überschreiben:** Geerbte Elemente verdecken, indem man sie durch eigene ersetzt.

**Definieren:** Eine Vorgabe realisieren.

## Zugriff auf verdeckte Elemente der Superklasse

Auf verdeckte Instanz - Attribute und Instanz - Methoden greift man mittels

„super.element“

zu, wobei „element“ für das Attribut bzw. die Methode steht.

Auf verdeckte Klassen-Attribute und Klassen-Methoden greift man mittels

„name.element“

zu, wobei „name“ für den Klassennamen der Superklasse steht.

*Pakete*

Klassen werden in Pakete organisiert. In einem Paket werden solche Klassen zusammengefasst, die eng zusammenarbeiten oder ein gemeinsames Thema haben. Mit einem Paket entsteht ein neuer Geltungsbereich für die in dem Paket enthaltenen Klassen. Innerhalb eines Pakets besteht Eindeutigkeit der Klassennamen.

Jedes Paket hat einen Namen. Dieser besteht aus einer Folge von Bezeichnern, die durch einen Punkt miteinander verbunden sind.

Bsp.:

package sew.java.a;

Innerhalb eines Pakets besteht **"Sicht"**. Wenn eine Klasse des Pakets eine andere Klasse des Pakets benützen will, so kann sie das ohne weitere Vorkehrungen tun. Wenn eine Klasse außerhalb eines Pakets eine Klasse des Pakets benützen will, muss die Klasse innerhalb des Pakets public sein. Außerdem muss man das Paket, welche die public Klasse enthält importieren, indem entweder das ganze Paket importiert oder nur die gewünschte Klasse.

Bsp.:

import sew.java.a.\*; // Alle Klassen werden importiert

import sew.java.a.Uebung; // Nur die eine Klasse wird importiert

Zwischen Paketnamen und dem Pfad der .java und .class Dateien besteht ein unmittelbarer **Zusammenhang.**

* Elemente der Klasse sind nur zugreifbar wenn die Klasse zugreifbar ist.
* Public Elemente sind für jeden Code zugreifbar für den auch die Klasse zugreifbar ist.
* private Elemente sind nur innerhalb der Klassendeklaration zugreifbar.
* Protected Elemente sind innerhalb des Paketes, dem die Klasse angehört und in Subklassen zugreifbar.
* Notation in der UML per Doppelpunkt.

*Einlesen*

Man liest die Eingabe des Benutzers als String ein. Was, wenn er eine Zahl eingeben soll? Dann muss der String hinterher in eine Zahl, meistens int, manchmal double, umgewandelt werden. Das geht mit **Integer.parseInt(stringVariable)** bzw. **Double.parseDouble(stringVariable)**. Integer und Double sind Klassen im Package java.lang und müssen deshalb nicht importiert werden. Für die anderen primitiven Zahlentypen gibt es dort auch diese sogenannten Wrapperklassen und die entsprechenden Parse-Methoden (z.B. Short.parseShort etc.)

import java.io.\*;

class AlterEingabe

{

public static void main(String[] args)

{

InputStreamReader isr = new InputStreamReader(System.in);

BufferedReader br = new BufferedReader(isr);

System.out.print("Wie alt bist du: ");

String eingabe = br.readLine();

int alter = Integer.parseInt(eingabe);

if (alter >= 18) { System.out.println("Du darfst dieses Spiel spielen."); } else { System.out.println("Kein Spiel für dich."); }

}

}

Auch die Klasse Scanner kann man zum Einlesen benutzen. Zahlen einlesen mit dem Scanner ist deutlich einfacher - man benutzt einfach nextInt() statt next(), welches man für das Einlesen von Text benutzt:

import java.util.\*;

class AlterEingabe2

{

public static void main(String[] args)

{

Scanner sc = new Scanner(System.in);

System.out.print("Wie alt bist du: ");

int alter = sc.nextInt();

if (alter >= 18)

{System.out.println("Du darfst diese Spiel spielen."); }

else

{ System.out.println("Kein Spiel für dich."); }

}

}

*Zufallszahlen*

Eine Methode Zufallszahlen zwischen z.B.: 1 und 6 würfeln und diese in einem Array speichern. Benutze dazu Math.random (z.B. wuerfe[i]=(int)(Math.random()\*6)+1;)

Für 10 – 20: wuerfe[i] = (int)(Math.random()\*11)+10;

*Sortierverfahren*

**Selectionsort**

**public** **static** **int**[] Sort(**int**[] ungeordnet)

{

**int** j,k,l,temp;

l=ungeordnet.length;

**for**(j=0;j<l-1;j++)

{

**for**(k=j+1;k<l;k++)

{

**if**(ungeordnet[k] < ungeordnet[j])

{

temp = ungeordnet[j];

ungeordnet[j] = ungeordnet[k];

ungeordnet[k] = temp;

}

}

}

**return** ungeordnet;

}

**Insertionsort**

**public** **static** **int**[] Sort(**int**[] Array)

{

**int** temp,i,j=0;

**for**(i=1;i<Array.length;i++)

{

temp=Array[i];

j=i;

**while**(j>0 && Array[j-1]>temp)

{

Array[j]=Array[j-1];

j--;

}

Array[j]=temp;

}

**return** Array;

}

**Selectionsort mit Strings**

**int** i,j=0;

String temp = "";

**for**(i=0;i<Array.length-1;i++)

{

**for**(j=i+1;j<Array.length;j++)

{

**if**(Array[i].compareTo(Array[j]) > 0)

{

temp = Array[i];

Array[i]=Array[j];

Array[j]=temp;

}

}

}

*Interfaces & abstrakte Klassen*

Eine abstrakte Klasse hat den Zweck, als Superklasse eine grundlegende Funktionalität für Subklassen anzubieten.

Zusätzlich kann eine abstrakte Klasse Methoden deklarieren, die keine Implementierung haben. Solche Methoden nennt man abstrakte Methoden. Subklassen einer abstrakten Klasse bieten eine konkrete Implementierung der in der Superklasse abstrakt deklarierten Methoden an.

Eine Klasse wird als abstrakte Klasse deklariert, indem man ihr den Modifizierer abstract gibt.

Von einer abstrakten Klasse können keine Objekte erzeugt werden.

Ein Interface ist mit einer abstrakten Klasse vergleichbar, die ausschließlich public und abstract Methoden enthält.

*Exceptions*

Eine Ausnahme (Exception) ist ein Fehler oder ein nicht geplantes

Ereignis, das während der Ausführung eines Programms

vorkommt und dessen normalen Ablauf stört.

Eine Exception in Java ist ein Signal während der Programmausführung, das anzeigt, dass eine Ausnahmesituation eingetreten ist.

Beispiele für Ausnahmesituationen:

Es wird eine Referenz mit dem Wert null benutzt.  
Es wird bei einem Array ein Index benutzt, der jenseits der Arraygrenzen liegt.  
Ein Objekt ist nicht im Scope.  
Eine Datei wird nicht gefunden.  
Ein Netzwerk-Link ist fehlgeschlagen.

**throw-and-catch**

Java liefert einen Mechanismus für einen systematischen Umgang mit Ausnahmesituationen. Der Mechanismus folgt dem throw-and-catch Paradigma:

**"throw an exception"**: Es wird eine Ausnahmesituation signalisiert. Das geschieht an der Stelle, an der die Ausnahmesituation festgestellt wird

**"catch an exception":** Die Ausnahmesituation wird in geeigneter Form behandelt. Das geschieht an einer bestimmten anderen Stelle.

Die try-Klausel enthält die Anweisungen die man möglichst ohne eine Exception durchbringen will.

In den catch-Blöcken befinden sich die Exception-Handler. Jeder catch-Block steht für einen eigenen Exception Typ.