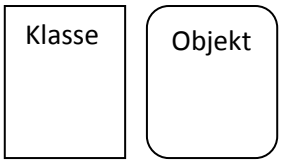
Objektorientierte Programmierung:

1) Sie können den Unterschied zwischen einer Klasse und einem Objekt erklären

- Klassen: Sind Baupläne für «Objekte» → Bauplan für einen Wagen
  - wird eckig dargestellt
- Objekte: Sind Instanzen der Klasse → der rote Wagen
  - wird mit runden «Ecken» dargestellt
- Von einer Klasse können mehrere Instanzen erzeugt werden



2) Sie kennen die Grundkonzepte von Abstraktion und Modularisierung

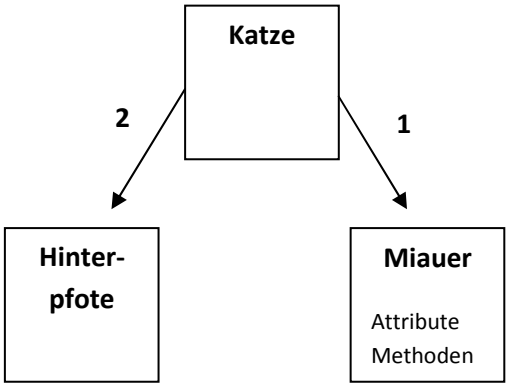
- Abstraktion bezeichnet einen Vorgang, bei dem die Sicht von der realen Welt oder einem Modell in ein (vereinfachtes) Modell gefasst wird, welches die wesentlichen Bestandteile hervorhebt und / oder sich darauf beschränkt.
  - sich auf das wesentliche konzentrieren
  - unwichtige Details vernachlässigen
  - Abstraktion ist abhängig von der Perspektive des Betrachters
- Modularisierung bezeichnet die Zerlegung einer Gesamtaufgabe in Teilaufgaben und die Definition der erforderlichen Schnittstellen, so dass die entstehenden Module möglichst unabhängig voneinander bearbeitet werden können.
  - Zerlegen und Strukturieren eines Gesamtsystems
  - Reduktion der Systemkomplexität
  - Teilsysteme erfüllen abgegrenzte Systemfunktionen
  - **Beispiel: Abstrahieren Sie die wesentlichen Funktionen einer Stereoanlage** (was).

    → Abspielen von CD's
    → Radio wiedergabe
    → Lautstärke regeln
    → MP3 abspielen

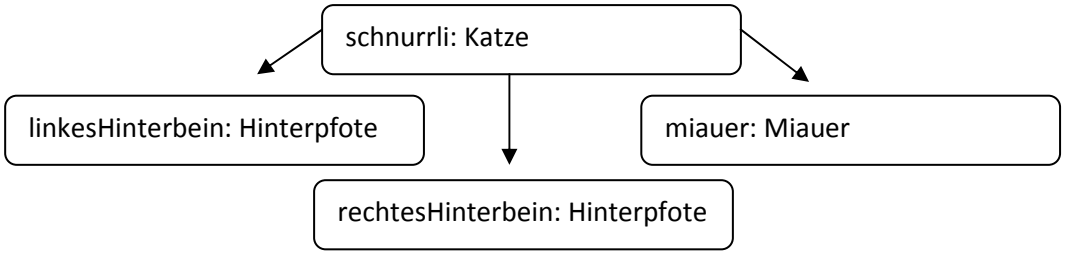
    Modularisieren Sie anschliessend die Stereoanlage für die wesentlichen Funktionen.
    → CD Laufwerk
    → Boxen
    → MP3

3) Sie können Klassen- und Objektdiagramme interpretieren

- Klassendiagramme:
  - zeigen Klassen und deren Beziehung untereinander
  - zeigen die statische Sicht eines Programms
  - halten Informationen über den Source-Code fest



- Objekte
  - zeigen Objekte zu einem bestimmten Zeitpunkt im Programmablauf
  - zeigen die dynamische Sicht eines Programms zur Laufzeit
  - stellen Beziehungen zwischen den Objekten dar (welches Objekt referenziert / benutzt andere Objekte)



4) Sie können Beziehungen zwischen Klassen erkennen und gemäss Lehrbuch (BlueJ) darstellen.

- siehe Punkt 3

5) Sie kennen das Konzept des Information Hiding und wissen, wie es durch Zugriffsmodifizierer unterstützt wird

- Als Datenkapselung (information hiding) bezeichnet man das Verbergen von Daten oder Informationen vor dem Zugriff von aussen. Der direkte Zugriff auf die interne Datenstruktur wird unterbunden und erfolgt statt dessen über definierte Schnittstellen.
- Idee: Eine Programmiererin / ein Programmierer muss nur wissen, welche Methoden eine Klasse anbietet (die «Interface»-Dokumentation). Sie / Er

braucht die Implementationsdetails nicht zu kennen (vgl. Modularisierung und Abstraktion).
→ no need to know

- Ein weiterer wichtiger Vorteil von Information Hiding ist eine lose Kopplung zwischen der selber programmierten und der verwendeten Klasse.
  → not being allowed to know
- begünstigt starke Kohäsion und lose Kopplung
- Ein wichtiges Mittel für das Information Hiding ist der Zugriffsmodifizierer private.
- Attribute sollten immer private sein.
- Methoden, die Verhalten nach aussen zur Verfügung stellen, sollten public sein.
- Es sollte möglichst wenig public Methoden geben.
- Methoden werden als private definiert, wenn sie nur von anderen Methoden derselben Klasse aufgerufen werden. In der Regel wurden sie erzeugt, um
  - Verhalten, das verschiedene Methoden verwenden, nur einmal zu implementieren (Vermeidung von Codeduplikation).
  - den Code besser lesbar zu machen (statt einer «grossen» Methode eine Methode, die andere verwendet).
- Public
  - sichtbar in der Klasse selbst (also in ihren Methoden), in Methoden abgeleiteter Klassen und für den Benutzer von Instanzen der Klasse
  - public Elemente bilden die für alle sichtbaren Teile einer Klassendefinition
- Private
  - sichtbar lediglich in der Klasse selbst
  - für abgeleitete Klassen und für Benutzer von Instanzen verdeckt
  - private Elemente immer dann verwenden, wenn implementierungsabhängige Details zu verstecken sind (auch in abgeleiteten Klassen nicht sichtbar)
- Protected
  - sichtbar in der Klasse selbst und in Methoden abgeleiteter Klassen
  - zusätzlich sichtbar in Klassen desselben Pakets
  - protected Elemente sind vor Zugriffen von aussen geschützt, können aber von abgeleiteten Klassen verwendet werden
- Elemente ohne Zugriffsmodifizierer (Standard-Sichtbarkeit)
  - sichtbar in der Klasse selbst
  - zusätzlich sichtbar nur innerhalb des Pakets, zu dem diese Klasse gehört, in anderen Paketen dagegen unsichtbar
  - Element mit Standard-Sichtbarkeit verhalten sich innerhalb des Pakets wie public und ausserhalb wie private Elemente

6) Sie kennen die Begriffe Kohäsion und Kopplung

- Kohäsion
  - der innere Zusammenhalt einer Klasse gilt analog auch für:

→ Methoden
→ Packages und Komponenten

- Kopplung
  - der Zusammenhang zwischen Klassen
- Ziel von gutem Klassen-Design
  - Starke Kohäsion → starker innerer Zusammenhalt
  - lose Kopplung → loser Zusammenhang zwischen Klassen
- Was heisst starke Kohäsion?
  - 1 Klasse repräsentiert 1 logische Einheit (Entität)
  - 1 Methode repräsentiert 1 zusammenhängende Aufgabe
  - jede Klasse handhabt ihre Daten möglichst selber (Responsibility-Driven Design)
  - es liegt möglichst wenig Code-Duplikation vor
  - Potenzielle Änderungen haben nur lokale Auswirkungen
    → Frühzeitig daran denken
  - Nur eine Klasse bzw. möglichst wenige Klassen kapseln Informationen betreffend des Unser-Interfaces.
- Was heisst lose Kopplung?
  - Das Klassendiagramm zeigt nur wenige Beziehungen zwischen den Klassen, d.h. Klassen sind möglichst unabhängig voneinander.

7) Sie wissen, wie sich Kopplung im Source Code manifestiert

- siehe Punkt 6

8) Sie kennen je 3 Folgen von gutem und von schlechtem Klassendesign

- Folgen von schlechtem Klassen-Design
  - langsame Entwicklung
  - schwierige Entwicklung
  - fehlerhafte Entwicklung
  - abgebrochene Entwicklung
  - nachträgliche Änderungen schwierig umsetzbar
- Folgen von gutem Klassen-Design
  - Lesbarkeit
  - Testbarkeit
  - Wiederverwendung

9) Sie können das Konzept der Vererbung anwenden und kennen Vor- und Nachteile

- Vererbung (vereinfacht)
  - Eine abgeleitete Klasse besitzt («erbt») Methoden und Attribute der Basisklasse
  - Sie kann auf diese zugreifen (falls nicht durch Zugriffsmodifizierer eingeschränkt)
  - Schlüsselwort **extens** definiert die Vererbungs-Beziehung

```
public class Unterklasse extends Basisklasse {

}
```

- Schlüsselwort **super** verwendet im Konstruktor der Unterklasse
  - Vorgängig zum Kreieren des Objekts der Unterklasse muss zuerst das Objekt der Basisklasse initialisiert (erzeugt) werden.
  - Erzeugung erfolgt automatisch, falls Basisklasse einen parameterlosen Konstruktor hat (sonst Syntax-Fehler)
  - andernfalls muss die erste Anweisung im Konstruktor der Unterklasse **super** sein

```
public class Unterklasse extends Basisklasse {

    public Unterklasse()
    {
        super();
    }
}
```

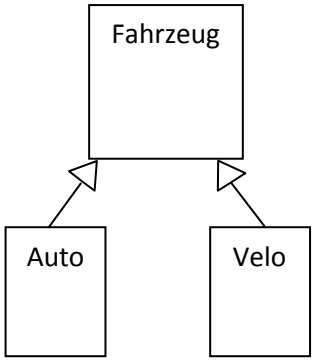
- Vorteile
  - hilft, Code-Duplikation zu vermeiden
  - unterstützt Code-Wiederverwendung
  - vereinfacht den Unterhalt
  - unterstützt einfache Erweiterungen bestehender Klassen
- Nachteile
  - erhöht die Kopplung (neue Abhängigkeiten zwischen den Unterklassen und der Basisklasse)

10) Sie verstehen das Konzept der Polymorphie von Klassentyp-Variablen

- Variablen eines Klassentyps in Java sind polymorph
  - Sie können sowohl Objekte des deklarierten Types aufnehmen wie auch alle Objekte aller abgeleiteten Subtypen.

11) Sie können das Konzept des Subtyping, insbesondere das entsprechende Handling von Referenzvariablen, nutzen

- Subtypen
  - Analog zur Klassenhierarchie bilden Klassentypen ebenfalls eine Hierarchie
  - Der Klassentyp einer Unterklasse ist ein Untertyp (Subtyp) des Basis-Klassentyps.
- Variablen und Subtypen
  - Variablen referenzieren ein Objekt des deklarierten Typs oder auch Objekte eines Subtyps des deklarierten Typs
- Substitution
  - Subtypen Objekte können verwendet werden, auch wenn ein Basisklassen Objekttyp erwartet wird




```
Fahrzeug f1 = new Fahrzeug();
Fahrzeug f2 = new Auto();
Fahrzeug f3 = new Velo();

//falsch
Auto a2 = new Fahrzeug();
```

```
Fahrzeug fahrzeug1;
Fahrzeug fahrzeug2;
Auto auto;
Velo velo;

public void ObjektErstellen()
{
//implizit
auto = new Auto();
fahrzeug1 = auto;

//explizit
auto = (Auto)fahrzeug1;

//Compiler-Fehler (Subklasse - Subklasse)
velo = (Velo)auto;

//Laufzeit-Fehler
fahrzeug2 = auto;
velo = (Velo)fahrzeug2;
```

Klassen in Java:

12) Sie kennen den prinzipiellen Aufbau einer Klasse

```
public class Test {

    //1. Instanzvariablen, Attribute, Felder, Membervariablen
    //2. Konstruktor
    //3. Methoden / Instanzmethoden

}
```

- 1. Eigenschaften eines Objekts
- 2. Initialisierung eines Objekts
- 3. Verhalten eines Objekts

13) Sie wissen, wozu Instanzvariablen, Methoden und Konstruktoren dienen

- Instanzvariablen

```
public class Test {

    private int position;
}
```

  - Halten die Eigenschaften eines Objekts fest
  - Jede Instanzvariable dient zum Speichern von Daten und besitzt einen entsprechenden Datentyp
  - Daten bzw. Datenwerte können im Verlaufe der Zeit verändert werden
  - Die Gesamtheit der Datenwerte eines Objekts definieren seinen aktuellen Zustand
  - Instanzvariablen existieren zeitlebens eines Objekts
  - Sie sind innerhalb der ganzen Klasse sichtbar / ansprechbar

- Sie sollten trotz automatischer Initialisierung explizit initialisiert werden
- Sie erfordern zur Laufzeit für jedes Objekt Speicherplatz auf dem Heap

- Konstruktor

```
public class Test {

    private int position;

    public Test()
    {
        position = 0;
    }
}
```

  - Ermöglicht beim Erzeugen das ordentliche initialisieren eines Objekts
  - Beinhaltet ausführbare Anweisungen
  - Wird beim Erzeugen eines Objekts automatisch aufgerufen bzw. abgearbeitet
  - Initialisiert typischerweise die Instanzvariablen
  - Besitzt typischerweise Parameter
  - Eine Klasse kann mehrere verschiedene Konstruktoren besitzen
  - Vergleichbar mit einer speziellen Methode ohne Rückgabewert
- Methoden
  - Wichtiges Konstrukt der strukturierten und objektorientierten Programmierung (vgl. Prozedur, Unterprogramm, Subroutine, Funktion, Methode)
  - Abstraktion (einfacher Methodenaufruf kann von komplexem Sachverhalt abstrahieren)
  - Wiederverwendung (1 Mal implementieren, n Mal wieder verwenden)
  - Verhalten von Objekten realisieren
  - Zugriff bzw. Kommunikation / Interaktion unter Objekten ermöglichen
    - Methodenaufruf = Anfrage an ein Objekt
    - Rückgabewert = Antwort des Objektes
  - Methodenkopf
    - Zugriffmodifizierer typisch public oder private
    - spezieller Rückgabetyp «void», d.h. kein Rückgabewert
    - eine Methode kann auch keine Parameter besitzen
    - Signatur gemäss Java-Spezifikation: Methodenname inkl. allen Parametern
    - Innerhalb derselben Klasse müssen alle Methoden in ihrer Signatur unterschiedlich sein. (unterschiedliche Namen oder Parameter mit unterschiedlichen Datentypen oder unterschiedlicher Anzahl Parameter / nur Rückgabetyp von z.B. int auf long ändern genügt nicht!)
  - Methodenrumpf
    - Block nach Methodenkopf

- beinhaltet Deklarationen und Anweisungen
- Anweisungen können ihrerseits wieder Blöcke beinhalten
- Innerhalb eines Blockes werden Deklarationen / Anweisungen durch «;» voneinander getrennt
- Rückgabewert (falls deklariert) wird mittels «return» zurückgegeben
- «return» beendet unmittelbar die Ausführung einer Methode; es darf keine unerreichbaren Anweisungen geben

14) Sie kennen mindestens je 3 Eigenheiten von Instanzvariablen, Klassenvariablen, lokalen Variablen und von formalen Parametern.

- Instanzvariablen → siehe Punkt 13
- Klassenvariablen
  - werden in der Klasse und nicht im Objekt gespeichert
  - Verwendung: wenn Werte der Variablen für alle Objekte der Klasse gleich sein sollen
  - Definition mit Schlüsselwort **static**
  - weil die Variable der Klasse und nicht dem Objekt gehört, kann man auf sie zugreifen, ohne zuvor ein Objekt zu instanzieren.
    - System.out → definiert als public static PrintStream out;
    - Math.round → definiert als public static long round(double a);
- lokale Variablen

```
public int getVolume()
{
    int r3;
    r3 = 1 + 2;
    return r3;
}
```

  - werden innerhalb einer Methode bzw. Block deklariert
  - müssen explizit initialisiert werden (erfolgt nicht automatisch)
  - sind nur während der Ausführung des entsprechenden Blocks sichtbar
  - dienen als temporären Speicher
  - Erfordern intern Speicherplatz auf dem Stack
- formale Parameter

```
public void setLocation(int pos, int alti)
{
    position = pos;
    altitude = alti;
}
```

  - werden im Kopf einer Methode / eines Konstruktors deklariert
  - sind sozusagen spezielle lokale Variablen
  - Beim Aufruf der Methode / des Konstruktors müssen Initialisierungswert bzw. aktuelle Parameter übergeben werden
  - Variablen werden bei der Übergabe kopiert
  - Sind nur während der Ausführung der Methode / des Konstruktors existent bzw. nur innerhalb der Methode / des Konstruktors sichtbar
  - Über Parameter lässt sich das Verhalten / die Initialisierung von Objekten steuern
  - Erfordern intern Speicherplatz auf dem Call-Stack

15) Sie können die Rollen von Methodenkopf und Methodenrumpf erklären

- siehe Punkt 13

16) Sie können einen Konstruktor-Kopf ohne/mit Parameter(n) formulieren

```
public class Test {
   private int position;

    public Test()
    {
        position = 0;
    }

   public Test(int i)
   {
       position = i;
   }
}
```

17) Sie können einen Methoden-Kopf ohne/mit Parameter(n) bzw. ohne/mit Rückgabewert formulieren

```
public void setLocation(int pos, int alti)
{
    position = pos;
    altitude = alti;
}

public void starten()
{

}

public int rechnen(int a, int b)
{
    return (a * b);
}
```

18) Sie können überladene Konstruktoren und Methoden erstellen

```
public Test()
{
    position = 0;
    altitude = 0;
}

public Test(int pos)
{
    position = pos;
    altitude = 0;
}

public Test(int pos, int alti)
{
    position = pos;
    altitude = alti;
}
```

```
public int rechnen(int a, int b)
{
    return (a * b);
}

public int rechnen(int a, int b, int c)
{
    return (a * b * c);
}
```

19) Sie können Accessor Methoden formulieren

```
public boolean istKanalOffen()
{
    return status;
}
```

20) Sie können Mutator Methoden formulieren

```
public void schliessen()
{
    status = false;
}
```

21) Sie können den Unterschied zwischen einem formalen und einem aktuellen Parameter erklären

```
public void starten()
{
     rechnen(10, 12);
}
```

aktuelle Parameter

```
public void rechnen(int zahl1, int zahl2)
{
    resultat = zahl1 * zahl2;
}
```

formale Parameter

22) Sie können Instanzvariablen und lokale Variablen deklarieren

- Instanzvariablen

```
public class Test {

    private int position;
}
```

- lokale Variablen

```
public int getVolume()
{
    int r3;
    r3 = 1 + 2;
    return r3;
}
```

23) Sie wissen, wie man mit Objekten arbeitet

24) Sie wissen, wieso und wie man Klassen importiert

- Klassen-Bibliothek hat viele Klassen
  → benötigt Struktur
- Einteilung von zusammengehörigen Klassen in Pakete (package) und Unterpakete

```
import java.util.Random;
import java.util.Arrays;
import java.util.*; //alle Klassen des Pakets importieren
```

25) Sie können Klassenmethoden definieren

- In statischen Methoden besteht kein Zugriff auf nicht statische Attribute
- In statischen Methoden dürfen keine nicht-statischen Methoden aufgerufen werden.

```
public static double roundScale(double d)
{
    return Math.rint(d * 100) / 100;
}
```

26) Sie können Klassenvariablen definieren

```
public static PrintStream out;
```

27) Sie können Konstanten und Klassenkonstanten definieren

```
public static final double PI = 3.141592653589793;
private final int i = 7;
```

28) Sie können eine main()-Methode implementieren

- Beim Start der Klasse wird immer zuerst die Main-Methode aufgerufen
- In dieser Methode können dann Objekte erzeugt und weitere Methoden aufgerufen werden.

```
public static void main(String[] args) {

}
```

29) Sie verwenden die Interfacebeschreibung einer Klasse (vgl. API-Dokumentation), um sie in Ihren Programmen zu nutzen

30) Sie können einfache Klassenhierarchien in Java implementieren

- siehe Subtyping Punkt 11

```
public class Fahrzeug {

}
public class Velo extends Fahrzeug {

}
public class Auto extends Fahrzeug {

}
```

31) Sie verstehen die Bedeutung der Klasse Object und kennen überblicksmässig deren Basisfunktionalität

- Implizit ist die Klasse Objekt die Basisklasse für alle Klassen.
- Somit ist jede Klasse eine Unterklasse der Klasse Objekt.

32) Sie kennen die vier Zugriffsmodifizierer und wissen, wie man sie anwendet

- siehe Punkt 5

Datentypen und Operatoren in Java:

33) Sie können die acht primitiven Datentypen benennen und einordnen

Schlüsselwort/ Typ	Länge in Byte	Belegung (Wertebereich)
boolean	1	true oder false
char	2	16-Bit-Unicode-Zeichen (0x0000 ... 0xFFFF)
byte	1	–2^7 bis 2^7 – 1 (–128 ... 127)
short	2	–2^15 bis 2^15 – 1 (–32.768 ... 32.767)
int	4	–2^31 bis 2^31 – 1 (–2.147.483.648 ... 2.147.483.647)
long	8	–2^63 bis 2^63 – 1 (–9.223.372.036.854.775.808 ... 9.223.372.036.854.775.807)
float	4	1,40239846E–45f ... 3,40282347E+38f
double	8	4,94065645841246544E–324 ... 1,79769131486231570E+308

Operator	Typ	Beschreibung
++, --	arithmetisch	Inkrement und Dekrement
+, -	arithmetisch	unäres Plus und Minus
!	boolean	logisches Komplement
*, /, %	arithmetisch	Multiplikation, Division, Rest
+, -	arithmetisch	Addition und Subtraktion
+	String	String-Konkatenation
<, <=, >, >=	arithmetisch	numerische Vergleiche
==, !=	primitiv	Gleich-/Ungleichheit von Werten
==, !=	Objekt	Gleich-/Ungleichheit von Referenzen
&	boolean	logisches Und
^	boolean	logisches Xor
|	boolean	logisches Oder
&&	boolean	logisches konditionales Und, Kurzschluss
||	boolean	logisches konditionales Oder, Kurzschluss
=	jeder	Zuweisung

34) Sie können vergleichende Bedingungen formulieren und logische Operatoren anwenden

35) Sie können arithmetische und logische Ausdrücke auswerten

- siehe Punkt 34

36) Sie können das implizite und das explizite Type-Casting anwenden

von	nach (implizit)
byte	char, short, int, long, float, double
char, short	int, long, float, double
int	long, float, double
long	float, double

float	double
«alle»	nach String

```
public void rechnen()
{
    int zahl1 = 5;
    int zahl2 = 2;
    float kommazahl = 2.0f;
    float ergebnis = 0;

    ergebnis = zahl1 / kommazahl;       //2.5
    ergebnis = zahl1 / zahl2;           //2.0
    ergebnis = (float)zahl1 / zahl2;    //2.5
    ergebnis = (float)(zahl1 / zahl2);  //2.0
    ergebnis = zahl1 / (int)zahl2;      //2.0
}
```

- Ganzzahlige Werte: wird eine längere Zahl an eine Variable eines Datentyps mit kleinerem Wertebereich angepasst, werden die führenden Stellen abgeschnitten.
- Wird eine Gleitkommazahl an eine Variable eines ganzzahligen Datentyps angepasst, werden die Nachkommastellen abgeschnitten.

37) Sie können Arrays deklarieren, initialisieren sowie auf Arrays zugreifen

- Kann Werte eines bestimmten elementaren Datentyps oder Objekts (Referenzen) eines bestimmten Klassentyps aufnehmen.
- Auf die Werte bzw. Objekte kann via Index (ganze Zahl) direkt zugegriffen werden: 0 to length - 1
- Ein Array selbst wird wie ein Objekt gehandhabt.
- Mit dem Erzeugen eines Array wird seine Länge (length) unveränderlich festgelegt (statische Datenstruktur)
- Bei der Ausführung überprüft die Java Virtual Machine vorgängig jeden Array-Zugriff (unzulässig → Exception)

```
public void array()
{
    int[] a;
    a = new int[5];

    //oder

    int[] b = new int[5];

    a[0] = 1;
    a[1] = 2;

    //Schleife
    for(int i = 0; i < a.length; i++) {
        a[i] = i * 2;
    }

    //Deklaration inkl. Initialisierung
    int[]c = {0, 2, 4, 6, 8};
}
```

38) Sie können einfache Methoden mit Array-Parametern implementieren und diese aufrufen

```
public void array()
{
    int[] table = new int[10];

    for(int i = 0; i < table.length; i++) {
        table[i] = i;
    }
    int total = sum(table);
}

private int sum(int[] values)
{
    int s = 0;
    for(int i = 0; i < values.length; i++){
        s = s + values[i];
    }
    return s;
}
```

Objektorientierte Programmierung:

39) Sie können die Selektion mit if anwenden

- Operatoren:

<	kleiner	<=	kleiner oder gleich
==	gleich	!=	ungleich
>	grösser	>=	grösser oder gleich

```
if(expression) {
    //statements
}
else {  //optional
    //statements
}
```

40) Sie können eine einfache Selektion mit switch formulieren

```
switch(day) {
    case 1: dayString = "Monday";
            break;
    case 2: dayString = "Tuesday";
            break;
    default: dayString = "invalid day";
            break;
}
```

- Wenn **break** fehlt werden alle nachfolgenden **case** auch geprüft.

41) Sie kennen die Eigenheiten der verschiedenen Schleifen-Anweisungen (while, do, for, foreach)

- While
  - Anzahl Schleifendurchgänge unbekannt
  - eventuell kein Durchgang
  - ist abweisend

```
while(expression){
    //statements
}
```

→ solange bis Ausdruck **false** ist

- do-While
  - Anzahl Schleifendurchläufe unbekannt
  - mindestens 1 Schleifen-Durchlauf
  - nicht abweisend

```
int nr = 0;
do {
    nr = (int)(Math.random() * 6) + 1;
} while (!(nr == 6));
```

  → solange bis Ausdruck **false** ist
- for
  - Anzahl Schleifendurchläufe bekannt oder im Voraus (eventuell erst zur Laufzeit) berechenbar
  - eventuell kein Schleifen-Durchlauf
  - Initialisierung und Finalisierung dürfen auch mehrere Anweisungen beinhalten (durch Komma getrennt)
  - abweisend

```
public int fak(int n)
{
    int result = 1;
    for(int i = 0; i < n; i++) {
        result = result * i;
    }
    return result;
}
```

  - Finalisierung (i++) nach jedem Schleifendurchlauf
  - Initialisierung (int i = 0) 1 Mal zu Beginn
- In Java sind alle 3 Schleifen-Anweisungen gleich mächtig.
- break-Anweisung innerhalb eines Schleifen-Rumpfes bewirkt das sofortige Beenden der Schleife
- Eine continue-Anweisung innerhalb eines Schleifen-Rumpfes bewirkt das Springen ans Ende des Rumpfes
- for each
  - Ermöglicht das einfache Zugreifen auf alle Elemente einer Collection; aber auch auf jene eines Arrays
  - Auf Werte eines primitiven Datentypes kann nur lesend zugegriffen werden

```
char[] a = {'a', 'b', 'c', 'd', 'e'};
for(char ch : a) {
    System.out.println(ch);
}
```

42) Sie können für eine gegebene Aufgabenstellung die geeignete Schleifen-Anweisung bestimmen

43) Sie können einfache Methoden mit Schleifen-Anweisungen implementieren

Klassen und Methoden der Java Klassenbibliothek:

44) Sie können Klassen der Klassenbibliothek importieren und wissen, wieso man Klassen importiert

- siehe Punkt 24

45) Sie wissen, dass die Klasse String existiert, und können Variablen des Typs String deklarieren

```
private String name;
```

46) Sie können die String-Konkatenation mit + ausführen

```
neuesWort = "Weihnacht" + "s" + "mann";
```

47) Sie können Strings vergleichen

```
String name = "Hans";
if(name.equals("Hans")){ //Test auf Inhalt

}

if(name == "Hans"){ //Test auf Referenz

}
```

48) Sie verwenden die Methode System.out.println()

```
System.out.println("drucken");
```

49) Sie kennen überblicksmässig das Konzept der Wrapper-Klassen (z.B. Integer, Long, Float)

- Klasse Float aus package java.lang
- Deklaration und Initialsierung der ArrayList für Float Objekte (ohne Generics)

```
ArrayList floats = new ArrayList();
```

- Zum Einfügen und Auslesen muss der primitive Datentyp float in ein Objekt der Klasse Float «gewrappt» werden.

```
float f = 3.2f;
Float floatObject;

floatObject = new Float(f);          //in Float-Objekt einpacken
floats.add(floatObject);             //einfügen

floatObject = (Float)floats.get(0);  //auslesen
f = floatObject.floatValue();        //konvertieren
```

50) Sie können Wrapper Klassen anwenden und verstehen den auto-boxing / -unboxing Mechanismus

- Durch die Generics in Java entfällt das manuelle Einpacken / Auspacken (boxing / unboxing) der primitiven Datentypen.
- Der Compiler hat genug Informationen und kann das wrapping / unwrapping automatisch übernehmen und die Typen überprüfen.

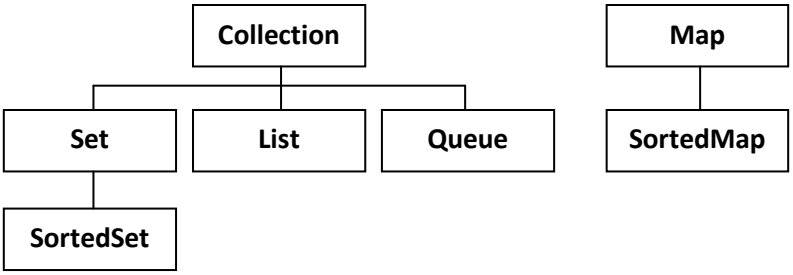
```
ArrayList<Float> floats = new ArrayList<Float>();
float f = 3.2f;

floats.add(f);       //einfügen, Compiler übernimmt boxing
f = floats.get(0);   //auslesen, Compiler übernimmt unboxing
```

51) Sie verstehen "Generics" und können sie anwenden.

- z.B. ArrayList für bestimmte Objekte

```
ArrayList<Float> floats = new ArrayList<Float>();
```

52) Sie kennen die Container-Klassen überblicksmässig



- pro Container-Typ eine Schnittstelle (Interface)
- Verschiedene Implementierungen der Container für verschiedene Aufgabenstellungen
  - ArrayList: Array Implementation einer Liste
- List
  - Geordnete Collection (Sequenz)
  - Benutzer hat Kontrolle, wo Elemente in der Liste abgelegt werden → Index (Position der Liste)
  - zero-based Index (erstes Element hat Index 0)
  - kann mehrmals das gleiche Objekt referenzieren
  - Iteration
    - Iterator
    - ListIterator
- Set
  - Collection, welche keine zwei gleichen Objekte enthält (auch nur höchstens ein Null-Element)
  - Iteration
    - Iterator
- Map
  - verbindet einen Schlüssel mit einem Objekt (ähnlich Wörterbuch)

- jeder Schlüssel muss eindeutig sein (jedoch gleiches Objekt mit verschiedenen Schlüsseln möglich)
- drei Sichten: - Set von Schlüsseln
  - Collection von Werten
  - Set von Schlüssel / Objekt Paaren
- Iteration
  - keine

53) Sie kennen das Konzept der Iteratoren und können dieses anwenden

- erlauben das Traversieren einer Collection ohne Bezug auf den Index zu nehmen (falls überhaupt vorhanden)
- jedes Element wird während einer Iteration genau einmal zurückgegeben
- bemerkt in der Regel (aber Vorsicht) eine Veränderung der Collection während eine Iteration läuft
- zwei Arten
  - Iterator
    - boolean hasNext()
    - E next()
    - void remove()

```
ArrayList<Float> floats = new ArrayList<Float>();
Iterator<Float> iterator;
float fl;

iterator = floats.iterator();
while(iterator.hasNext()) {
    //Statements
    fl = iterator.next();
}
```

  - ListIterator
    - wie Iterator, aber zusätzlich
    - boolean hasPrevious()
    - E previous()

54) Sie kennen verschiedene Möglichkeiten, um alle Elemente einer Collection (z.B. ArrayList) zu traversieren bzw. bearbeiten zu können (for, foreach, Iterator)

- siehe Punkt 41 und 53

55) Sie können die wichtigsten Methoden der Klassen List/ArrayList, Set/HashSet und Map/HashMap anwenden

- List:
  - boolean add(E element)
  - E set(int index, E element)
  - E get(int index)
- ArrayList:
  - int size()
  - boolean isEmpty()
  - get(int index)

    - set(int index, E element) → Replace the element
    - iterator()
    - listIterator()
    - add()
- Map:
    - V steht für den Klassentypen des Wert-Objekts, K steht für den Klassentyp des Schlüssel-Objektes.
    - V put(K key, V value)
    - V get(Object key)
    - int size()
    - boolean containsKey(Object key)
    - boolean containsValue(Object value)
    - boolean isEmpty()
- Set:
    - E steht für den Klassentyp des Elements
    - boolean add(E o)
    - int size()
    - Iterator<E>iterator()
    - boolean contains(Object o)
    - boolean isEmpty()
    - boolean remove(Object o)

Algorithmen:

56) Sie kennen die Merkmale eines Algorithmus

- Verfahren, mit dem eine Problemkategorie gelöst werden kann:
  - schrittweises Verfahren
  - nächster Schritt eindeutig
  - ausführbare Schritte
  - endet nach endlich vielen Schritten
- Computer ist prädestiniert, Algorithmen auszuführen
- zentrale Themen der Informatik und Mathematik
  - Algorithmentheorie
  - Komplexitätstheorie
  - Berechenbarkeitstheorie
- eigenes Gebiet in der Informatik
- Beispiele von Algorithmen
  - Suchverfahren
  - Sortieralgorithmus (Bubblesort, Quicksort)
  - Berechnung von Schaltjahren
  - zyklische Redundanzprüfung (CRC)
  - kryptographisches Verfahren (z.B. DES / 3DES)
  - Audio- / Video-Kompression (MPEG)
  - «Berechnung» von Zufallszahlen

57) Sie können den Begriff der Komplexität eines Algorithmus definieren

- Komplexität (auch Aufwand oder Kosten) eines Algorithmus:
  - maximaler Ressourcenbedarf in Abhängigkeit der Eingabedaten (häufig Grösse der Datenmenge oder Werte)
- Ressourcen sind:
  - Anzahl benötigter Rechenschritte (Zeitkomplexität)
  - Speicherbedarf (Speicherkomplexität)
- Wie wächst der Ressourcenbedarf, wenn mehr Daten zu verarbeiten sind?
  → verdoppelt oder quadriert
- kein Interesse am präzisen Aufwand eines konkreten Programms auf einem bestimmten Computer

58) Sie verstehen den Begriff "Ordnung eines Algorithmus" und können die Ordnung einer gegebenen Funktion bestimmen

- Abhängigkeit aus Analyse des Algorithmus
- jedoch kein Interesse am genauen Funktionsverlauf f(n), nur an der Ordnung O
- Ordnung ignoriert konstante Faktoren
  - diese können praktisch relevant sein

- Ordnung berücksichtigt Verhalten bei kleinen n nicht
  - langsamere Algorithmen bei kleinen n oft brauchbar / sinnvoll
- Die exakte mathematische Analyse von vielen Algorithmen ist schwierig oder unmöglich
- Algorithmen mit exponentieller Ordnung dauern trotz schnellen Computern oft zu lange → Suche nach schnelleren Algorithmen

$f(n) = n^3 + 0.1*2^n$	$f(n) = 2 + 37*n^3 + 0.01 * n^4$
$f(n) = O(2^n)$	$f(n) = O(n^4)$
$f(n) = 1000*n + n^3$	$f(n) = 5326 + \ln(n)$
$f(n) = O(n^3)$	$f(n) = O(\ln(n))$
$f(n) = \ln(n) + 1000*n$	$f(n) = n^7 + n!$
$f(n) = O(n)$	$f(n) = O(n!)$

Konstant	$O(1)$
Logarithmisch	$O(\ln(n))$
Linear	$O(n)$
n log n	$O(n * \ln(n))$
Polynomial ①	$O(n^m)$
Exponential	$O(d^n)$
Fakultät	$O(n!)$

59) Sie kennen das Prinzip der Rekursion

- Eine Funktion / Methode heisst rekursiv, wenn sie sich (direkt oder indirekt) selber aufruft
- Eine Rekursion besteht aus einer Rekursionsbasis und einer Rekursionsvorschrift
- Die Rekursionsbasis beschreibt, wie ein einfaches Problem der angegebenen Problemklasse gelöst werden kann
- Die Rekursionsvorschrift gibt an, wie ein Problem auf ein einfacheres Problem zurückgeführt werden kann. Innerhalb der Rekursionsvorschrift wird die Funktion / Methode selber wieder aufgerufen

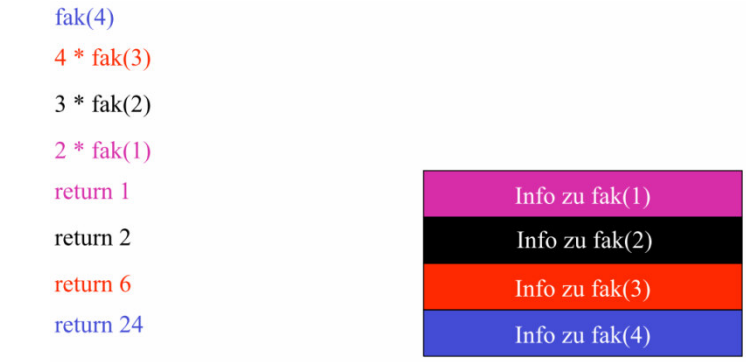
60) Sie können die Rekursionsbasis und Rekursionsvorschrift bestimmen

```
static void permute(char[] a, int endIndex)
{
    if (endIndex == 0) {                    // Rekursionsbasis
        System.out.println(a);
    }
    else {                                  // Rekursionsvorschrift
    for (int i=0; i <= endIndex-1; i++)  {
        exchange(a, i, endIndex);
        permute(a, endIndex-1);
        exchange(a, i, endIndex);
    }
    permute(a, endIndex-1);
}
```

61) Sie wissen, wie eine rekursive Methode abgearbeitet wird

- Auf dem Stack im Speicher (Call Stack) befinden sich die Informationen der lokalen Variablen und Parameter der aktuell ausgeführten Methode m1().
- Ruft diese Methode m1() eine andere Methode m2() auf, so werden zusätzlich die Informationen von m2() auf dem Call Stack abgelegt (die Informationen von m1() sind dann nicht sichtbar).
- Jeder neue Methodenaufruf bedeutet, dass mehr Platz auf dem Call Stack benötigt wird.

- Berechnung der Fakultät von 4: fak(4)



62) Sie können das Konzept "Rekursion" an einem Beispiel erklären, z.B. mit den Türmen von Hanoi

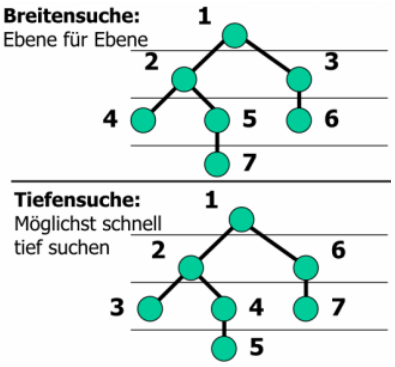
```
 * Bewegt n Scheiben von Turm a nach Turm c und benutzt als
 * Zwischenspeicher Turm b.
 */
private static void bewege (char a, char b, char c, int n)
{
if (n == 1)
    System.out.println("Lege die oberste Scheibe von " +
       "Turm " + a + " auf Turm " + c + ".");
else {
     bewege(a, c, b, n-1);
     bewege(a, b, c, 1);
     bewege(b, a, c, n-1);
    }
}
public static void main (String[] args)
{
    bewege('a', 'b', 'c', 4);
}
```

63) Sie können einfache rekursive Methoden implementieren

```
public long fak(int n)
{
    if ((n == 1) || (n == 0)){
        return 1;
    }
    else {
        return n * fak(n-1);
    }
}
```

64) Sie können das Konzept "Backtracking" erklären und anwenden

- Backtracking ist eine
  - systematische Art der Suche in einem vorgegebenem Lösungsraum
- Wenn eine Teillösung in eine
  - Sackgasse führt, dann wird der jeweils letzte Schritt rückgängig gemacht («back-tracking»)



Backtracking ist Tiefensuche!

65) Sie können an einem Beispiel erläutern, was die Eigenschaft "Stabilität" eines Sortieralgorithmus bedeutet

- Falls mehrere Objekte mit gleichem Schlüssel vorliegen
  - Reihenfolge dieser Objekte vor und nach dem Sortieren
    - gleich → stabiler Sortieralgorithmus
    - ungleich → unstabiler Sortieralgorithmus
  - stabil:

unsortiert

5	33	12_1	13	8	1	41	12_2

sortiert

1	5	8	12_1	12_2	13	33	41

  - stabile Algorithmen: direktes Einfügen, Bubblesort, Mergesort
  - unstabile Algorithmen: direktes Auswählen, Shellsort, Quicksort

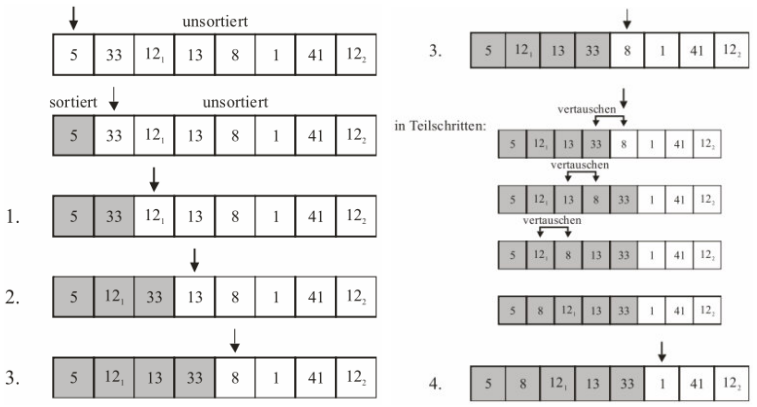
66) Sie kennen die Ordnungen der behandelten direkten/einfachen und höheren Sortieralgorithmen

- Einfache / Direkte Sortieralgorithmen → Aufwand $O(n^2)$
- Höhere Sortieralgorithmen → Aufwand O(n * log n)

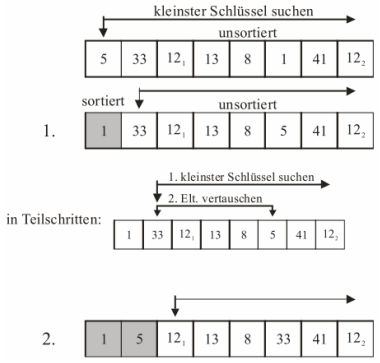
	vorsortiert	umgekehrt sortiert	∅
directInsert	O(n)	$O(n^2)$	$O(n^2)$
direktes Ausw.			$O(n^2)$
direktes Aust. Bubblesort			
Shell (optimal mit Folgen $O(n^{1,2})$)			$O(n^2)$
Quick	$O(n^2)$		O(n * log n) O(2 * n * log n)
Mergesort			O(n * log n)

67) Sie können die Arbeitsweise der behandelten direkten/einfachen und höheren Sortieralgorithmen erklären und an einfachen Beispielen aufzeigen

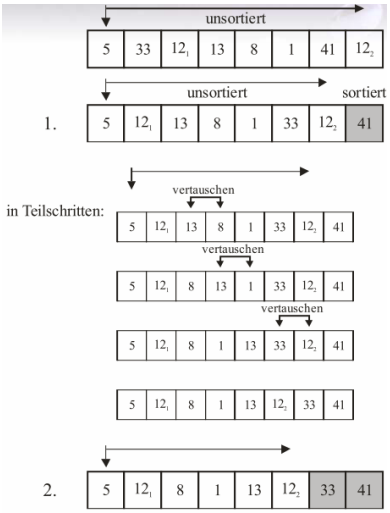
- Direktes Einfügen (Insertion Sort)
  - Das zu sortierende Array wird in einen unsortierten Teil (weiss) und einen sortierten Teil (grau) unterteilt.
  - Das jeweils erste Element im unsortierten Teil wird gemäss seinem Schlüssel im sortierten Teil eingefügt.
  - Alle Elemente mit grösserem Schlüssel müssen somit nach rechts verschoben werden.
  - Die Suche nach der Einfüge-Position wird mit dem Verschieben kombiniert.
  - Nach n-1 Durchläufen ist das Array sortiert
  - Dieses benachbarte «Sesselrücken» ist sicher nicht effizient! Ideal wäre ein Verschieben über grössere Distanzen.



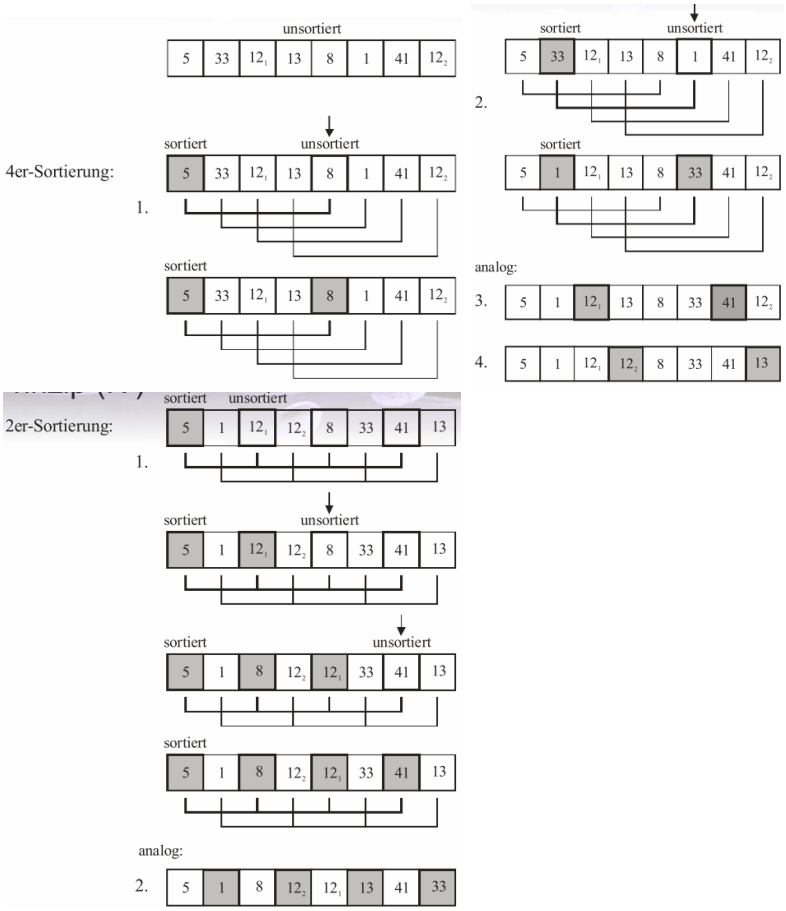
- Direktes Auswählen (Selection Sort)
  - Das zu sortierende Array wird in einen unsortierten Teil (weiss) und einen sortierten Teil (grau) unterteilt.
  - Im unsortierten Teil des Arrays wird das Element mit dem kleinsten Schlüssel ausgewählt bzw. gesucht. Dieses vertauscht man dann mit dem ersten Element des unsortierten Teils.
  - Nach n-1 Durchläufen ist das Array sortiert
  - Der Aufwand für das direkte Auswählen ist grundsätzlich ebenfalls von der Ordnung (n^2). Das Vertauschen der Elemente über grössere Distanzen wirkt sich aber positiv auf die Effizienz aus.



- Direktes Austauschen (Bubble Sort)
  - Das zu sortierende Array wird in einen unsortierten Teil (weiss) und einen sortierten Teil (grau) unterteilt.
  - Zwei benachbarte Elemente werden einfach ausgetauscht, falls deren Schlüssel nicht aufsteigend geordnet ist.
  - Das Element mit dem grössten Schlüssel «blubbert» im Verlaufe eines Durchlaufs im unsortierten Teil ganz nach rechts. Der sortierte Teil wird damit um ein Element länger.
  - Nach n-1 Durchläufen ist das Array sortiert
  - Dieses sture benachbarte «Sesselrücken» ist sicher nicht effizient.

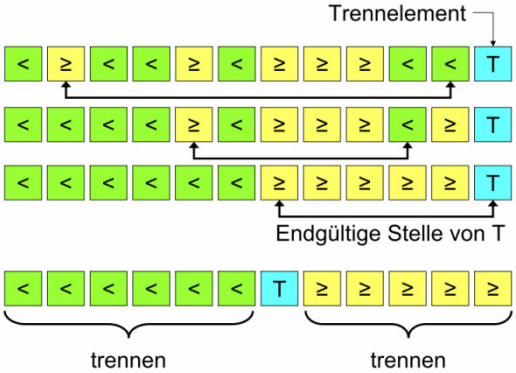


- Shellsort
  - Shellsort ist eine Verfeinerung des direkten Einfügens.
  - Nachteilig beim direkten Einfügen war das ausschliesslich benachbarte «Sesselrücken».
  - Bei Shellsort erfolgt das Sortieren in mehreren Stufen von «grob» bis «fein».
  - Bei der Grobsortierung erfolgt das Sortieren über «grosse Distanzen»; bei der Feinsortierung sind es - falls überhaupt noch notwendig - nur noch «kleine Distanzen».
  - Das eigentliche Sortieren erfolgt aber wie beim direkten Einfügen.



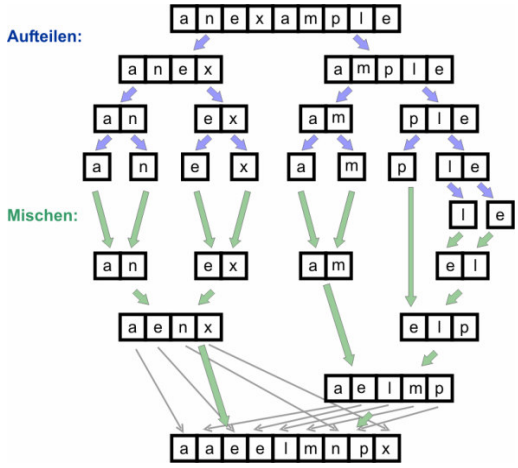
- Quicksort
  - Quicksort ist ein höherer Sortieralgorithmus mit einem durchschnittlichen Aufwand von O(n * log n).
  - Verwendet das «teile-und-herrsche» Prinzip
  - ist ein instabiles Sortierverfahren
  - Wird mittels Rekursion beschrieben:
    - Rekursionsvorschrift
      - Die zu sortierende Folge wird in zwei Teilfolgen getrennt.
      - Die Teilfolgen werden anschliessend mit dem gleichen Verfahren weiter getrennt.
    - Rekursionsbasis
      - Eine Folge von einem Element ist sortiert.

- Trennverfahren:
  - Bestimme ein Trennelement (idealerweise von der Wertigkeit her genau in der Mitte der Folge)
  - Ziele beim Trennen
    - Das Trennelement steht an seiner endgültigen Position.
    - Alle Elemente rechts vom Trennelement sollen grösser oder gleich diesem sein.
    - Alle Elemente links vom Trennelement sollen kleiner dem Trennelement sein.
- Bestimmen des Trennelementes
  - Ideal: Die Teilfolgen sind nach dem Trennen in etwa gleich gross
  - Jedoch: Zur Bestimmung des Trennelementes soll nur wenig Aufwand betrieben werden.
  - Zwei gängige Verfahren zur Bestimmung des Trennelementes:
    - Nimm das letzte Element der Folge.
    - median-of-three: Nimm das Element mit dem mittleren Wert von drei Elementen (z.B. folge[anfang], folge[mitte], folge[ende]) und tausche es mit dem letzten Element der Folge.
- Trennverfahren
  - Nimm das letzte Element der Folge als Trennelement.
  - Ordne die Folge so um, dass alle Glieder, welche kleiner dem Trennzeichen sind, nach links und grössere oder gleiche Elemente nach rechts verschoben werden.
  - Verschiebe (tausche) das Trennelement an seinen endgültigen Platz.
  - rekursiver Aufruf jeweils getrennt für die linke sowie rechte Teilfolge



- Mergesort
  - Mergesort ist ein rekursiver, stabiler Sortieralgorithmus
  - Funktioniert nach dem Prinzip «teile-und-herrsche» (ähnlich wie Quicksort)
  - Zerlegt eine Folge in zwei gleich grosse Teilfolgen
  - Sortiert jede Teilfolge für sich (rekursiver Aufruf solange Teilfolge länger 1)
  - Mischen (engl. merge)
    - Die sortierten, kleinen Teilfolgen werden im Reissverschlussverfahren zu einer grösseren Folge zusammengefügt.

- Am Ende ist die Folge sortiert.
- Konzept von Mergesort
  - Bei der Aufteilung in kleinere Teilfolgen ist die Grösse jeder Teilfolge ungefähr gleich (+/- 1 Element)
    - Binärbaum, wie bei Quicksort im besten Fall
    - Zeitkomplexität garantiert immer O(n * log n)
  - Das Verfahren benötigt nochmals soviel Speicherplatz wie die Folge (für das Mischen) → Speicherkomplexität O(n)
- Mergesort ist gegenüber der ursprünglichen Reihenfolge der Eingabedaten zeit-unempfindlich (speziell bei bereits sortierten Folgen)
- Mergesort ist ideal für:
  - sortieren von verketteten Listen
  - sortieren von Daten auf externen Datenspeichern
- Quicksort wird im Allgemeinen bevorzugt:
  - Bei zufälligen Eingabedaten braucht Mergesort im Mittel mehr Zeit als der klassische Quicksort.
  - Geschwindigkeitsvorteil von Quicksort: 10 % bis 50 %
- Mergesort als externes Sortierverfahren:
  - Daten so aufteilen, dass sie mit einem internen Sortierverfahren sortiert werden können
  - abspeichern als Datei
  - zwei Dateien parallel einlesen und in eine dritte Datei mischen



68) Sie kennen die Vor- und Nachteile der behandelten Sortieralgorithmen

- siehe Punkt 67

69) Sie verstehen die im Unterricht betrachteten konkreten Implementierungen von Sortieralgorithmen, z.B. von Quicksort

```
private void exchange(char[] a, int firstIndex, int secondIndex)
{
    char tmp;
    tmp            = a[firstIndex];
    a[firstIndex]  = a[secondIndex];
    a[secondIndex] = tmp;
}
/**
* Quicksort Algorithmus.
* @param a     Array zum Sortieren
* @param left  Linke Grenze, zu Beginn 0
* @param right Rechte Grenze, zu Beginn a.length-1
*/
public void quickSort(char[] a, int left, int right)
{
    int     up         = left;      // linke Grenze
    int     down       = right - 1; // rechte Grenze (ohne Trennelement)
    char    t          = a[right];  // rechtes Element als Trennelement
    boolean allChecked = false;     // austauschen beendet
    do {
        while (a[up] < t) {
            up++;              // suche grösseres (>=) Element von links an
        }
        while ((a[down] >= t) && (down > up)) {
            down--;            // suche kleineres(<) Element von rechts an
        }
        if (up < down) {           // Solange keine Überschneidung
            exchange(a, up, down); // austauschen
            up++; down--;          // linke und rechte Grenze verschieben
        }
        else {
            allChecked = true;     // Abbruch
        }
    } while (!allChecked);
    exchange(a, up, right);  // Trennelement an endgültige Position (a[up])
    if (left < up - 1)
        quickSort(a, left, up - 1);  // linke Hälfte
    if (up + 1 < right)
        quickSort(a, up + 1, right); // rechte Hälfte (ohne Trennelement)
}
```

70) Sie haben einen Überblick über Bibliotheksklassen von Java zum Sortieren

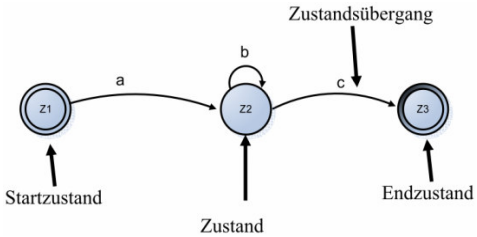
- Zwei Klassen enthalten Sortieralgorithmen
  - java.util.Arrays → Sortieren von Array
  - java.util.Collections → Sortieren von Listen
- zwei Sortierverfahren werden verwendet
  - modifizierter Mergesort (in Klasse Collection) → stabil
  - tuned Quicksort (mit Optimierung)(in Klasse Array) → instabil

71) Sie können einen einfachen endlichen Automaten verstehen, aufzeichnen und in Java umsetzen

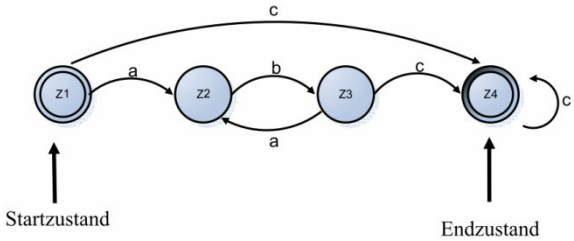
- Kaugummiautomaten sind Automaten, die bei Eingabe einer Münze einen Kaugummi herausgeben.
  - In der Informatik bezeichnet man als Automaten vorwiegend mathematische Modelle (…), die Zeichenfolgen verarbeiten und

dabei Antworten geben, wobei die Eingabe oft nur gelesen aber nicht verändert werden darf.

- Beispiel:
  - Der folgende endliche Automat akzeptiert alle Werte von folgender Struktur:
    - der erste Buchstabe ist ein a
    - danach folgen beliebig viele (auch 0-mal) b
    - danach folgt ein c



  - Der folgende endliche Automat akzeptiert alle Worte von folgender Struktur:
    - zu Beginn kommt eine beliebig lange Folge der Zeichenkette «ab» (auch 0-mal)
    - danach folgen beliebig viele c (mindestens 1)



72) Sie können die Ränder aller Teilworte eines Wortes bestimmen

- Der Rand eines Wortes ist die (maximale) Buchstabenfolge, die kürzer ist als das Wort selber und die sowohl am Anfang als auch am Ende des Wortes auftritt.

Ränder aller Teilworte des Musters „ababaa“

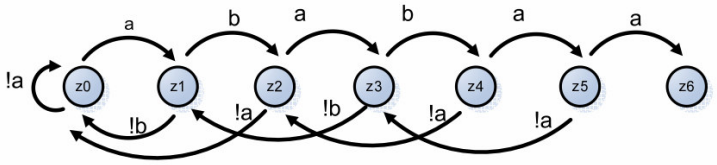
Teilwort	Rand	Länge des Randes
a	Ø	0
ab	Ø	0
aba	a	1
abab	ab	2
ababa	aba	3

73) Sie können einen "Musterautomaten" zu einem einfachen Muster aufzeichnen

- Algorithmus von einem Musterautomaten
  - Man überprüft jedes Zeichen des Textes genau einmal.
  - Wenn das Zeichen dem nächsten Zeichen des Musters entspricht, folgt man dem Zustandsübergang gemäss «Musterautomaten»

- Wenn das Zeichen nicht dem nächsten Zeichen des Musters entspricht, folgt man dem Zustandsübergang (!Zeichen) des «Musterautomaten»
- *Dies macht man so lange, bis man entweder am Anfangszustand (Z0) ist, oder bis der Wert des Zeichens einem Übergang entspricht, dem man dann folgt.*
- Danach überprüft man das nächste Zeichen

gegeben „Musterautomat“ für „ababaa“



gegeben Text:
a b a a b b a b a b a b a a b a b a a

74) Sie kennen den Aufwand und die Merkmale des KMP Suchalgorithmus, von Quicksearch und von Optimal Mismatch

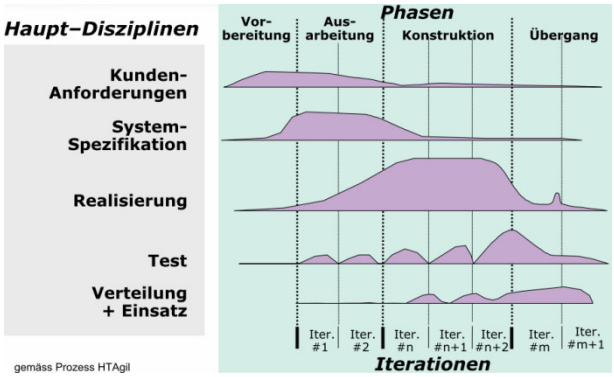
- KMP-Algorithmus (Knuth, Morris, Pratt)
  - Der KMP-Algorithmus beinhaltet 2 Schritte:
    - Entsprechend dem Pattern (Länge m) den Automaten generieren → Array next
    - Mit den Automaten bzw. mit Hilfe von next die Zeichenkette (Länge n) nach dem Pattern durchsuchen
  - Für den KMP-Algorithmus resultiert ein Aufwand von O(m + n), weil sich das Pattern und die Zeichenkette je rein sequentiell Zeichen für Zeichen abarbeiten lassen.
- Quicksearch
  - Quicksearch ermöglicht schnelles Durchsuchen einer Zeichenkette a nach einem Text-Pattern.
  - Ansatz: Durch die zu durchsuchende Zeichenkette nicht «kriechen» sondern «springen».
  - Bei Zeichenunstimmigkeit (Mismatch) zwischen dem Pattern und der Zeichenkette sofort das Zeichen unmittelbar nach dem Pattern analysieren.
  - min. tendiert gegen n/(m + 1)
- Optimal-Mismatch
  - Optimal-Mismatch basiert auf Quicksearch
  - Quicksearch besitzt die Eigenschaft, dass Zeichenvergleiche grundsätzlich in beliebiger Reihenfolge stattfinden können.
  - Die Optimierung basiert auf der Tatsache, dass die Zeichenhäufigkeit in den meisten Texten nicht gleich verteil ist.
  - z.B. im Deutschen:
    - Buchstabe e an erster Stelle mit 12 %
    - Buchstabe x an letzter Stelle mit 0.002 %

- Für das Vergleichen zieht man deshalb zuerst die seltenen Zeichen aus dem Pattern heran, so dass die Wahrscheinlichkeit für einem Mismatch bzw. für das Weiterspringen gross ist → Optimal-Mismatch
- Oft fehlen Statistiken zur Zeichenhäufigkeit. Dann sind andere Strategien gefragt:
  - Man reorganisiert das Pattern nach einem Mismatch so, dass das entsprechende Zeichen zukünftig als erstes verglichen wird. Die anderen Zeichen verschieben sich dann einfach um eine Stelle nach hinten.
  - Noch einfacher ist es, dieses Zeichen im Pattern nur um eine Stelle nach vorn zu vertauschen.
- Optimal-Mismatch besitzt theoretisch die gleichen Grenzen wie Quicksearch für die minimale und maximale Anzahl Vergleiche. Praktisch liefert Optimal-Mismatch aber noch schnellere Suchergebnisse.

Software Engineering:

75) Sie können die fünf Hauptdisziplinen des Software Engineering benennen

- Kunden-Anforderungen erfassen und beschreiben
- System-Spezifikationen
  - Anforderungen an Software und Umgebung spezifizieren, Architektur-Übersicht und Datenmodell erstellen, GUI-Skizzen entwerfen, Realisierbarkeit prüfen.
- Realisierung
  - Software schrittweise entwerfen, programmieren, testen und dokumentieren
  - Klassen-Design (vgl. Abstraktion, Interfaces, Abhängigkeiten)
  - Programmierung
  - Test (vgl. JUnit)
  - Dokumentieren (vgl. Javadoc)
- Test
  - Software schrittweise zusammenführen / integrieren und testen.
- Verteilung und Einsatz
  - Software verteilen /verfügbar machen und einsetzen.



76) Sie sehen die Notwendigkeit des Testens ein und können Gründe dafür benennen

- Testen = Systematisches Probieren nach verschiedenen Qualitätskriterien.
- Qualität: Übereinstimmung mit (formulierten) Anforderungen
  - nach Kriterien wie Funktionalität, Zweckdienlichkeit, Robustheit, Zuverlässigkeit, Sicherheit, Effizienz, Benutzbarkeit, Geschwindigkeit
- verschiedene Testmethoden
- Fehlerarten
  - Syntaxfehler sind Abweichungen von der Sprachsyntax, also von den grammatikalischen Regeln, die bestimmen, wie Symbole der Sprache korrekt zu benutzen sind (formale Fehler)
  - «Der Leser las dem Buch» `iff(i <! 10) {`

- Semantische Fehler (inhaltliche Fehler) sind bedeutungsmässige Unstimmigkeiten, die auf Kenntnisse über die Objekte beruhen. «Der Elefant ass die Erdnuss» vs. «Die Erdnuss ass den Elefanten»
- Logikfehler: Ein Programm kann syntaktisch einwandfrei sein und keine semantischen Unstimmigkeiten aufweisen, es beschreibt aber den gewünschten Ablauf nicht richtig.
- Qualitätssicherende Begleitmassnahmen:
  - Massnahmen, welche die Softwarequalität positiv beeinflussen, wie z.B. Reviews, definierter Entwicklungsprozess, Walkthrough
- Vertifikation:
  - formaler Nachweis von Eigenschaften von Programmen oder Programmteilen.
- Aussagekraft von Tests:
  - Nach einer erfolgreichen Vertifikation weiss man, dass ein Programm für alle möglichen Eingabedaten korrekt ist.
  - Nach einem Test weiss man nur, dass ein Programm für die getesteten Eingabedaten korrekt ist. Man möchte aber auf alle möglichen Eingabedaten schliessen.
  - Man kann immer nur das Vorhandensein von Fehlern zeigen; nie die Abwesenheit von Fehlern.
  - Testmengen müssen typisch und untypische Eingabewerte enthalten (z.B. 1, 2, 3, 4, 5 für ein Sortierprogramm, d.h. bereits sortiert)
  - Man bilde Gruppen von Eingabewerten, die gemeinsame Merkmale haben.

77) Sie kennen die Haupteigenschaften von Blackbox-Test, Whitebox-Test, Walkthrough und Debugger-Einsatz

- Blackbox-Testing
  - Programm als Blackbox
    d.h. undurchsichtig betrachtet. Zugänglich sind nur die Ein- und Ausgaben.
  - Frage: Erfüllt das Programm die Programmspezifikation?
  - Ausgehend von der Programmspezifikation wird vor dem Testen eine Testspezifikation erstellt mit nummerierten Testfällen.
  - Jeder Testfall hält für ganz bestimmte Programmeingaben die geforderten Programmausgaben fest.
  - Die Testfälle unterteilt man typisch in verschiedene Gruppen:
    - Normalfälle oder zulässige Wertebereiche
    - zulässige Spezialfälle oder Übergangsbereiche
    - unzulässige Spezialfälle oder unzulässige Wertebereiche

Programmspezifikation:

- Basierend auf den 3 Seitenlängen a, b und c eines Dreiecks soll die Dreiecksfläche berechnet werden.
- Die Seitenlängen werden als Gleitkommazahlen eingegeben.
- Negative Seitenlängen werden nicht verrechnet.
- Die maximale Seitenlänge beträgt 1'000.
- Die Fläche soll als ganze Zahl (konsequent aufgerundet) ausgegeben werden.
- Gegebenenfalls soll das Programm Hinweise ausgeben.

- **Testspezifikation**:

	Testfall	Eingabe	Ausgabe	bestanden?
Normalfälle:	Nr. 1	a = 4, b = 8, c = 10	16	
	Nr. 2	a = 10.0, b = 4.0, c = 8.0	16	
	Nr. 3	a = 12.3, b = 6.54, c = 7.89	24	
zulässige Spezialfälle:	Nr. 4	a = 6, b = 6, c = 6 (gleichseitiges Dreieck)	16	
	Nr. 5	a = 1000, b = 1E3, c = 1E3 (grösstmögliches Dreieck)	433013	
	Nr. 6	a = 2, b = 8, c = 10 (flachgedrücktes Dreieck)	0	
unzulässige Spezialfälle	Nr. 7	a = -4, b = 8, c = 10	Seite < 0!	
	Nr. 8	a = 1, b = 1, c = 10	kein Dreieck!	

- Whitebox-Testing
  - Das Programm wird als Whitebox bzw. als durchsichtig betrachtet, d.h. der interne Programmcode wird zusätzlich herangezogen.
  - Datenorientierte Abdeckung:
    - Verwendungsüberdeckung:
      - Jede definierte Variable wird durch einen Testfall erfasst
    - Wertebereichsüberdeckung:
      - Für jede definierte Variable werden alle Randwerte durch einen Testfall erfasst.
    - jede Variable wird vor der ersten Verwendung initialisiert
  - Code Abdeckung
    - Auf Grund des Programmcodes wird die Testspezifikation so definiert, dass sämtliche Programmpfade mindestens ein Mal zur Ausführung gelangen. (Code, Coverage, Code Abdeckung, zum Teil mit Tool Unterstützung)
- Walkthrough
  - Möglichst eine «fremde» Person (nicht der Programmierer) studiert bzw. folgt dem Programmcode (normalerweise nicht bis zum tiefsten Methodenaufruf), angeleitet durch den Programmierer.
- Debugger
  - Hilfsmittel der Entwicklungsumgebung dass einen «Walkthrough in Natura» ermöglicht, z.B. «Step by Step» oder «go to next Break-point» sowie das Beobachten der Variableninhalte.
- Zusatz: Testorganisation:
  - Testen basiert auf Anforderungen.
  - Erstellung eines detaillierten Dokumentes für:
    - Testziele: Welche Qualitätskriterien sind einzuhalten?
    - Teststufen: Welche Aktivitäten in welchen Projektphasen?
    - Testmethoden: Welche Tests mit welchen Werkzeugen?
    - Testspezifikationen
    - Randbedingungen: Hardware / Softwareumgebung
    - Rollen und Verantwortlichkeiten
    - Meilensteine und Deliverables

- testfreundliches Programmieren:
  - Unterteilung des Programms in Module / Units, die einzeln getestet werden
  - Standardisierung von Schnittstellen
  - Einhaltung von Programmierrichtlinien
  - Verzicht auf untypische, nicht standardisierte Elemente / Möglichkeiten / Nebeneffekte der Programmiersprache
  - Schon beim Programmieren ans Testen denken

78) Sie können auf Grund einer einfachen Programmspezifikation eine Testspezifikation erstellen

- Beschreibt die im Testplan genannten Vorgehensweisen im Detail

79) Sie können Ihren Code gemäss den Richtlinien von javadoc unter Verwendung von javadoc Tags kommentieren

- Unterstützt Programmierer bei der Dokumentation ihrer Klasse
- für die Dokumentation des nach aussen sichtbaren «Interfaces» (was), nicht der Implementation (wie)
- Das Javadoc Tool verarbeitet die Deklarationen und javadoc Kommentare (/**…*/) in Java Source Code Dateien und erzeugt eine Menge von html-Dateien. Damit kann eine Programm-Dokumentation erzeugt werden.
- In den Kommentaren werden bestimmte Schlüsselwörter verwendet. Diese beginnen mit @:
  - @author
  - @version
  - @param
  - @return

80) Sie können die drei Vorgehensschritte beim Refactoring beschreiben

- Ein gutes Klassen-Design kann nie alle Eventualitäten berücksichtigen
- Ein gutes Klassen-Design ist im Hinblick auf eine Erweiterung unter Umständen nicht mehr gut.
- Refactoring ist wichtig
  - Also das Anpassen des Design / Codes an neue Anforderungen und Gelegenheiten.
- Dies bedingt häufig das Aufspalten von Klassen und Methoden in mehrere.
- Die Wahrscheinlichkeit von Fehlern ist gross
- Vorgehen:
  - Falls der bestehende Code noch nicht getestet wurde: Testspezifikationen schreiben und den Code testen.
  - Code redesignen ohne funktionale Änderungen und Code erneut testen (analog oben).
  - Code erweitern mit funktionalen Änderungen und Code erneut entsprechend testen.