

Java in 11. Klasse – 2. Halbjahr

<http://worgtsone.scienceontheweb.net/worgtsone/> - [mailto: worgtsone @ hush.com](mailto:worgtsone@hush.com)

Wed Mar 11 00:22:00 CET 2009 – 13. Oktober 2011

Inhaltsverzeichnis

1 Objektorientierte Programmierung (OOP)	4
2 Lexikon	5
3 Klassen	6
3.1 Schreibweise	6
3.2 Beispiel Schüler. Mit Konstruktor.	6
4 Objekte	7
5 Meine erste, sinnlose Klasse	8
5.1 Autopsie	9
5.2 Frankenstein	9
6 Selbstgeschriebene Klassen	10
6.1 Regeln, mit und ohne Cache	10
6.2 Quadrat	10
6.3 Kreis, Kugel und Würfel	10
6.4 Rechteck	10
6.5 Auto01	10
6.6 Auto02	10
6.7 Auto03	11
7 Vom Klonen oder Copy-Konstruktor	12
8 Böartiger Test über OOP	13
9 Modifier	14
9.1 Static Methoden in selbstgebastelten Klassen	15
9.2 Übung : worgtsone's Helferlein : Q	15
10 MVGSÜ - (BIC ?)	16
10.1 Modellierung	16
10.2 Vererbung	16
10.3 Generalisierung	17
10.4 Spezialisierung	17
10.5 Überladen	17
10.6 Arbeitsblatt Klassen	18
11 Vordefinierte Klasse String	19
11.1 String-Übung	20

12 Vordefinierte Klasse Frame	21
12.1 Quelltext <code>Frame1</code>	21
12.2 Fragen zum Quelltext	22
12.3 Lesen der Sun-Dokumentation	22
12.4 Übung VierFenster	22
12.5 Optional: Knöpfe (Für Neugierige)	23
13 Rekursion	24
13.1 Beispiel Fakultät	24
13.2 Beispiel Fibonacci	24
13.3 Beispiel Türme von Hanoi	24
14 Rekursion2 mit Sortierverfahren (optional)	25
14.1 MergeSort	25
14.2 QuickSort	25
15 Rekursive Grafik (Fraktale) (optional)	26
15.1 Use the Source, Luke.	26
15.2 Vorschläge für Übung	29
16 Kommandozeilenparameter	30
16.1 Übung	30
17 Klasse <code>System</code> – verschoben nach St. Nimmerlein	31
18 Interfaces und Adapterklassen – 2 be done – verschoben nach St. Nimmerlein	31
19 2- und 3-dim Arrays, Enums, Hashes, Trees, Sets und Maps – 2 be done (optional) – verschoben nach St. Nimmerlein	31
20 Dateien lesen und schreiben	32
20.1 So gehts:	32
20.2 Nach Zahlen	32
20.3 Übung : Erzeugen	32
20.4 <code>copy</code>	32
20.5 Übung : Sortieren	32
21 TODO : Arbeitsblatt – Streams am Beispiel von <code>Tastatur.java</code>	33
21.1 Byte-Based Input	33
21.2 Byte-Based Output	33
21.3 Puffern	33
21.4 Unicode-Basierte Ein- und Ausgabe	33
21.5 Konvertieren zwischen Unicode und <code>Bytestream</code>	33
22 Ein simpler Taschenrechner (optional)	34
23 Projekte	37
23.1 Rationale Zahlen (Brüche)	37
23.2 Anmerkungen	37
23.3 Menüprogramm <code>Layoutdemo</code>	37
23.4 Freier Fall	37
23.5 Pendel und Doppelpendel	37
23.6 Mandelbrot-Mengen	37
23.7 Schülerverwaltung	37
23.8 <code>TaschenRechner</code>	37

Disclaimer

Wissen ist zum Teilen da. Ich teile mein Wissen mit Ihnen, lieber Kollege.
Ich bin aber nicht perfekt. Unter worgtsone@hush.com
nehme ich dankbar Ihre Verbesserungsvorschläge entgegen.

*

Legal Blurb: Alle Informationen in diesem Dokument sind falsch, unvollständig,
irreführend, irrelevant und / oder funktionieren einfach nicht.

Wenn Sie es trotzdem benutzen, und es geht dabei etwas kaputt, ist das Ihr
Problem, nicht meins.

*

Bitte teilen Sie meine Web-Adresse nicht Ihren Schülern mit.

1 Objektorientierte Programmierung (OOP)

Moderne Programme bilden Objekte der realen Welt (Zahlen, Schüler, Filme, ...) ab. Sie lassen sich besser verwalten und einfacher handhaben.

Unsere bisherigen Datenstrukturen schaffen das nur eingeschränkt und treten sich mit ihren Funktionen bisweilen gegenseitig auf die Zehen.

Vorteile der OOP:

- **Alles, was ein Objekt hat und kann, wandert in seine Klasse.**
Was hat es? Nur das Wissen über sich selbst. Gerade genug, daß es funktioniert.
Was kann es? Nur das Können, was es braucht.
- Die Klasse bietet eine einheitliche Schnittstelle nach außen.
- Falls Änderungen erforderlich werden, werden sie in der Klasse gemacht, und fertig.
- Gut geschriebene und dokumentierte Klassen können beliebig oft wiederverwendet werden.
- Klassen können erweitert und modifiziert werden, indem man sie beerbt.

Das Haben und das Können sitzen zur Laufzeit im RAM.

Das Haben (Vorname etc.) ist bei jedem Objekt anders und muß für jedes Objekt einzeln verwaltet werden.

Das Können ist für alle Objekte einer Klasse gleich und sitzt nur einmal im RAM.

2 Lexikon

Wir fangen an dieser Stelle an, ein Lexikon zu schreiben. Das ist eine leere Seite, oben rechts kommen Initialen und Datum, oben links steht LEXIKON, und ansonsten kommen nur die wichtigsten Begriffe und Erklärungen drauf.

3 Klassen

Eine Klasse ist...

- eine Sammlung von Daten und Methoden.
- eine Idee, wie ein Ding funktioniert, was es kann und was es dazu braucht.
- eine Konstruktionszeichnung, ein Bauplan, eine Gußform für ein oder viele Dinge derselben Art.

Daten und Methoden bilden zusammen die Mitglieder (Member) einer Klasse.

3.1 Schreibweise

- Schreiben Sie Klassennamen groß.
- Schreiben Sie Objekte, Daten und Methoden klein.
- Schreiben Sie bei einem neuen Wort einen Großbuchstaben.

3.2 Beispiel Schüler. Mit Konstruktor.

Schreiben Sie einige Sachen auf, die ein Schüler über sich weiß, und einige Sachen, die er kann. Verwenden Sie UML-ähnliche Notation, d.h. ein dreigeteilter Kasten: oben Klassename, mitten Methodennamen(), unten Attribute.

Achtung : Basteln Sie auch etwas, das einen neuen Schüler anlegen und auf Plausibilität prüfen kann.

Lösung:

```
Schueler
-----
String vorname
String nachname
Date geburtsdatum
String handy
String strasseNr
-----
void kichern()
void schwaetzen()
void mitPapierRascheln()
void zuSpaetKommen (Minutes m)
Note unterrichtFolgen()
```

4 Objekte

Ein Objekt ist eine Instanz einer Klasse. Jedes Objekt gehört zu einer Klasse.
Eine Klasse kann 0 bis viele Objekte haben.
Ein Objekt funktioniert nach genau den Ideen, die in seiner Klasse festgelegt sind.

Ein Objekt kann seinen eigenen Satz Eigenschaften haben.

5 Meine erste, sinnlose Klasse

```
class Mek {  
  
    // Attribute  
    int i;  
    String name;  
  
    // Konstruktor  
    Mek (String name) {  
        this.name = name;  
    }  
  
    // noch ne Methode  
    public void sprich () {  
        System.out.println ("Ich bin " + name + " und sitze in " + this);  
    }  
  
    // main - gehört eigentlich nach MekErzeuger.java  
    public static void main (String[] args) {  
        Mek m1 = new Mek ("m1");  
        Mek m2 = new Mek ("m2");  
        Mek mIrrerName = new Mek ("Langer Name.");  
        m2.sprich ();  
    }  
}
```

5.1 Autopsie

Schreiben Sie die richtigen Nummern an die richtige Stelle im Quelltext.

1. ist der Beginn der Klassenbeschreibung.
2. ist ein Kommentar.
3. instanziert ein Objekt namens `mIrrerName` der Klasse
4. Wer ein neues Objekt haben will, bestellt es mit dem dreibuchstabigen Schlüsselwort
5. Korrekte Klassen haben mindestens einen Konstruktor.
6. Methoden haben Klammern, wo Übergabeparameter drin sein können.
7. Methoden haben einen Rückgabetyt. Falls sie nichts zurückgeben, haben sie den Rückgabetyt
8. Konstruktoren dürfen Parameter übernehmen.
9. Diese Methode macht die Klasse als Programm ausführbar.
10. Diese Methode macht die Klasse geschwätzig.
11. `this` zeigt auf die Adresse, an der DIESES Objekt sitzt. Oder: `this` zeigt auf das aktuelle Objekt.
12. Konstruktoren kann man überladen.
13. Klassenmethoden kann man `idR`¹ nicht direkt aufrufen², sondern man muß zunächst einieren und kann dann erst eine der dieses aufrufen.
14. Objektmethoden ruft man so auf:

5.2 Frankenstein

Basteln Sie die Klasse um, so daß sie

1. `MeineErsteSinnloseKlasse` heißt.
2. 4 Objekte instanziert und jedes 1mal sprechen läßt.
3. mitzählt, wie oft sie gesprochen hat. Benutzen Sie dabei die bereits angelegte Variable `i`.
4. 1000mal ein zufällig gewähltes Objekt sprechen läßt.
5. Lassen Sie Objekt `m4` 4mal sprechen, ändern Sie seinen Namen, und lassen Sie es erneut sprechen.
6. Setzen Sie seinen Namen auf `private`, und versuchen Sie's nochmal.
7. Schreiben Sie `get/set` Methoden für `name` und `i`.
8. Schreiben Sie sie so um, daß in `main` nur ein Konstruktor aufgerufen wird.
9. Schreiben Sie eine `Gott`-Klasse, die `Mek`-Objekte erzeugt und sprechen läßt.

¹*Seufz* bedeutet: in der Regel

²Ich weiß - in der Klasse `Tastatur` kann man das dennoch tun. Das ist aber die Ausnahme und nicht die Regel.

6 Selbstgeschriebene Klassen

6.1 Regeln, mit und ohne Cache

- Du sollst so wenig Variablen wie möglich benutzen. Du sollst die Ausgabegrößen dann berechnen lassen, wenn sie gebraucht werden.
- Es gibt keinen Kreis mit Durchmesser 9 und Radius 9. Wenn der Durchmesser 9 ist, muß der Radius 4,5 sein, es geht nicht anders.
- Wenn du sehr oft set-Methoden aufrufst und selten get-Methoden, brauchst du keinen Cache.
- Aber wenn auf ein setDurchmesser() eine Million getDurchmesser() kommen, ist häufiges Berechnen unfug. Bestimme in diesem Fall eine weitere Variable: durchmesser.
- Ändere in setRadius() dann den Durchmesser gleich mit. Denn wenn sich der Radius ändert, ändert sich bei einem richtigen Kreis der Durchmesser ebenfalls gleich mit.

6.2 Quadrat

k. setK, setFlaeche, setDiagonale. getK, getFlaeche, getDiagonale.

6.3 Kreis, Kugel und Würfel

Schreiben Sie sie so, daß Sie Kreise anhand des Radius ODER Durchmesser ODER Fläche ODER Umfang erzeugen können.

6.4 Rechteck

Laenge und Breite. setLaenge(), setBreite(), setSeitenverhaeltnis() (dabei bleibt Flaeche gleich), setFlaeche() (dabei bleibt Breite gleich) und alle get-Methoden, zusammengefaßt in sprich().

6.5 Auto01

Modellieren Sie : Ein Auto hat maker, model, einen Verbrauch (default: 7,2l/100km), einen Tankinhalt (default: 50l) und einen Kilometerstand (default : 0km). Es kann fahren(double km), tanken(immer voll), sagTankinhalt(), getTankinhalt() und sprich() (alle Betriebsdaten ausgeben).

Wenn der Sprit alle ist, bleibt es liegen. Dann tankt es und fährt weiter.

Es kann zählen, wie oft es liegengeblieben ist. Dazu brauchen Sie die Methode reichweite().

Basteln Sie einen VW Käfer und einen Mercedes C111-2. Führen Sie an jedem Auto zufallsgesteuert 999 Methodenaufrufe durch.

6.6 Auto02

Kopieren Sie Auto01 nach Auto02, und fügen Sie hinzu:

tanken() elegant kopieren nach volltanken(). tanken (int liter) hinzufügen.

Eingegebene Kilometer dürfen höchstens gleich der Reichweite sein – sonst fährt das Auto seinen Tank leer und bleibt dann liegen.

Getankte Liter dürfen das verfügbare Volumen nicht überschreiten – sonst läufsts über, und dann dürfen Sie schimpfen und aufwischen.

6.7 Auto03

Hinzufügen: schiebedach(). Default=zu.

Verbrauch: falls $km > 100$, fahren wir über Land, mit 6,1l/100km; andernfalls in der Stadt mit 9,8l/100km.

7 Vom Klonen oder Copy-Konstruktor

Gelegentlich wollen wir ein Objekt (zB ein Rechteck) klonen, d.h. ein vorhandenes kopieren. Das neue soll dann, unabhängig vom Original, selbständig existieren.

So geht's nicht:

```
...
Rechteck r1 = new Rechteck(3,3);
Rechteck r2 = r1;
r1.setBreite=99;
r2.sprich();
r2.setBreite=1;
r1.sprich();
...
```

... denn die beiden Rechtecke sitzen am selben Speicherplatz (Ausgabe von this) und sind daher dooferweise identisch.

Man kann der Rechteck-erschaffenden Klasse beibringen, wie man ordentlich klonet:

```
...
Rechteck r1 = new Rechteck(3,3);
Rechteck r2 = new Rechteck(r1.getBreite(), r1.getLaenge());
...
```

... und das funktioniert ja auch ...

... aber es ist unbefriedigend, denn ein Rechteck sollte selber wissen, wie man sich klonet, und dazu keine Hilfe von seinem Schöpfer brauchen.

Also schreiben wir einen Konstruktor in die Klasse Rechteck, dem wir das Original als Parameter mitgeben, und hoffen, von dem eine selbständige Kopie zu erhalten ...

```
...
Rechteck (Rechteck orig){           // achtung, funktioniert nicht!!!
    double origL = orig.getLaenge();
    double origB = orig.getBreite();
    return Rechteck (origL, origB);
}
...
```

... und siehe da, es kompiliert nicht. Der Compiler sagt :

1. Falls du innerhalb dieses Konstruktors was konstruieren willst, benutze this.
2. Falls du this benutzt, muß es die erste Zeile sein.

Mit diesen Änderungen ...

```
...
Rechteck (Rechteck orig){           // aber der funktioniert.
    this (orig.getLaenge(), orig.getBreite());
}
...
```

... funktioniert das dann auch, und nun können wir klonen.

Aufgabe Klonen Sie Rechtecke, Kreise, Würfel und Quadrate, und erfreuen Sie sich am Output.

8 Bösertiger Test über OOP

Wenn es richtig ist, mach einen Haken dran. Wenn nicht, KORRIGIERE ES.

1. Mit OOP kann ich prima Schüler, Filme, Werkzeugmaschinen und Gebäude verwalten.
2. Alles, was ein Objekt hat und kann, wandert in seine Klasse.
3. Es hat nur das Wissen über sich selbst. Wird genannt:
4. Es kann nur das Können, was es braucht. Wird genannt:
5. Das Wissen ist für jedes Objekt anders und belegt verschiedenste Speicherplätze.
6. Das Können ist für jedes Objekt derselben Klasse gleich und belegt verschiedenste Speicherplätze.
7. Auf meine Methoden greife ich mit get- und set-Methoden zu.
8. In die Get-Methode kann ich Gültigkeits- und Plausibilitätsprüfungen einbauen.
9. Falls ich keinen Lesezugriff auf mein Wissen haben will,
10. Falls ich keinen Schreibzugriff auf mein Wissen haben will,
11. Falls Änderungen erforderlich werden, werden sie in der Klasse gemacht, und fertig.
12. Gut geschriebene und dokumentierte Klassen können beliebig oft wiederverwendet werden.
13. Von jeder Klasse kann ich bis 5 Instanzen bestellen. Sie unterscheiden sich in den Methoden, aber haben die gleichen
14. Eine Klasse ist eine Sammlung von Daten und Methoden.
15. Eine Klasse ist eine Idee, wie ein Ding funktioniert, was es kann und was es dazu braucht.
16. Wenn ich Klassen entwerfe, male ich einen dreigeteilten Kasten:
17.
18.
19.
20. get-Methoden sind immer `void`.
21. set-Methoden sind immer `void`.
22. sag-Methoden sind immer `void`.
23. sag-Methoden sind eine Erfindung von `worgt.sone`.
24. Konstruktoren dürfen Parameter übernehmen.
25. macht die Klasse als Programm ausführbar.
26. Wenn ich nicht will, daß die Klasse ausführbar ist,
27. Diese Methode macht die Klasse gesprächig.
28. `this` zeigt auf die Adresse, an der DIESES Objekt sitzt. Oder: `this` zeigt auf das aktuelle Objekt.

9 Modifier

Wir schreiben seit der ersten Stunde Modifier vor unsere Klassen, Attribute und Methoden. Hier steht, warum.

---NIX--- class	default = package-private.
abstract class	enthält unimplementierte Methoden. Kann nicht instanziiert werden.
abstract class	enthält unimplementierte Methoden. Kann nicht instanziiert werden.
abstract interface	Alle Schnittstellen (interfaces) sind abstract.
abstract method	Methode ohne Implementierung, in abstrakter Klasse. Sie hat nur einen Namen und Datentypen und Zugriffsrechte, Java bastelt daraus die Signatur.
final class	Keine Vererbung möglich.
final method	Kann nicht überschrieben werden. Kann nicht dynamisch gesucht werden.
final attribut	Kann nicht geändert werden, ist konstant.
final parameter in Parameterliste einer Methode (= Argumente)	Kann von der Methode nicht geändert werden.
private member (Methode oder Attribut)	Ist nur in seiner Basisklasse zugänglich.
protected member (Methode oder Attribut)	Ist nur in seinem Package (???) und seinen Unterklassen ("Kindern") zugänglich.
public class	Überall zugänglich.
public interface	Überall zugänglich.
public member (Attribute oder Methoden)	Überall zugänglich, solange die Basisklasse vorhanden ist.
static class	Wird Top-Level Klasse, wens eine innere Klasse ist. (???)
static method	Ist eine Klassenmethode. Aufruf: <code>Klassenname.method()</code> . Siehe unten.
static attribute	Ist ein Klassenattribut. Aufruf: <code>Klassenname.attributname</code> . Existiert, sobald die Klasse geladen ist. Existiert nur einmal, egal wie viele Instanzen der Klasse existieren. Anwendung: ZB als Zähler, wieviele Objekte einer Klasse konstruiert wurden.
synchronized	(engl.: synchronisiert)
transient	(engl.: vorübergehend; dem Zeitverlauf folgend)
volatile	(engl.: flüchtig)

9.1 Static Methoden in selbstgebastelten Klassen

```
class MyStaticClass {           // no main method
    public static int add (int a, int b){ return (a+b);
}}

class MyStaticClassTest {
    public static void main (String[] args){
        System.out.println( "17+4= " + MyStaticClass.add(17, 2*2));
    }
}
===== gibt : =====
17+4= 21
```

9.2 Übung : worgtsone's Helferlein : Q

Basteln Sie Klasse Q mit den Methoden (überlegen Sie die Rückgabe-Datentypen!)

- `p(String)` : macht `System.out.println`,
- `umlauteAn()` : schaltet im Windos-Konsolenfenster korrekte deutsche Umlaute an,
- `longfakt(int)` : liefert Fakultäten in long. Liefert -1 bei bösen Aufrufen.
- `longdouble(int)` : liefert Fakultäten in double. Liefert -1 bei bösen Aufrufen.
- `s` : ein String, bestehend aus einem einzigen Leerzeichen.

10 MVGSÜ - (BIC ?)

10.1 Modellierung

Modellierung ist, wenn man sich hinsetzt und überlegt, welche Klassen mit welchen Beziehungen man nehmen möchte.

10.2 Vererbung

Klassen schreiben ohne auf vorhandene Klassen zurückzugreifen ist Unfug und Verschwendung. Erfinden Sie das Rad nicht neu. Auch Nabe, Speichen, Felgen und Reifen nicht. Normalerweise verwenden wir in unseren Programmen die höchstentwickelten, schlauesten Klassen. Diese haben bereits viel geerbt und sind durch eigene Methoden (Know-How) noch schlauer geworden.

Sie stehen normalerweise ganz unten in der Objekthierarchie. Die Objekthierarchie (Vererbungsbeziehungen) werden normalerweise als ein nach unten wachsender Baum dargestellt, so wie ein Verzeichnisbaum auf dem Rechner.

In Java heißt die oberste Klasse `Object`.

**Vererbung ist, wenn eine Klasse alle Variablen und Methoden von einer anderen Klasse übernimmt und ggf. noch eigene hinzufügt.
In der OOP können nur Klassen von Klassen erben.**

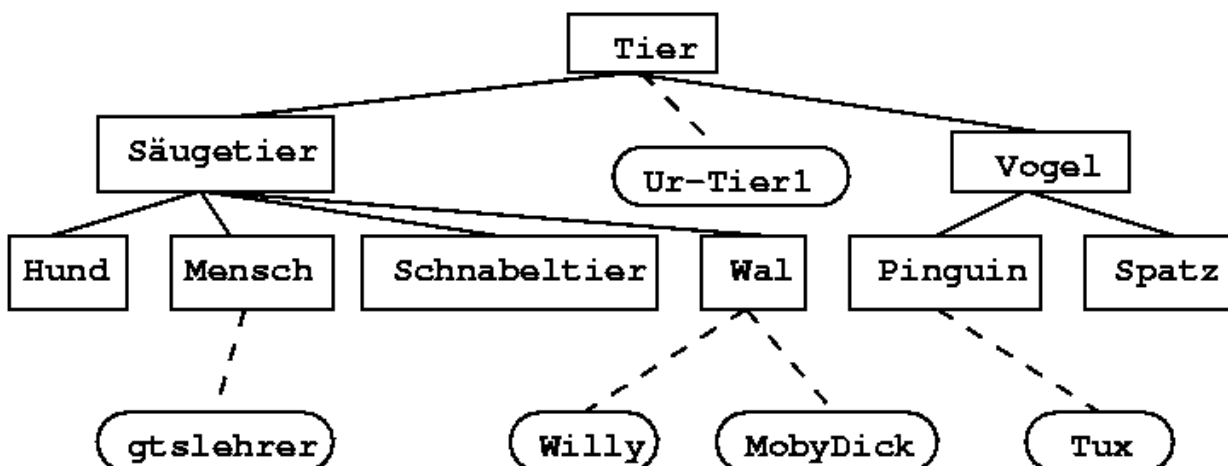
Die erbende Klasse heißt Unterklasse (subclass). Die vererbende Klasse heißt Oberklasse (super class).

Vererbung kann man mit folgenden Worten ausdrücken: "(Unterklasse) ist ein Spezialfall von (Oberklasse)." oder (einfacher): "(Unterklasse) ist ein (Oberklasse)."

Vererbung ist **nicht**, wenn aus der Zusammenarbeit zweier Hunde ein neuer Hund entsteht. Das Ergebnis ist bloß ein weiteres Objekt, ebenfalls der Klasse Hund.

Beispiel Tier: kann geboren werden, atmen, sterben.

Hund: kann alles, was Tier kann. Zudem: einen Namen haben, bellen und beißen.



Vererbung sollte nur eingesetzt werden, wenn sie die anstehenden Aufgaben erleichtert. Vererbung mit mehr als drei Hierarchieebenen gilt als unübersichtlich, da man zum Verstehen einer Unterklasse stets in ihre Oberklassen schauen muß.

Achtung: Vererbung wird sehr gern als Baumdiagramm gemalt. Nehmen Sie für ihre Klassen (zB Affe) Kästen. Wenn Sie noch Objekte dazunehmen wollen (zB Äffin Lisa), nehmen Sie Kreise oder Kästen mit runden Ecken oder so.

10.3 Generalisierung

Bei der Generalisierung wird untersucht, ob mehrere Klassen eine gemeinsame Oberklasse haben.

Zweck: wenn einige Attribute und/oder Methoden sehr ähnlich oder gar gleich sind, modelliert man sie einmal in der Oberklasse – statt zB dreimal in den Unterklassen.

Beispiel Schulleiter, Hausmeister und Schüler sind Menschen.

10.4 Spezialisierung

Spezialisierung ist das Gegenteil von Generalisierung: wenn man vom Allgemeinen zum Spezialisieren, von der Oberklasse zur Unterklasse kommt.

Beispiel Ein Schulleiter ist ein Mensch - aber ein spezieller.

10.5 Überladen

Überladung ist, wenn eine Methode mit *demselben Namen*, aber *verschiedenen Parameterlisten* (incl. leerer Liste) vorkommt.

10.6 Arbeitsblatt Klassen

Notieren Sie, welcher Satzanfang zu welchem Satzende gehört. Mehrfachnennungen möglich.

1. Eine Klasse ...
2. Ein Objekt ...
3. Eine Objekthierarchie ...
4. Vererbung ...
5. Spezialisierung ...
6. Generalisierung ...
7. Abstraktion ...
8. Modellierung ...
9. Instanziierung ...

-
1. ist eine Instanz einer Klasse.
 2. ist z.B., wenn ich die Klassen `Knopf` und `CheckBox` von der Klasse `ClickableObject` ableite, denn beide müssen Mausklicks empfangen und verarbeiten.
 3. ist eine Sammlung von Daten und Methoden.
 4. ist, wenn Klassen von schlauen Klassen erben und durch zusätzliche Mitglieder noch schlauer werden.
 5. erfordert das Aufsetzen einiger neuer `get-` und `set-`Methoden.
 6. ist, wenn neue, sehr schlaue Klassen nicht mehr zu allgemeineren Aufgaben eingesetzt werden können.
 7. ist, wenn man sich Richtung Objekt-Hierarchie-Wurzel bewegt.
 8. wirft die Frage auf: Was soll welche Klasse über sich selbst wissen?
 9. ist das Entwerfen neuer Klassen.
 10. ist, wenn man sich verzweifelt Oberklassen wünscht.
 11. ist z.B., wenn ich nach einem Bauplan 5 gleiche Wohnhäuser in der Seestr. 13-17 baue.
 12. ist eine Art Schaltplan und zeigt, wie ein Ding aufgebaut ist.
 13. ist z.B., wenn ich nach einem Bauplan 6 Doppelhaushälften baue: eine mit Eckturm, eine mit Flachdach, vier mit Doppelgarage, zwei mit Tiefgarage.
 14. ist, wenn man sich von der Objekt-Hierarchie-Wurzel wegbewegt.
 15. ist das schwierige Herausfinden der Member einer neu zu erstellenden Klasse.
 16. ist eine Art Verhaltensmuster und zeigt, was ein Ding kann.
 17. definiert sich durch ihre Mitglieder (Attribute und Methoden).
 18. ist z.B., wenn ich die Klassen `Affe`, `Mensch` und `Wal` von der Klasse `Saeugetier` ableite, denn alle Säugetiere können atmen, essen und sich Geschlechtlich Fortpflanzen.

11 Vordefinierte Klasse String

Wir werden die Klasse String untersuchen.

1. Suchen Sie die Datei `String.class` und schauen Sie sie in einem Hex-Editor an.
2. Suchen Sie Java-Dokumentation auf Ihrem Rechner.
3. Schauen Sie in Ihr Java-Buch.
4. Besuchen Sie `google` mit den Suchworten `java doc String`.
5. Was ist der Unterschied zwischen ASCII, ISO-8859-1, utf-8 und utf-16?
6. Was ist der Unterschied zwischen `s1 == s2` und `s1.equals(s2)` ?
7. Schauen Sie sich die Liste an, finden Sie die wichtigsten Methoden und finden Sie heraus, wie sie funktionieren.

```
public final class String
    public String() {
    public String(String original) {
    public String(char value[]) {
    public String(char value[], int offset, int count) {
    public String(byte ascii[], int hibyte, int offset, int count) {
    public String(byte ascii[], int hibyte) {
    public String(byte bytes[], int offset, int length, String charsetName)
    public String(byte bytes[], String charsetName)
    public String(byte bytes[], int offset, int length) {
    public String(byte bytes[]) {
    public String (StringBuffer buffer) {
    public int length() {
    public char charAt(int index) {
    public void getChars(int srcBegin, int srcEnd, char dst[], int dstBegin) {
    public void getBytes(int srcBegin, int srcEnd, byte dst[], int dstBegin) {
    public byte[] getBytes(String charsetName)
    public byte[] getBytes() {
    public boolean equals(Object anObject) {
    public boolean contentEquals(StringBuffer sb) {
    public boolean equalsIgnoreCase(String anotherString) {
    public int compareTo(String anotherString) {
    public int compareTo(Object o) {
    public int compareToIgnoreCase(String str) {
    public boolean startsWith(String prefix, int toffset) {
    public boolean startsWith(String prefix) {
    public boolean endsWith(String suffix) {
    public int hashCode() {
    public int indexOf(int ch) {
    public int indexOf(int ch, int fromIndex) {
    public int lastIndexOf(int ch) {
    public int lastIndexOf(int ch, int fromIndex) {
    public int indexOf(String str) {
    public int indexOf(String str, int fromIndex) {
    public int lastIndexOf(String str) {
    public int lastIndexOf(String str, int fromIndex) {
    public String substring(int beginIndex) {
    public String substring(int beginIndex, int endIndex) {
    public CharSequence subSequence(int beginIndex, int endIndex) {
    public String concat(String str) {
    public String replace(char oldChar, char newChar) {
```

```

public boolean matches(String regex) {
public String replaceFirst(String regex, String replacement) {
public String replaceAll(String regex, String replacement) {
public String[] split(String regex, int limit) {
public String[] split(String regex) {
public String toLowerCase(Locale locale) {
public String toLowerCase() {
public String toUpperCase(Locale locale) {
public String toUpperCase() {
public String trim() {
public String toString() {
public char[] toCharArray() {
public static String valueOf(Object obj) {
public static String valueOf(char data[]) {
public static String valueOf(char data[], int offset, int count) {
public static String copyValueOf(char data[], int offset, int count) {
public static String copyValueOf(char data[]) {
public static String valueOf(boolean b) {
public static String valueOf(char c) {
public static String valueOf(int i) {
public static String valueOf(long l) {
public static String valueOf(float f) {
public static String valueOf(double d) {
public native String intern();

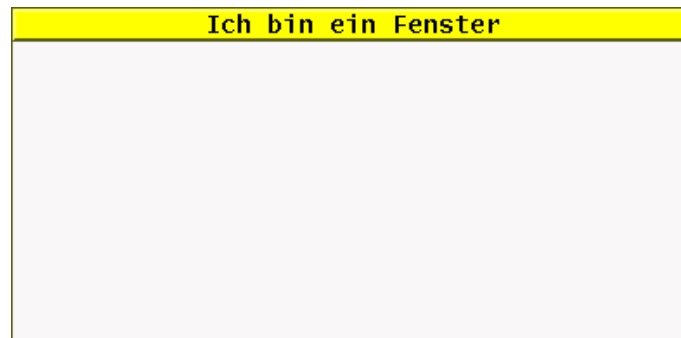
```

11.1 String-Übung

Schreiben Sie ein Programm, das erzeugt und ausgibt:

1. Einen String `s` mit dem Inhalt "nichts";
2. `s` mit dem Inhalt "";
3. `s` mit dem Inhalt "äörüß gtsLEHrer ";
4. führende und folgende Leerzeichen vorübergehend entfernt;
5. zu Kleinbuchstaben konvertiert und alle `r` durch `.` ersetzt;
6. den String " \nkekse\n " anhängt;
7. die Position von "tsL" und "XXX" findet;
8. die Position des letzten `r` findet;
9. den Hash-Code ausgibt;
10. untersucht, ob `s` mit 3 Leerzeichen anfängt;
11. `s` mit "kekse" vergleicht;
12. die Länge von `s` und von `s.trim()` ausgibt.
13. die Strings "int", "integer" und "Integer" sortiert.

12 Vordefinierte Klasse Frame



Schreiben Sie Sachen und Datentypen auf, die jedes Fenster über sich weiß. Schreiben Sie auf, was es kann, und die Parameter, die es dazu braucht.

Hinweis: Eine *dimension* ist ein Paar aus zwei Integer-Zahlen.

Lösung:

- String Titel, Dimension Position, Dimension Größe, bool resizable;
- void setTitle(String neuerTitel),
- String getTitle(),
- void setPosition(Dimension d),
- get / set Groesse(Dimension d),
- int getFlaecheInPixel(),
- Dimension getRechteUntereEcke(),
- neuesFenster(Dimension position, Dimension groesse, bool resizable).

12.1 Quelltext Frame1

```
import java.awt.*;

public class Frame1 {

    public static void main (String args[])    {
        Frame f1 = new Frame ();
        f1.setVisible (true);
        f1.setTitle ("Ich bin ein Fenster");
        f1.setSize (400, 200);
        f1.setLocation (1, 1);
        int i = 0;
        while (true) {
            i++;
            int x = (int) (200 + 100 * Math.cos (i / 99.0));
            int y = (int) (200 + 100 * Math.sin (i / 99.0));
            f1.setLocation (x, y);
            try { Thread.sleep (10); } catch (Exception e) { }
        }
    }
}
```

12.2 Fragen zum Quelltext

Lücken ausfüllen, ergänzen, streichen, die Fragennummern in den Quelltext schreiben. Bis alles richtig ist.

1. Die Klasse `Frame` sitzt im Abstract Window Toolkit (Abk.: _____) und muß von dort importiert werden.
2. `Frame` ist bereit zum Benutzen, wie unser Rechteck.
3. Um `Frame` zu benutzen, muß man zunächst _____).
4. Um die Position einzustellen, benutzt man `frameName.setPosition(double ypos, float xpos)`.
5. Die Größe wird eingestellt mit `setGröße()`.
6. Die Größe kann mehrfach eingestellt werden.
7. Der Vektor $(r * \cos(x); r * \sin(x))$ beschreibt einen Kreis für alle x .
8. Ein Thread kann schlafen.
9. Man kann Anweisungsblöcke versuchsweise ablaufen lassen, indem man die dabei möglicherweise auftretenden Exceptions fängt.
10. Beim Fangen kann man zB gar nichts tun.
11. Man könnte noch einen `Frame` erzeugen und ihn, um 200 Pixel versetzt, gegenläufig rotieren lassen.
12. Man könnte noch einen `Frame` erzeugen und ihn eine liegende Acht fliegen lassen.
13. Man könnte noch einen `Frame` erzeugen und ihn entlang eines Quadrates fliegen lassen.

12.3 Lesen der Sun-Dokumentation

Alle Java-Klassen erben von `Object`.

Browsen Sie zu java.sun.com/j2se/1.4.2/docs/api/index.html und finden Sie heraus:

1. wie `Frame` von `Object` erbt;
2. in welcher Klasse die Methode zum Setzen der Position definiert wird;
3. wie es kommt, daß `Frame` davon weiß;
4. ob man das Verändern der Framegröße durch den Benutzer abschalten kann;
5. in welcher Klasse die Methode zum Setzen des Titels definiert wird und warum;
6. welches *default layout* ein `Frame` hat und wo das steht.

12.4 Übung VierFenster

Setzen Sie die oben genannten Änderungen um.

12.5 Optional: Knöpfe (Für Neugierige)

```
import java.awt.*;

public class Frame2 extends Frame{

    public static void main (String args[]) { Frame2 f2 = new Frame2 (); }

    public Frame2 () {
        setVisible (true);
        setLayout (new FlowLayout ());
        Button b1 = new Button ("knopf01");
        add (b1);
        pack ();
        Button b2 = new Button ("knopf02");
        add (b2);
        pack ();
        setTitle ("Ich bin ein Fenster");
        setSize (400, 200);
        setLocation (1, 1);
        int i = 0;
        while (true) {
            i++;
            int x = (int) (200 + 100 * Math.cos (i / 99.0));
            int y = (int) (200 + 100 * Math.sin (i / 99.0));
            setLocation (x, y);
            try { Thread.sleep (10); } catch (Exception e) {}
        }
    }
}
```

13 Rekursion

13.1 Beispiel Fakultät

$n!$ (lies: Enn Fakultät) ist $1 * 2 * 3 * \dots * n$. $0!$ ist 1. $1!$ ist auch 1.

Schreiben Sie : Legen Sie eine Tabelle an mit n von 0 bis 6 und $n!$.
Erkennen Sie:

$$5! = 5 * 4!$$

oder allgemeiner:

$$n! = n * (n - 1)!$$

Die Funktion `faku (int i)` könnte man also schreiben wie folgt:

Wenn $i=0$ oder 1; dann gib 1 zurück; sonst gib $i * \text{faku}(i - 1)$ zurück.

Schreiben Sie : die Funktion `faku(int j)`. Sie soll `long` zurückgeben. Finden Sie heraus, bis zu welcher Fakultät dieser Datentyp ausreicht.

13.2 Beispiel Fibonacci

Die erste und zweite Fibonacci'sche Zahl ist 1.

Die anderen sind die Summe ihrer beiden Vorgänger.

Lassen Sie Ihren Rechner rekursiv die 40. Fibonacci'sche Zahl ausrechnen. Haben Sie Geduld.
Kontrolle: Die 40. F'sche zahl ist 165580141.

13.3 Beispiel Türme von Hanoi

Ein König hat einen wunderschönen Turm aus 7 aufeinanderliegenden Kreisscheiben, von denen jede ein wenig kleiner als die untere ist. Er steht auf Bauplatz 1.

Zum Bau einer Hyperraum-Umgehungsstraße muß der Turm aber nach Bauplatz 3 umziehen.

Leider kann der Turm nur Scheibe für Scheibe transportiert werden. Leider können Scheiben nur auf Bauplatz 1, 2 oder 3 zwischengelagert werden. Leider kann jede Scheibe nur kleinere Scheiben tragen.

Nehmen Sie ein Buch, ein kleines Buch, eine Zigarettenschachtel, ein Radiergummi und Ihren Nachbarn, und bauen Sie einen 4er Turm um.

Erkennen Sie: Um die n . Scheibe von oben umlegen zu können, muß ich

1. die $n-1$. bis zur obersten Scheibe umlegen;
2. die n . Scheibe umlegen;
3. die $n-1$. bis oberste Scheibe da oben drauf legen.

Programmieren Sie den Umzug eines 5er Turms. Lösung:

```
Hanoi by wortgsone, Mon Apr 9 2007, Hanoi of 5
1-3 1-2 3-2 1-3 2-1 2-3 1-3 1-2 3-2 3-1 2-1 3-2 1-3 1-2
3-2 1-3 2-1 2-3 1-3 2-1 3-2 3-1 2-1 2-3 1-3 1-2 3-2 1-3
2-1 2-3 1-3
count = 31
```

14 Rekursion2 mit Sortierverfahren (optional)

Neben Bubblesort, Minsort und BozoSort³ gibt es ernstzunehmende, schnelle Sortierverfahren.

14.1 MergeSort

1. Denk dir eine schlaue Abbruchbedingung aus.
2. Teile die zu sortierenden Werte in zwei Haufen, und Mergesortiere jeden davon.
3. Setze die beiden sortierten Haufen zu **einem** sortierten Haufen zusammen.

Die Programmierung ist komplex und braucht stets ein zweites Feld zur Aufnahme des zusammenzusetzenden Feldes.

14.2 QuickSort

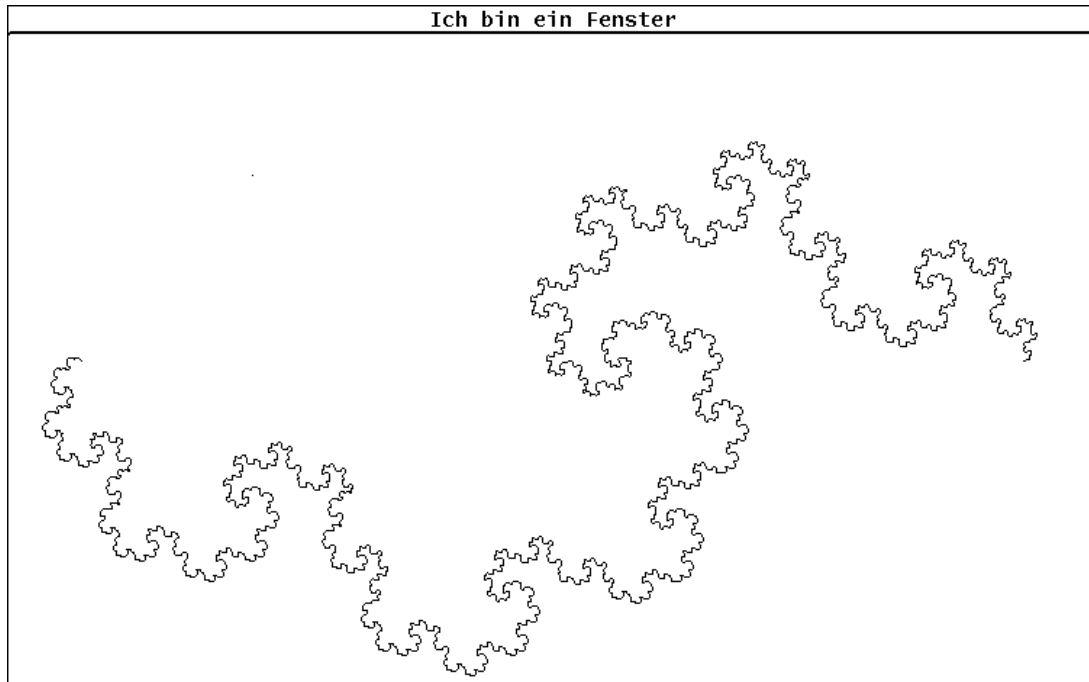
1. Denk dir eine schlaue Abbruchbedingung aus.
2. Such dir einen Median (Vergleichswert), zB die erste Zahl.
3. Wirf alle Werte, die größer sind als der Median, rechts ins Feld.
4. Wirf alle Werte, die kleiner sind als der Median, links ins Feld.
5. Setze den Median in die Mitte zwischen dem rechten und dem linken Haufen.
6. Quicksortiere den kleineren Haufen.
7. Quicksortiere den größeren Haufen.

Die Programmierung ist sehr komplex. Muß. Hat nämlich bei mir nicht geklappt, ohne ein zweites Feld zu Hilfe zu nehmen.

³(1) Mischen. (2) Gucken: Ist's sortiert? Wenn nicht, weiter bei (1).
Frißt Rechner-Ressourcen ohne Ende.....

15 Rekursive Grafik (Fraktale) (optional)

Mit einem Frame, einer `paint()`-Methode und einem schlaunen, rekursiven `linie()` ist es möglich, Fraktale zu malen.



Fraktale Linie mit 2 Knicks, Tiefe=12.

15.1 Use the Source, Luke.

```
import java.awt.*;

public class DrawLineTest extends Frame {

    static double f = 0.4;           // smaller factor
    static double f2 = 0.3;          // smaller factor 2
    static double w = 0.5;           // angle
    static double w2 = 0.7;          // angle 2
    final static int MAXDEPTH = 12;  // by trying
    static int depth;
    final static int N = 99999;      // max number of lines
    static int lc = 0;                // line counter
    static int x1, y1, x2, y2;
    static int p[] = new int[N];      // x1
    static int q[] = new int[N];      // y1
    static int r[] = new int[N];      // x2
    static int s[] = new int[N];      // y2

    public static void linie
        (int lx1, int ly1, int lx2, int ly2, int depth2) {
        char c = ' ';
        double dx = lx2 - lx1;
        double dy = ly2 - ly1;
        double rr = Math.sqrt (dx * dx + dy * dy);
    // no wait forever
```

```

    if (depth2 >= MAXDEPTH || depth2 > depth || lc > N - 3 || rr < 2) {
        p[lc] = lx1;
        q[lc] = ly1;
        r[lc] = lx2;
        s[lc] = ly2;
        lc++;
    } else {
        double xx = dx * Math.cos (w) - dy * Math.sin (w);
        double yy = dy * Math.cos (w) + dx * Math.sin (w);
        int xxi = lx1 + (int) (xx * f);
        int yyi = ly1 + (int) (yy * f);
        linie (lx1, ly1, xxi, yyi, depth2 + 1);
        linie (xxi, yyi, lx2, ly2, depth2 + 1);
    } }

public static void linie2
(int lx1, int ly1, int lx2, int ly2, int depth2) {
    char c = ' ';
    double dx = lx2 - lx1;
    double dy = ly2 - ly1;
    double rr = Math.sqrt (dx * dx + dy * dy);
    if (depth2 >= MAXDEPTH || depth2 > depth || lc > N - 3 || rr < 2) {
        p[lc] = lx1;
        q[lc] = ly1;
        r[lc] = lx2;
        s[lc] = ly2;
        lc++;
    } else {
        double w2 = Math.atan2 (dy, dx);
        double xx = dx * Math.cos (w2) - dy * Math.sin (w2);
        double yy = dy * Math.cos (w2) + dx * Math.sin (w2);
        int xxi = lx1 + (int) (xx * f);
        int yyi = ly1 + (int) (yy * f);
        int xxi2 = lx2 - (int) (xx * f2);
        int yyi2 = ly2 - (int) (yy * f2);
        linie2 (lx1, ly1, xxi, yyi, depth2 + 1);
        linie2 (xxi, yyi, xxi2, yyi2, depth2 + 1);
        linie2 (xxi2, yyi2, lx2, ly2, depth2 + 1);
    } }

public static void main (String args[]) {
    DrawLineTest dlt = new DrawLineTest ();
}

public DrawLineTest () {
    setVisible (true);
    setTitle ("Ich bin ein Fenster");
    setSize (750, 450);
    setLocation (9, 9);
    for (depth = 1; depth <= MAXDEPTH; depth++) {
        System.out.println ("depth = " + depth);
//        linie (50, 50, 700, 50, 0);
        linie2 (50, 225, 700, 225, 0);
    }
}

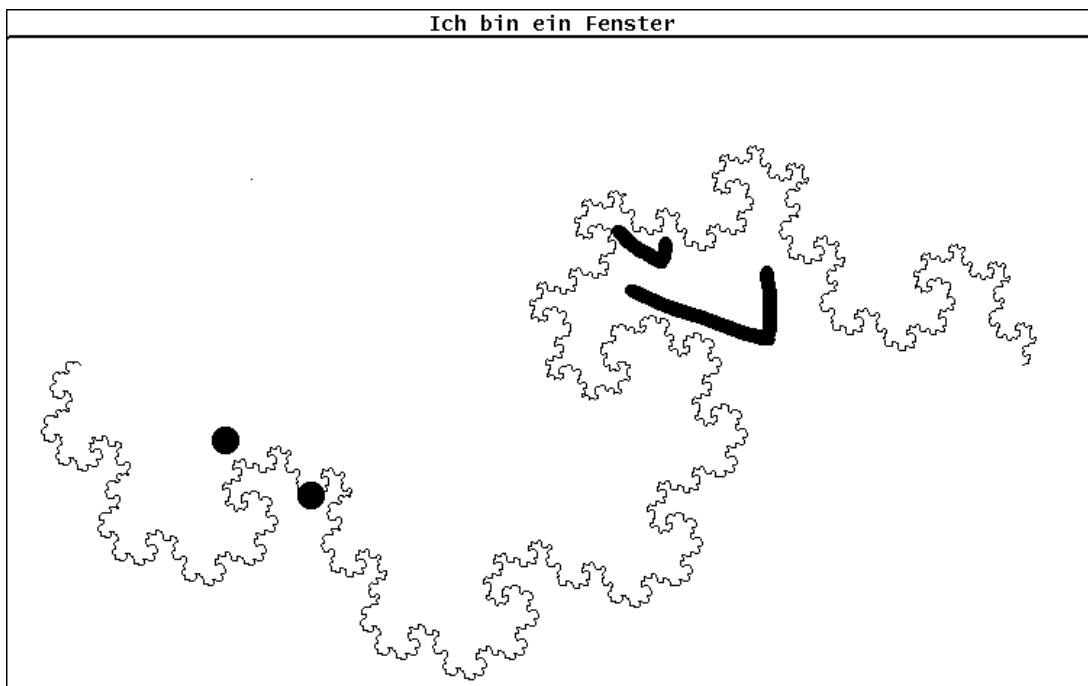
```

```

    repaint ();
    try { Thread.sleep (1000); } catch (Exception e) { }
} }

public void paint (Graphics g) {
    char c = ' ';
    lc++;
    while (lc > 0) {
        lc--;
        g.drawLine (p[lc], q[lc], r[lc], s[lc]);
    } } }

```



Fraktale Linie mit 2 Knicks, Tiefe=12, mit grafischen Anmerkungen.



Fraktale Linie mit 2 Knicks, Tiefe=12, Ausschnitt.

15.2 Vorschläge für Übung

Was ist ein Fraktal?

Was bedeutet "selbstähnlich"?

Warum steht das im Kapitel "Rekursion"?

Erweitern Sie das Fenster um

- Ausgabe "Ich kann nix" auf der Konsole
- Ausgabe im Fenster
- Reagieren auf 'q'
- Reagieren auf Schließen-Schaltfläche

16 Kommandozeilenparameter

In Kürze : man kann java Dateien ausführen durch F9 im Java Editor oder mit

```
java Klassenname
```

oder mit

```
java Klassenname keks kiks konkret
```

Im letzteren Fall gibt es ein Feld von Strings namens `args` (engl.: Argumente, d.h. Optionen die man auf der Kommandozeile mitgibt). Es kann die Methode `length` (Zweck mußst du raten) und hat eine gewisse Anzahl Elemente, das erste ist `args[0]`.

```
class CommandLineParams {
    public static void main (String[]args) {
        System.out.println ("CommandLineParams by worgtsone");
        if (args.length > 0) {
            Q.p ("Es gibt Parameter!");
            for (int i=0; i<args.length; i++)
                Q.p (i + Q.e + args[i]);
        }
    }
}
```

16.1 Übung

Schreiben Sie ein Programm, das 5 bis 10 Kommandozeilenparameter liest (oder sagt : "Usage : programmname und 5 bis 10 integer!!!") und sie aufsteigend sortiert ausgibt.

Erweitern Sie es: falls die erste Option `-reverse` lautet, soll es **aufsteigend** sortieren.

- 17 Klasse System – verschoben nach St. Nimmerlein**
- 18 Interfaces und Adapterklassen – 2 be done – verschoben nach St. Nimmerlein**
- 19 2- und 3-dim Arrays, Enums, Hashes, Trees, Sets und Maps – 2 be done (optional) – verschoben nach St. Nimmerlein**

20 Dateien lesen und schreiben

20.1 So gehts:

```
import java.io.*;

public class FileOps {
    public static void main (String[] args) throws IOException {
        File file = new File("Fout.txt");
        Writer out = new BufferedWriter(new FileWriter(file));
        int max=9;
        for (int i=1; i<=max; i++)
            out.write("Zeile " + i + " von " + max + "\n");
        out.close();
        Q.p("file written.");

// read it and display it
        BufferedReader in = new BufferedReader(new FileReader(file));
        String line;
        while ((line = in.readLine()) != null)    {
            Q.p ("gelesen : " + line);
        }
        in.close();
    }
}
```

20.2 Nach Zahlen

Damit haben wir Strings aus der Datei gelesen, aber keine Zahlen. Zum Glück gibt es in den KlassenFürPrimitiveDatentypen (`Integer`, `Double` etc.) einen Parser, der den primitiven Wert ausgibt und `valueOf()` heißt.

```
int i=Integer.valueOf(s); double d=Double.valueOf(s);
```

20.3 Übung : Erzeugen

Erzeugen Sie eine Datei mit 2 000 000 000 `long` Zufallszahlen.

20.4 copy

Schreiben Sie ein Programm, das die Datei im ersten Kommandozeilenparameter in die Datei im zweiten Kommandozeilenparameter kopiert.

Vorsicht: Überschreiben Sie nichts, was Sie später bereuen.

Vorsicht: Sie dürfen die Kopie auf Existenz testen, bevor Sie sie überschreiben. TESTEN SIE DIESE ROUTINE VORHER SORGFÄLTIG.

20.5 Übung : Sortieren

Erzeugen Sie eine Datei `unsorted.txt` mit 1 000 000 `double` Zufallszahlen.

Sortieren Sie sie anschließend nach `sorted.txt`. Machen Sie so viele Nebendateien wie sie brauchen.

21 TODO : Arbeitsblatt – Streams am Beispiel von Tastatur.java

21.1 Byte-Based Input

InputStream ist **die** Basisklasse für byte-basierte Eingabe.

Sie hat **viele** nützliche Unterklassen, zB FileInputStream mit den wichtigen Methoden read(), read(byte()) und void close().

```
FileInputStream fis = new FileInputStream ("filename");
```

21.2 Byte-Based Output

OutputStream ist **die** Basisklasse für byte-basierte Ausgabe.

Sie hat **viele** nützliche Unterklassen, zB FileOutputStream mit den wichtigen Methoden write(int), write(byte()), flush() und void close().

```
FileOutputStream fos = new FileOutputStream ("filename");
```

21.3 Puffern

BufferedInputStream und BufferedOutputStream puffern und beschleunigen Ein- und Ausgabe.

21.4 Unicode-Basierte Ein- und Ausgabe

Reader und FileReader; Writer und FileWriter; BufferedReader und Buffered Writer:

```
BufferedReader br = new BufferedReader (new FileReader("filename"));
```

21.5 Konvertieren zwischen Unicode und Bytestream

mit InputStreamReader und OutputStreamWriter.

22 Ein simpler Taschenrechner (optional)

Aufgaben: Abtippen. Laufenlassen. Umbenennen und herumspielen.
Anschließend Fragen beantworten.

```
import java.awt.*;
import java.awt.event.*;

class TR2 extends Frame {

    TextField tf1 = new TextField();
    TextField tf2 = new TextField();
    TextField tf3 = new TextField();
    Button plus = new Button ("plus");
    Button pack = new Button ("pack()");
    Button b03 = new Button ("FlowLayout!");
    Button b04 = new Button ("GridLayout(0,0)");
    Button b05 = new Button ("GridLayout(0,4)");
    Button b06 = new Button ("GridLayout(4,0)");

    TR2 (String title){
        super (title);
        addWindowListener(new WindowAdapter() {
            public void windowClosing(WindowEvent evt)    { System.exit(0) ; } } );
        setSize (300,300);
        setLocation(99,99);
        setVisible(true);
        setLayout(new FlowLayout());
        tf1.setText("      ");
        tf2.setText("      ");
        tf3.setText("      ");
        add (tf1);
        add (tf2);
        add (plus);
        add (tf3);
        add(pack);
        add(b03);
        add(b04);
        add(b05);
        add(b06);
        pack();  sleep (999);
        pack.setBackground(new Color(99,99,199));
        pack.setForeground(new Color(255,255,255));
        for (int i=0; i<25; i++)
            if (i!=24) add (new Button ("button "+i));
            else add (new Button ("Ich bin der \n boese boese \n Knopf!!!"));
        add (new Label ("Ich bin das \n boese boese \n Label!!!"));

        plus.addActionListener ( new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                double a=Double.parseDouble(tf1.getText());
                double b=Double.parseDouble(tf2.getText());
                double c=a+b;    tf3.setText(c+"");
            }
        }
    }
}
```

```

    }
  });

  pack.addActionListener ( new ActionListener() {
    public void actionPerformed(ActionEvent e) {
      pack();      }  });

  b03.addActionListener ( new ActionListener() {
    public void actionPerformed(ActionEvent e) {
      setLayout(new FlowLayout()); } });

  b04.addActionListener ( new ActionListener() {
    public void actionPerformed(ActionEvent e) {
      setLayout(new GridLayout()); } });

  b05.addActionListener ( new ActionListener() {
    public void actionPerformed(ActionEvent e) {
      setLayout(new GridLayout(0,4)); } });

  b06.addActionListener ( new ActionListener() {
    public void actionPerformed(ActionEvent e) {
      setLayout(new GridLayout(4,0)); } });
}

public static void main (String[]args) { new TR2("TR2"); }

public void sleep (int msec) { try { Thread.sleep(msec); } catch (Exception e) {} }
}

```

1. Machen Sie eine Liste aller neuen Klassen mit allen Methoden und was sie dazu brauchen.
2. Wo haben sich die neuen Klassen versteckt?
3. Früher haben wir den Inhalt unserer Programme nach `main()` geschrieben. Was ist hier anders?
4. Was macht `sleep()`?
5. Beschreiben Sie jedes Layout in 1 bis 2 Sätzen.
6. Was passiert beim Klick auf einen Knopf? Warum?
7. Wie aktivieren Sie ein neues Layout?
8. Schauen Sie in das Verzeichnis mit den *.class-Dateien. Sind Unterklassen entstanden? Name?
9. Können Sie das Fenster schließbar machen?
10. Insgesamt wird hier nur bestellt, wie das Fenster aussehen soll und was beim Knopfdrücken passieren soll. Was tut das Fenster, solange nix gedrückt wird?
11. Angeblich kann man an ein TextField einen TextListener binden, der im Fall `textValueChanged` versucht, den TextField-Inhalt zu parsen, und den Hintergrund rosa färbt, sofern das nicht klappt.
12. Erweitern Sie den Taschenrechner um - * / mod ln ggT kgV.

13. Erweitern Sie den Taschenrechner um die pq-Formel. Schreiben Sie das Ergebnis in ein TextArea(8,8).
14. Erweitern Sie den Taschenrechner mittels BigInteger um die Fakultäten von 0 bis 1000 – bis zur letzten Stelle.

23 Projekte

23.1 Rationale Zahlen (Brüche)

Wir wollen eine Klasse für Brüche schreiben und sie anschließend auch benutzen.

Übung: Welche Fragen stellen sich?

Lösung:

- Was weiß eine Rationale Zahl über sich selbst? — Zähler, Nenner, istDefiniert.
- Was kann eine Rationale Zahl? — Entstehen, kürzen, eineRationaleZahlDazuaddieren, - Abziehen, -Multiplizieren, -Teilen.
- Was soll ich nur erben? — Gar nichts.
- Wie schreibt man das in Java?
- Wie benutzt man das?
- Welchen Zugriff erlaube ich von außen? — getZaehler, getNenner, getDezimal, getBruch.

23.2 Anmerkungen

- Kapseln Sie zu kapselnde Daten.
- Wer weiß, wie man Brüche addiert ? – Ein Bruch.
- Schreiben Sie eine Methode, die einen weiteren Bruch zu einem Bruch addiert.
- Schreiben Sie eine(bitte ausfüllen), die einen Bruch als Summe zweier Brüche NEU GENERIERT.

23.3 Menuprogramm Layoutdemo

23.4 Freier Fall

23.5 Pendel und Doppelpendel

23.6 Mandelbrot-Mengen

23.7 Schülerverwaltung

23.8 TaschenRechner