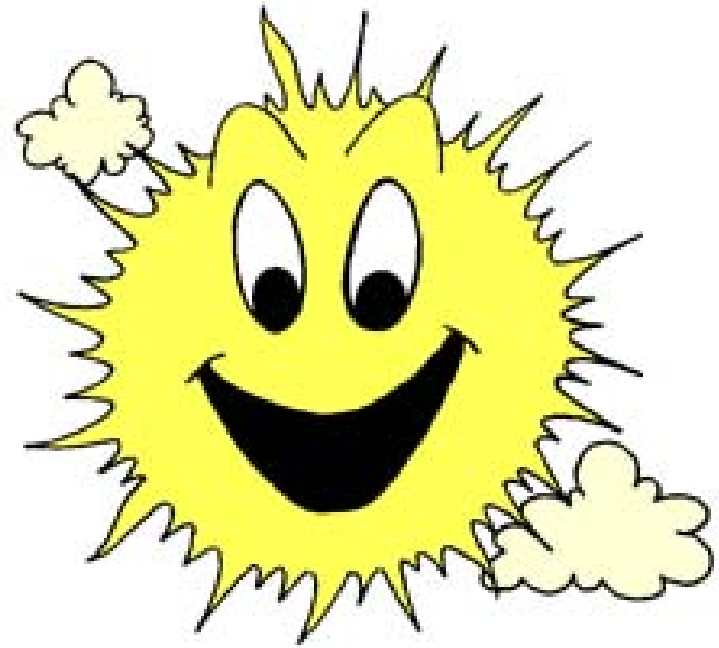
Wiederholung 12. Vorlesung



Vorgehensweise beim Layout

- notwendige Komponenten aus Use Cases ermitteln
- Komponenten freihändig gruppieren und positionieren
- Flucht- und Symmetrielinien identifizieren / definieren
  - Anzahl der Fluchtlinien klein halten
  - Fluchtlinien können gezeichnete Linien ersetzen: "gutes Design braucht keine (gezeichneten) Linien"

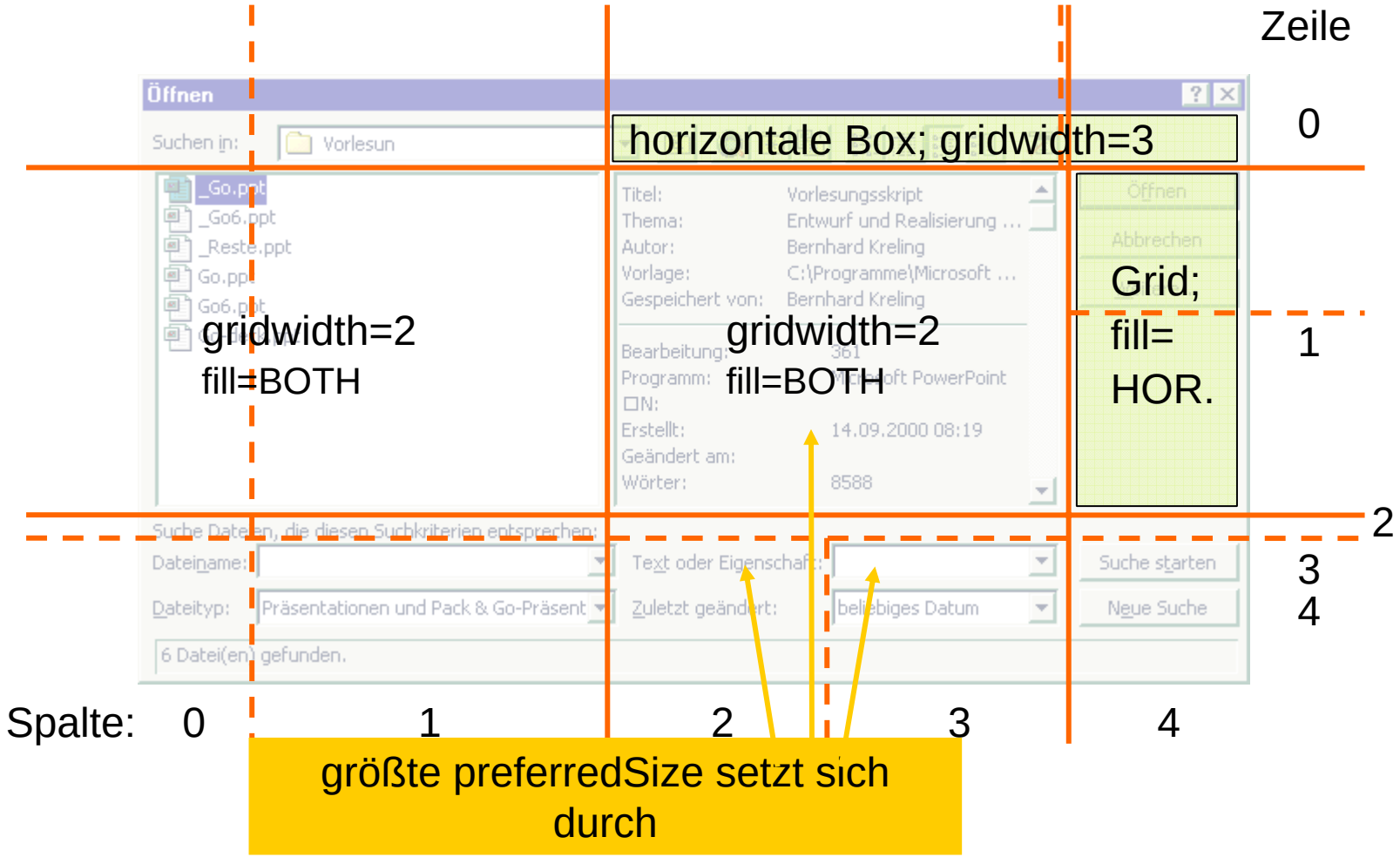
das Auge ist sehr empfindlich !

- Rasterzeilen und -spalten bemaßen
  - dyn. Layout: abhängig von Schrift-, Fenster- und Icongröße
    - bei mehrsprachigen Anwendungen auch abhängig von Textlänge
    - im allgemeinsten Fall sind alle Maße variabel!
  - (statisches Layout: absolute Zahlen in Pixel)

Analyse → Gestaltung → Konstruktion

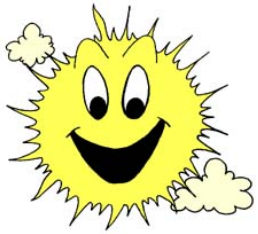
Layout mit GridBagLayout

alle Zellen in Zeilen 0,2,3,4: fill=HORIZONTAL



Debugging in JBuilder

Ende der Wiederholung



Evaluierung der Vorlesung WS05/06 Zug C

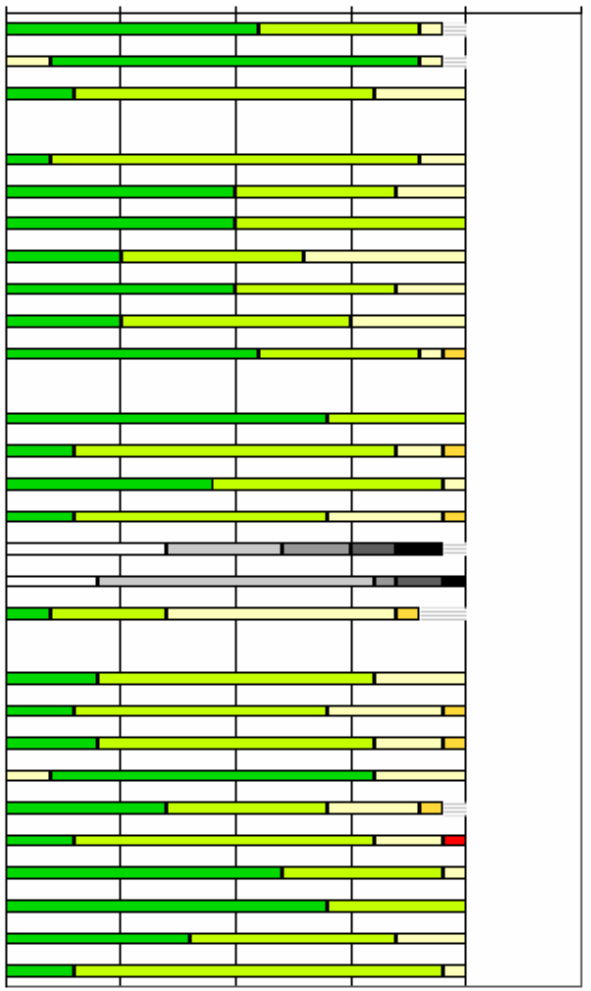
Informatik

Auswertung der Fragebögen einer Lehrveranstaltung **mayerBC304306kursc**

Farbe der Noten: 1 2 3 4 5 Enthaltung

Anzahl der Noten: 0 5 10 15 20 25

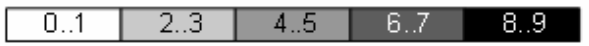
			Mittelwert
1	Die Lehrveranstaltung ist zur Erfüllung der Ziele des Studienprogramms notwendig		1,5
2	Diese Lehrveranstaltung ist in der Abfolge aller Lehrveranstaltungen zu früh / richtig / zu spät		2,9
3	LV bietet entspr. ihrer Zielsetzung eine angem. Mischung zw. prakt. Relevanz u. theor. Fundierur		2,1
4	Dozent(in) kann komplizierte Sachverhalte gut erklären		2,0
5	Dozent(in) geht partnerschaftlich mit den Studierenden um		1,7
6	Dozent(in) ist fachlich kompetent und gut vorbereitet		1,5
7	Dozent(in) vermittelt Begeisterung für das Fach		2,1
8	Dozent(in) drückt sich klar, einfach, gut verständlich und treffend aus		1,7
9	Dozent(in) führt gut in die Fachterminologie ein		2,0
10	Dozent(in) geht gut auf Fragen und Bemerkungen ein		1,6
11	Ich habe regelmäßig an den Lehrveranstaltungen teilgenommen		1,3
12	Wir Studierenden werden zur Beteiligung und aktiven Mitarbeit angeregt		2,1
13	Ich konnte der Veranstaltung gut folgen		1,6
14	Die Diskussionen in der Veranstaltung sind verständlich und informativ		2,2
15	Mein wöchentl. Arbeitsaufwand für den Kurs beträgt [0..1 / 2..3 / 4..5 / 6..7 / 8..9] Std.	*)	3,1
16	Zur Vorbereitung der Klausur werde ich [0..1 / 2..3 / 4..5 / 6..7 / 8..9] Tage lernen	*)	2,9
17	Am Ende meines Studiums werde ich zu den sehr guten ... schwächeren Studenten gehören		2,6
18	Die Inhalte der Lehrveranstaltung sind interessant und anregend		2,0
19	Die Ziele der Lehrveranstaltung sind klar erkennbar		2,2
20	Der Verlauf ist systematisch und gut gegliedert; der Stoff wird in logischer Folge präsentiert		2,1
21	Die Anforderungen sind zu niedrig=1 / .. / angemessen=3 / .. / zu hoch =5		3,1
22	Das Skript ist gut hinsichtlich Umfang, Inhalt und Gliederung		1,9
23	Man bekommt viel Problembewußtsein und gute Arbeitsmethodik vermittelt		2,2
24	Der Stoff wird durch Beispiele, Vergleiche, Skizzen und Diagramme gut veranschaulicht		1,5
25	Medien (Tafel, Overhead, Beamer, Video, ...) werden zweckmäßig eingesetzt		1,3
26	Die Lehrveranstaltung ist gut organisiert		1,8
27	Gesamteindruck der Lehrveranstaltung (übliche Notenskala von 1 bis 5)		1,9



Version 3.7.1

Die Länge der Teil-Balken entspricht der Häufigkeit der einzelnen Noten. Die Zahl ist das arithmetische Mittel.

*) Legende zu 15+16:



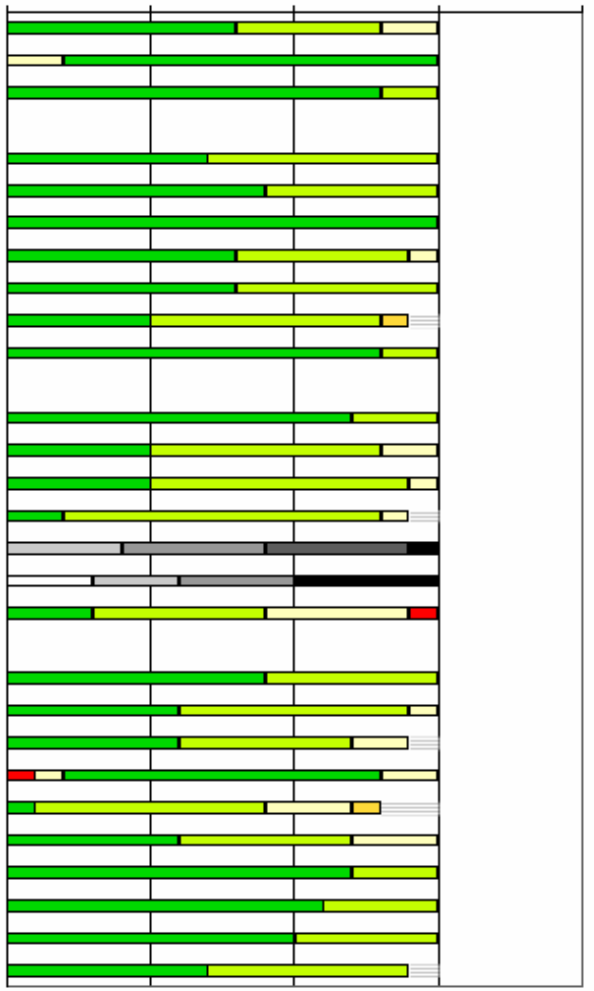
Evaluierung der Vorlesung WS05/06 Zug B

Informatik

Auswertung der Fragebögen einer Lehrveranstaltung **mayerBC304306kursb**

Farbe der Noten: 1 2 3 4 5 Enthaltung

			Anzahl der Noten: 0 5 10 15 20 Mittelwert
1	Die Lehrveranstaltung ist zur Erfüllung der Ziele des Studienprogramms notwendig		1,6
2	Diese Lehrveranstaltung ist in der Abfolge aller Lehrveranstaltungen zu früh / richtig / zu spät		2,9
3	LV bietet entspr. ihrer Zielsetzung eine angem. Mischung zw. prakt. Relevanz u. theor. Fundierur		1,1
4	Dozent(in) kann komplizierte Sachverhalte gut erklären		1,5
5	Dozent(in) geht partnerschaftlich mit den Studierenden um		1,4
6	Dozent(in) ist fachlich kompetent und gut vorbereitet		1,0
7	Dozent(in) vermittelt Begeisterung für das Fach		1,5
8	Dozent(in) drückt sich klar, einfach, gut verständlich und treffend aus		1,5
9	Dozent(in) führt gut in die Fachterminologie ein		1,8
10	Dozent(in) geht gut auf Fragen und Bemerkungen ein		1,1
11	Ich habe regelmäßig an den Lehrveranstaltungen teilgenommen		1,2
12	Wir Studierenden werden zur Beteiligung und aktiven Mitarbeit angeregt		1,8
13	Ich konnte der Veranstaltung gut folgen		1,7
14	Die Diskussionen in der Veranstaltung sind verständlich und informativ		1,9
15	Mein wöchentl. Arbeitsaufwand für den Kurs beträgt [0..1 / 2..3 / 4..5 / 6..7 / 8..9] Std.	*)	4,9
16	Zur Vorbereitung der Klausur werde ich [0..1 / 2..3 / 4..5 / 6..7 / 8..9] Tage lernen	*)	4,6
17	Am Ende meines Studiums werde ich zu den sehr guten ... schwächeren Studenten gehören		2,3
18	Die Inhalte der Lehrveranstaltung sind interessant und anregend		1,4
19	Die Ziele der Lehrveranstaltung sind klar erkennbar		1,7
20	Der Verlauf ist systematisch und gut gegliedert; der Stoff wird in logischer Folge präsentiert		1,7
21	Die Anforderungen sind zu niedrig=1 / .. / angemessen=3 / .. / zu hoch =5		2,9
22	Das Skript ist gut hinsichtlich Umfang, Inhalt und Gliederung		2,3
23	Man bekommt viel Problembewußtsein und gute Arbeitsmethodik vermittelt		1,8
24	Der Stoff wird durch Beispiele, Vergleiche, Skizzen und Diagramme gut veranschaulicht		1,2
25	Medien (Tafel, Overhead, Beamer, Video, ...) werden zweckmäßig eingesetzt		1,3
26	Die Lehrveranstaltung ist gut organisiert		1,3
27	Gesamteindruck der Lehrveranstaltung (übliche Notenskala von 1 bis 5)		1,5



Version 3.7.1

Die Länge der Teil-Balken entspricht der Häufigkeit der einzelnen Noten. Die Zahl ist das arithmetische Mittel.

*) Legende zu 15+16:

0..1	2..3	4..5	6..7	8..9

Java: Übersicht und Wiederholung

Allgemein

- **`package`** (1. Anweisung in der Datei) ordnet eine Klasse einem Paket (= Gruppe von logisch zusammenhängenden Klassen) zu
- Die Import Anweisung macht die entsprechende Klasse sichtbar/verfügbar
- **`instanceof`** - testet, ob ein Objekt Instanz einer best. Klasse ist:

```
... if (obj instanceof class) ...
```

- **`int, float, double, boolean,`** … sind intrinsic (= elementarer Bestandteil), Wrapper-Klassen (**`Integer, Float, Double, Boolean`**, ...) existieren etwa für Typumwandlungen

Bibliotheken: Packages

- Package ist ein Ordner im Dateisystem
  - enthält mehrere .java-Dateien und Subpackages (=Unterordner)
- Zugehörigkeit zu Package **`package meinpackage;`**
- Import-Anweisung macht andere Packages verfügbar
  - eine Klasse: **`import java.util.ArrayList;`**
  - alle Klassen: **`import java.util.*;`**
  - alle Klassen mit Kurzreferenz: **`import java.util;`**
    Kurzreferenz ist dann: **`util.ArrayList;`**
- weltweit eindeutige Namen für wichtige Packages
  - **`package de.fh-darmstadt.fbi.projekt.meinpackage;`**
  - spiegelt die Schachtelung der Ordner wider

Klassen

```
public class NeueKlasse extends Basisklasse
  { ... }
```

- Konstruktor
  - Vorinitialisierungsliste gibt es nicht
  - **`super(...)`** ruft den Konstruktor der Basisklasse auf
  - Attribute können an der Deklarationsstelle oder im Konstruktor initialisiert werden
- alle Methoden sind standardmäßig virtuell
  - Schlüsselwort final (="nicht-virtuell") verhindert Überschreiben
- keine separaten Header-Dateien
- *jede* Klasse kann zum Testen eine **`main`**-Methode haben

Mutter aller Klassen: class Object

- viele Klassen sind (indirekt) von class Object abgeleitet

```
public class Object
 {
 public String toString() {...}
    //liefert Textdarstellung des Objekts

  public boolean equals(Object obj) {...}
    // vergleicht Objekt-Inhalte

  protected Object clone()
    // dupliziert das Objekt
      throws CloneNotSupportedException {...}

  protected void finalize()
    // Aufruf vor der Garbage Collection
      throws Throwable {...}
}
```

Klassenhierarchie

```
class NeueKlasse extends Basisklasse
                 implements Interface1, Interface2
```

- keine Mehrfachvererbung, nur Einfachvererbung
  - aber Implementierung mehrerer `interfaces` möglich
  - damit wird das Problem in verschiedenen Basisklassen eventuell doppelt vorhandener Elementvariablen umgangen; virtuelle Ableitung ist daher überflüssig
- Besonderheiten
  - eine Klasse muß mit `abstract` markiert werden, wenn sie `abstract` Methoden enthält
  - eine `abstract` Klasse kann nicht instantiiert werden!
  - eine Klasse, die als `final` markiert ist, kann nicht mehr als Basisklasse dienen!

Objekte

- Objekte werden **immer** dynamisch allokiert
  - **new** liefert einen Zeiger auf das neue Objekt
  - **`Klasse Objektreferenz = new Klasse (Parameter);`**
- delete gibt es nicht
  - dynamisch allokierter Speicher wird automatisch vom Garbage Collector freigegeben, nachdem kein Zeiger mehr darauf zeigt
  - der Aufrufzeitpunkt des Garbage Collector ist undefiniert;
    das Laufzeitsystem macht das irgendwann
- Destruktor gibt es nicht
  - stattdessen Methode **`protected void finalize();`**
  - Aufruf von dem Garbage Collector – d.h. unter Umständen nie !

Zur Lebensdauer von Objekten

- grundsätzlich existiert ein Objekt in Java so lange, wie eine Referenz darauf existiert !
  - manchmal auch ein bißchen länger,
    wie es dem Garbage Collector gerade gefällt

- warum werden dann die Referenzen in JBuilder GUI-Applikationen nicht gespeichert ?
  - das Applikationsobjekt ist tatsächlich nur temporär und insofern ziemlich nutzlos – dort sollte man keine Attribute ansiedeln !
    - main ist auch sofort wieder zu Ende
  - die Klasse Frame verwaltet eine eigene Liste von Referenzen auf alle ihre Instanzen – dadurch bleiben diese am Leben
    - jede Instanz hat einen eigenen Thread – dadurch läuft die VM weiter

Datentypen

plattformunabhängig definiert

- Primitive Typen
  - byte (8 Bit), short (16 Bit), int (32 Bit), long (64 Bit), alle als vorzeichenbehaftete 2er-Komplementdarstellung
  - char (16 Bit vorzeichenlos, Unicode → Umlaute in Variablen erlaubt)
  - float (32 Bit), double (64 Bit); beide IEEE Standard 754
  - **kein** enum; stattdessen Klassen/Interfaces mit Konstanten (`static final int`)
- Referenztypen
  - "Zeiger" für Klassen, Interfaces, Arrays
- C++ macht keinen Unterschied zwischen primitiven Typen und Klassen!
  - neue primitive Typen werden in C++ als Klassen realisiert (CBruch, CComplex, CString)

Initialisierung von Variablen

- automatisch initialisiert werden
  - Klassenvariable (= **`static`**)
  - Instanzvariable (= Attribute)
  - Array-Komponenten (wenn es sich um Grundtypen handelt)
- lokale Variable werden nicht automatisch initialisiert !
  - Zugriff auf nicht initialisierte Variable bringt Fehler
- Initialisierungswerte sind
  - **`0`** für Zahlentypen
  - **`false`** für boolean
  - **`null`** für Objektreferenzen

Zeichenketten: String

- Klasse String (ähnlich wie CString in MFC)
  - char* gibt es nicht ! (Inhalts- und Addressoperator gibt es nicht)
- Operand + zur Verkettung
  - akzeptiert Werte beliebiger Typen oder Objekte beliebiger Klassen als Operand – verwendet Methode **`toString`**
  - Besonderheit: Operator + allokiert implizit einen neuen String!
- class Object hat die Methode **`toString`**()
  - liefert eine textliche Repräsentation des Objekts
  - jede abgeleitete Klasse sollte diese Methode sinnvoll überschreiben
  - nützlich auch zur Fehlersuche

Objektvariable: Zeiger versus Referenzen

- Java Referenzen entsprechen C++ Zeigern:
  - Java Referenz kann **`null`** sein
  - man kann damit z.B. verkettete Listen schreiben
  - Zugriff auf Elemente des Objekts mit . statt ->
  - ein Pendant zu C++ Referenzen gibt es nicht!
- größere Sicherheit gegen Programmierfehler
  - es gibt keine Zeigerarithmetik (so kann man sich auch nicht verrechnen...)
  - man kann kein Objekt löschen, auf das noch Zeiger zeigen

Arrays

- sicher durch Indexprüfung zur Laufzeit
- sind dynamisch allokierte Objekte
  - `float[] vector = new float[3];`
  - `int punkt[] = {40, 50};`
  - `float[][] matrix = new float[3][3];`
  - `int[][] brett = new int[3][]; // Array mit 3 Referenzen`

    ```
    brett[0][] = new int[10];
    brett[1][] = new int[8];
    brett[2][] = new int[4];
    ```
  - `int brett[3][3];` ist nicht möglich!
- Array ist eigentlich 1-dimensional
  - **`array.clone()`** klont auch bei mehrdimensionalen Arrays nur 1 Dimension

    ```
    int data[] = { 1, 2, 3, 4, 5};
    int copy[] = (int[]) data.clone();
    ```
  - siehe auch **`System.arraycopy(…)`** und Klasse **`Array`**

Typumwandlung/Casting

- Typumwandlung zwischen primitiven Typen ist wohldefiniert
  - nicht einfach Uminterpretation des Bitmusters
  - keine beliebigen Casts zwischen irgendwelchen Datentypen
- Typumwandlung zwischen Objektreferenzen wird zur Laufzeit überprüft
  - nur die Umwandlung zwischen Basisklasse und abgeleiteter Klasse ist zulässig

Exception Handling

- `throw`-Deklaration müssen im Funktionskopf angegeben werden
  `public void search() throws NotFoundException …`
  - der Compiler erzwingt es!
- alle möglichen Ausnahmen müssen auch eingefangen werden
- nur für Objekte von Klassen, die von Throwable abgeleitet sind
  - hat als direkte Unterklassen: `Error` und `Exception`
- **`try … catch (Exception e) … [finally]`**
  Die Anweisungen im finally-Block werden **grundsätzlich** ausgeführt
- Exception-Konstruktoren haben i.a. String-Objekt als Parameter
  - kann mit **`e.getMessage()`** zurückgewonnen werden

Zugriffsrechte

- public, protected, private wie in C++
- jedes Element hat sein eigenes Zugriffsrecht
  - Definition gilt (im Gegensatz zu C++) nicht für die nachfolgenden Elemente
- ohne Angabe bedeutet:
  - private für Klassen
  - public für Interfaces
  - für Attribute und Methoden
    - protected innerhalb des eigenen Packages
    - private außerhalb des eigenen Packages

→ zur Sicherheit immer angeben!

Sichtbarkeit

	Public	Protected	Package	Private
gleiche Klasse	Ja	Ja	Ja	ja
Klasse im gl. Paket	Ja	Ja	Ja	Nein
Subklasse in anderem Paket	Ja	Ja	Nein	Nein
Nicht Subklasse, and. Paket	Ja	Nein	Nein	Nein

(Auswahl von) Modifier (I)

- `void, int, long, float, double, boolean, char,... spez. Obj.` — normaler Rückgabewert;
- Konstruktoren haben KEINEN Rückgabewert
- `private` — Methoden/Komponenten können nur klassenintern verwendet werden
- `protected` — Das Element ist nur innerhalb des Paketes zu erreichen, in dem es definiert ist sowie über entsprechende Subklassen
- `public` — Methoden/Komponenten/Klasse/Interface stehen für Aufrufe von außen (=überall) zur Verfügung.

(Auswahl von) Modifier (II)

- `ohne` — Zugriff ist innerhalb des Paketes möglich (oft als "friendly" oder "package private " bezeichnet).
- `final` — Komponenten: Konstanten, die unmittelbar mit einem Wert versehen werden
  Bei Klassen verhindert das Schlüsselwort die weitere Verfeinerung der Klasse.
  Bei Methode: die Methode kann nicht überschrieben werden.

(Auswahl von) Modifier (III)

- `static` klassenweite Methoden/Komponenten, sie sind nicht an best. Objekte gebunden (das dann auch nicht zur Verfügung steht); sie werden so auch mit dem Klassennamen aufgerufen und nur einmal initialisiert, wenn das Programm geladen/initialisiert wird

(Auswahl von) Modifier (IV)

- `abstract`

Klassen die eine abstract Methode beinhalten müssen als abstract deklariert werden. Abstrakte Klassen enthalten nicht implementierte Methoden und können nicht instanziiert werden.

Abstracte Methoden müssen auch implementiert werden, sie werden allerdings in Subklassen spezifisch überschrieben (vgl. virtual in C++)

Wenn es nur abstract Methoden gibt, dann kann man die Klasse auch als **interface** deklarieren. Bei **interfaces** ist der Modifikator optional

Wiederholung 12. Vorlesung