JAVA

SE 8

AUTOR: TOMASZ LUBOWIECKI

plattformunabhängig

- Java Compiler erzeugt Bytecode
- Bytecode wird auf allen Systemen und jeder Hardware identisch durch den Interpreter übersetzt
- Bytecode wird durch die virtuelle Java Maschine (JVM) zur Laufzeit übersetzt

einfach

- übernimmt viele C und C++ Konzepte
- viele fehleranfällige Details wurden entfernt

dynamisch

- Umfangreiche Bibliothek im Lieferumfang enthalten
- Kann durch fremde Bibliotheken erweitert werden

objektorientiert

- Aufgaben werden in Klassen verpackt
- Vererbung ist erlaubt, aber keine Mehrfachvererbung

portierbar

- Wertebereiche der internen Datentypen sind fest vorgeschrieben
- Keine Differenzen auf unterschiedlichen Systemen

robust

- Typsicherheit
- Prüfung aller Typkonvertierungen

… und so läuft das ab …

Compiler

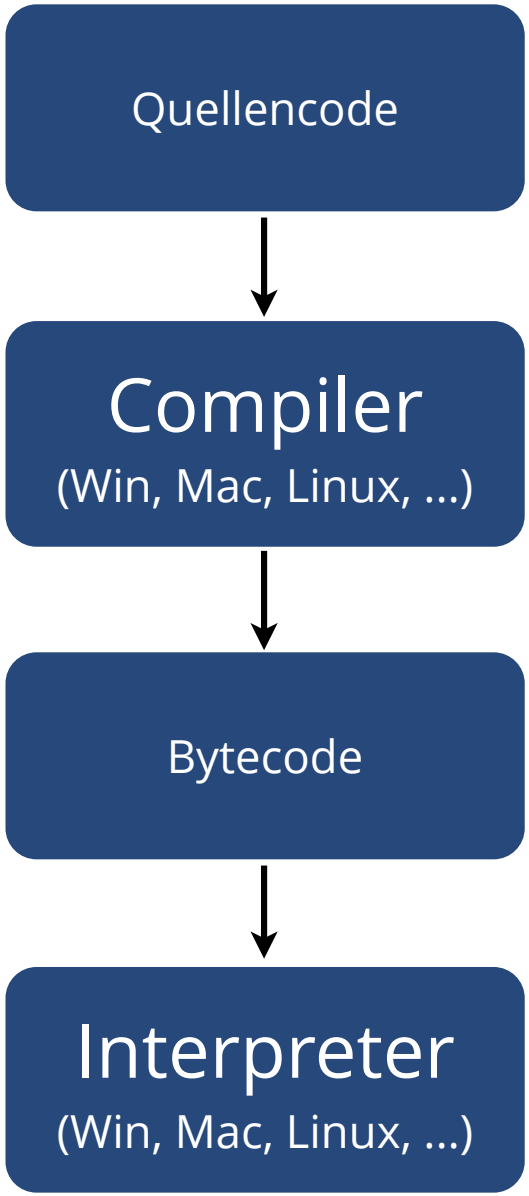
- Code wird kompiliert
- Syntax wird analysiert
- Alle Typumwandlungen werden auf ihre Gültigkeit hin getestet
- Stack wird permanent überprüft um eventuelle Überläufe oder Unterläufe von Werten aufzudecken
- Ergebnis: Bytecode

Bytecode

- Enthält an sich alle wichtigen Anweisungen
- Ist plattformunabhängig

Interpreter

- Bytecode wird gelesen und Anweisungen werden ausführt
- Fehlertests werden ständig durchgeführt
- Auftretende Fehler werden (sofern das möglich ist) abfangen und korrigiert


Quellencode
Compiler
(Win, Mac, Linux, ...)
Bytecode
Interpreter
(Win, Mac, Linux, ...)

was brauche ich?

- JRE (Java SE Runtime Environment)
- JDK (Java SE Development Kit)

Java Releas Zyklus

- Neue Java-Versionen werden innerhalb von sechs Monaten veröffentlicht
- Der Support der Versionen wird auf sechs Monate beschränkt.
  - Ausnahme: Long Term Support-Versionen (LTS-Versionen)
  - Alle 3 Jahre erscheint eine LTS-Version

Java Plattformen

- Java Standard Edition (Java SE)
- Java Enterprise Edition (Java EE)
  - integriert Pakete, die zur Entwicklung von Geschäftsanwendungen
- Java Micro Edition (Java ME)
  - Für mobile und embedded Devices, IoT

Struktur von Java-Klassen

Komponenten einer Java Klasse

- package Anweisung
- import Anweisung
- Kommentare
- Deklaration und Definition der Klasse
- Variablen
- Methoden
- Konstruktoren

Kommentare

```
/* multiline */

// end-of-line
```

Klassendeklaration

```
public final class Buch extends Medium
implements Lesbar {
    //…
}
```

- Pflichtangabe
- Optional

Klassendeklaration

```
class Book {
    String author;
    String title;

    Book(String title) {
        this.title = title;
    }

    void openPage(int page) {
        // code
    }
}
```

Klassen

- Eine public Klasse pro Source-File
- File wird nach der public Klasse oder Interface benannt
- mehrere nicht-public Klassen in einem File möglich

Compilieren / Ausführen

```
javac [options] [files]

java [options] class [args]
```

Ausführbare Klassen

- main-Methode nötig

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

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

- oder als varargs

```
public static void main(String… args)
```

- kann überladen werden

Pakete

- max. ein Paket pro Source-File
  - Beispiel:
    de.lubowiecki.oca
  - Korrespondiert mit Verzeichnisnamen

Import

- fully qualified names
- Imports verändern nicht die Größe der .class-Files
- * (nicht für Unterpakete)
- Statische imports

```
import static java.lang.System.out;
```

Variablen

- Instanzvariable
  - Jedes Objekt hat eigene
- Klassenvariable (statische Variable)
  - Alle Objekte der Klasse teilen sich die Variable

Methoden

- Instanzmethode
  - Arbeitet mit Instanz- und Klassenvariablen sowie mit Instanz- und Klassenmethoden
- Klassenmethode (statische Methode)
  - Arbeitet mit Klassenvariablen und Klassenmethoden

Konstruktor

- Erzeugt und Initialisiert ein Objekt
- Können überladen werden

Destruktoren

- protected void finalize()
- Speicher wird von Garbage Collector verwaltet
  - System.gc();
- Keine Sicherheit, dass finalize() aufgerufen wird

Modifikatoren (Access)

	public	protected	default (package)	private
eigene Klasse	ja	ja	ja	ja
Package	ja	ja	ja	
erbende Klasse	ja	ja		
fremde Klasse	ja			

Modifikatoren (Nonaccess)

- abstract
- static
- final

Modifikatoren (Nonaccess)

- synchronized
- native (Implementierung in einer anderen Programmiersprache und nutzbar über JNI, Java Native Interface)
- *transient (nicht persistent)*
- *volatile (wird im Threads beim Zugriff aktualisiert)*
- *strictfp (gleiche Genauigkeit auf allen Plattformen)*

Final Variablen

- Können nicht neu zugewiesen werden
- Bei Referenzen kann sich der Zustand des Objektes verändern

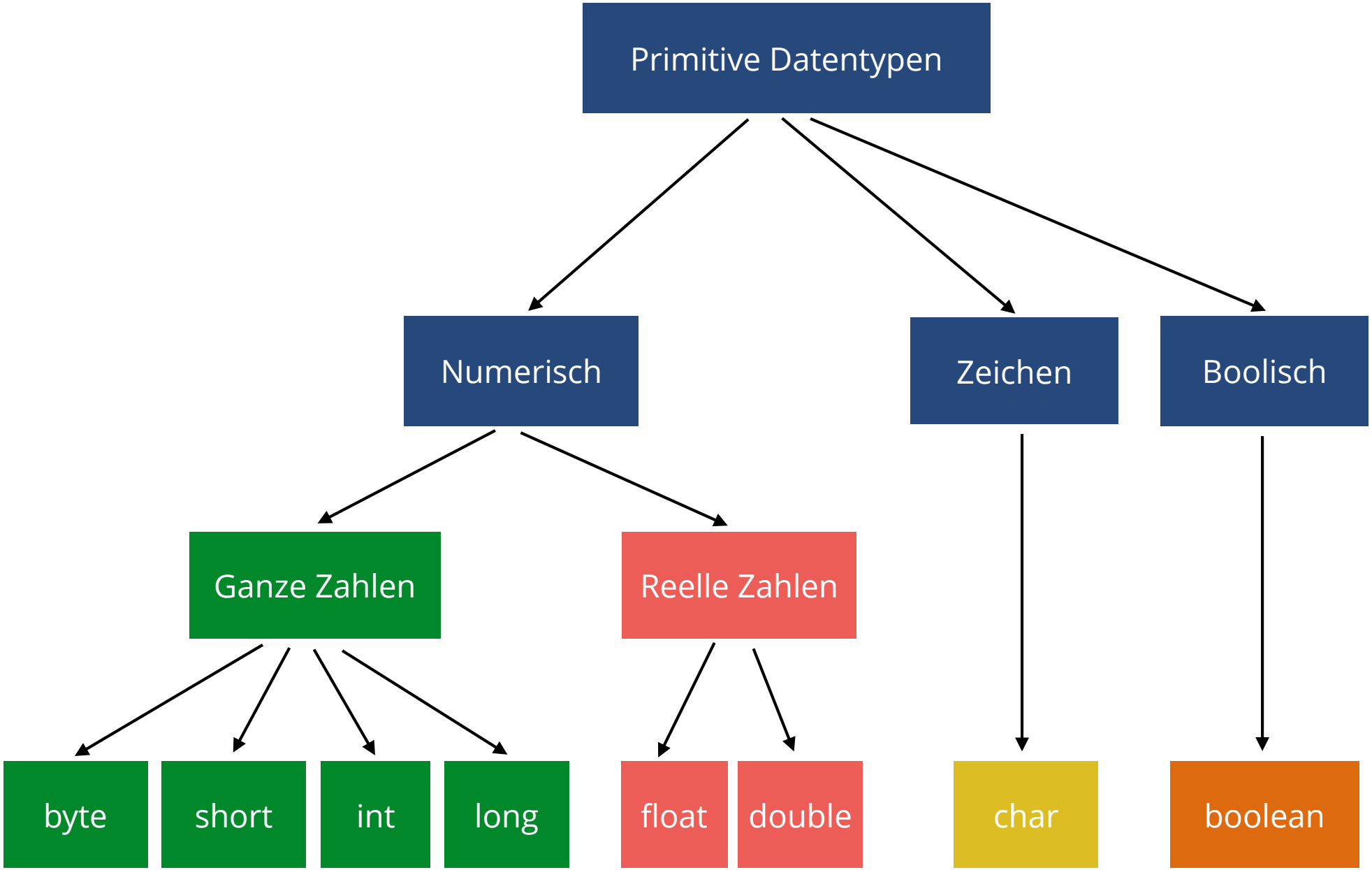
Statische Methoden

- Gehören der Klasse
- Können keine Instanzvariablen benutzen
- Können das statische Inventar der Klasse verwenden
- utility methods

Statische Variablen

- Gehören der Klasse
- static final

Datentypen


Primitive Datentypen
Numerisch
Zeichen
Boolisch
Ganze Zahlen
Reelle Zahlen
byte
short
int
long
float
double
char
boolean

Primitive Datentypen

- Vordefinierte Typen
- Keine Definition durch User möglich

Literale

- Ein fixer Wert der vor der Zuweisung zu einer Variable nicht umgerechnet werden muss

Boolean

- Typ: boolean
- 1 Byte
- Wertebereich: true, false
- Standardwert: false
- Keine Konvertierung zu und von anderen Typen möglich

Ganze Zahlen

- Typ: byte
- 8 Bit
- Wertebereich: -2^7 bis $2^7 -1$
  -128 bis 127
- Standardwert: 0

Ganze Zahlen

- Typ: short
- 16 Bit
- Wertebereich: -2^{15} bis $2^{15}-1$
  -32.768 bis 32.767
- Standardwert: 0

Ganze Zahlen

- Typ: int
- 32 Bit
- Wertebereich: -2^{31} bis 2^{31} -1
  -2.147.483.648 bis 2.147.483.647
- Standardwert: 0

Ganze Zahlen

- Typ: long
- 64 Bit
- Wertebereich: -2^{63} bis $2^{63}-1$
  -9.223.372.036.854.775.808 bis 9.223.372.036.854.775.807
- Standardwert: 0
- gefolgt von L oder l

Zahlen

- Binär (Basis 2) 0b / 0B
- Oktal (Basis 8) 0
- Dezimal (Basis 10)
- Hexadezimal (Basis 16) 0x / 0X
- _ kann zum gruppieren verwendet werden

Zahlen

```
int dec = 16;

int oct = 020;

int bin = 0b10000;

int hex = 0x10;
```

Underscore

- nicht am Anfang oder Ende des Literals
- nicht direkt nach 0, 0b, 0x
- bei long nicht direkt vor L
- nicht wo Zahlen als Strings erforderlich verwendbar
- Bei floats und doubles nicht direkt am Dezimalpunkt

Reelle Zahlen

- Typ: float (einfache Genauigkeit)
- 32 Bit
- Wertebereich:
  1,40239846E–45f bis 3,40282347E+38f
- Standardwert: 0.0
- gefolgt von F oder f
- Norm IEEE 754

Reelle Zahlen

- Typ: double (doppelte Genauigkeit)
- 64 Bit
- Wertebereich:
  4,94065645841246544E–324 bis
  1,79769131486231570E+308
- Standardwert: 0.0
- D oder d möglich

Reelle Zahlen (Ungenauigkeiten)

- Werden mehrere Berechnungen aufeinander folgend durchgeführt, so müssten die Zwischenergebnisse jedes Mal auf die Größe des jeweiligen Datentyps gerundet werden.

Zahlen

- ganzzahlige Werte standardmäßig vom Typ int
- Fließkomma Werte standardmäßig vom Typ double

Zeichen Datentypen

- Typ: char
- 16 Bit, Unicode Zeichen
- Wertebereich: alle Unicode - Zeichen
  von \u0000 (0) bis \uffff (65.535)
- Standardwert: \u0000

String

- Zeichenketten
- UTF16
- vergleich mit equals()

Array

- Container mit fester Größe
  - <Typ>[] <Variablenname>
- Zuweisung:
  - char[] a = {'a', 'b', 'c'};
  - a[1] = 'a';

Array

- int[] arr = new int[10];
- int arr[] = new int[10];

Mehrdimensionale Arrays

- int[][] arr = new int[10][10];
- int arr[][] = new int[10][10];

Typumwandlung

- Automatisch
  - byte < short < int < long < float < double
  - Immer zu größeren (mind. int)
  - Promotion
- Explizit (Casting)
  - (Typ) Variable
  - Verlust von Daten möglich

Deklaration

- Beschreibung einer Variable oder einer Klasse
- Es wird kein konkretes Objekt erzeugt und kein Speicher reserviert
- <Typ> <Name der Variable>;

Stack und Heap

- Instanzvariablen und Objekte liegen im Heap
- Lokale Variablen liegen auf dem Stack

Gültige Identifier

- Erlaubt
  - beginnt mit Buchstaben oder $ _
  - danach jede beliebige Kombination von Buchstaben, Zahlen und $ _

Gültige Identifier

- Nicht erlaubt
  - Zahlen am Anfang sind nicht erlaubt
  - Schlüsselwörter

Schlüsselwörter

- abstract, boolean, break, byte, case, catch, char, class, const, continue, default, do, double, else, extends, final, finally, float, etc.

Scope

- Lokale Variablen sind nur in dem Block erreichbar, in dem sie deklariert wurden

arithmetische Operatoren

- 1 Operand (als Vorzeichen), Prio 1
  - +, -
- 2 Operanden
  - *, /, % Prio 2
  - +, - Prio 3

Operatoren (für OCA)

- Zuweisungsoperatoren:
  =, +=, -=, *=, /=
- arithmetische Operatoren:
  +, -, *, /, %, ++, - -
- relationale-, und Vergleichsoperatoren:
  <, <=, >, >=, ==, !=
- logische Operatoren:
  !, &&, ||

short-circuit Operator (&&, ||)

```
if(x != null & x.length() > 0)
System.out.println(x.toUpperCase());

if(x != null && x.length() > 0)
System.out.println(x.toLowerCase());
```

AND: wenn der erste Operand zu false evaluiert kann das Endergebnis nie true sein

OR: wenn der erste Operand zu true evaluiert kann das Endergebnis nie false sein

Der zweite Operand wird nicht mehr evaluiert

Prezedenz

- Postfix
- Prefix und Vorzeichen
- *, /, %
- +, -
- <, >, <=, >=, instanceof
- ==, !=
- &, ^, |
- &&, ||
- =, +=, -=, *=, /=, %=

Numerische Promotion

- Wenn Werte unterschiedlicher Typen vorliegen, wird einer automatisch auf den größeren von beiden angepasst
- Wenn eine Ganzzahl und eine Gleitkommazahl vorliegen, wird die Ganzzahl automatisch in Gleitkommazahl konvertiert
- byte, short, und char werden zum Rechnen zu int konvertiert
- Der Ergebnis-Typ entspricht dem Typ der Operanden nach der Konvertierung zum gleichen Typ

Methods and Encapsulation

Lifecycle von Objekten

- beginnt mit der Initialisierung
- endet wenn das Objekt out of scope ist oder nicht mehr referenziert wird
- ist infrage kommend für den GC, wenn es nicht mehr erreichbar ist
- es kann nicht garantiert werden, dass ein unerreichbares Objekt vom GC abgeholt wird

Scopes von Variablen

- Klasse
- Instanz
- Methodenparameter
- lokal, sub-block

Scopes von Variablen

- Instanzvariablen
  - werden definiert und sind erreichbar innerhalb des Objektes
  - erreichbar für alle Instanzmethoden
- Klassenvariablen
  - werden von allen Objekten der gleichen Klasse gemeinsam genutzt

Methoden

- Code, die nur nach dem `return` ausgeführt werden soll führt zum Kompilerfehler
- `return` sollte die letzte Anweisung in einer Methode sein
- void-Methoden können ein `return` ohne nachfolgendem Wert enthalten
- Methoden mit Rückgabewert müssen ein `return` gefolgt vom Wert enthalten

Überladen von Methoden

- Argumentliste muss sich unterscheiden
  - Anzahl
  - Typen
  - Position
- Rückgabewert kann anders sein
- Zugrifsmodifikator kann anders sein
- Kann neue oder breitere Checked-Exceptions deklarieren

Konstruktoren

- heißen wie die Klasse
- definieren keinen Rückgabewert
- Default-Konstruktor nur vorhanden, solange kein neuer definiert wurde
- können public, protected, default oder private sein
- ist ein Rückgabewert vorhanden, dann gilt die Methode nicht mehr als Konstruktor

API

Strings

- Ist eine immutable Sequenz von Zeichen
- Strings die mit einem Literal erzeugt wurden landen im Pool
- Strings die mit `new` erzeugt wurden landen nicht im Pool
- `==` vergleicht die Referenzen
- `equals` vergleicht den Inhalt

StringBuilder

- Veränderbare Sequenz von Zeichen
- hat keine trim-Methode

Arrays

- Eine Sammlung von Werten
- Ist ein Objekt
- kann primitive oder komplexe Datentypen aufnehmen
- bei der Initialisierung muss die Größe vorgegeben werden

declaration, allocation, initialization

- declaration
  - typ variablenname und []
- allocation (mit new)
  - Speicher wird reserviert
  - Dimensionen und Größe
  - Standardwerte

Arrays

- Gültigkeit von Indexpositionen wird erst zu Laufzeit geprüft
  - ArrayIndexOutOfBoundsException
- Mehrdimensionale Arrays können asymmetrisch sein

ArrayList

- Gehört zum Collection framework
- Kann seine Größe nachträglich verändern
- Implementiert das List Interface
- akzeptiert `null`
- Erlaubt Duplikate
- Typsicherheit durch Generics

ArrayList

- intern ein Array von java.lang.Object
- Reihenfolge der Elemente bleibt erhalten
- Kann mit addAll Elemente anderer Listen aufnehmen
- **`clone`** erzeugt eine „Schattenkopie“

Objekte vergleichen

- `equals` aus java.lang.Object
- `equals` vergleicht in der Standardimplementierung ob die zwei Referenzen auf das gleiche Objekt zeigen
- `equals` muss überschrieben werden, wenn ein Vergleich auf basis von Instanzvariablen erfolgen soll

API contract

- `equals` auf `null` muss false zurückgeben
- `equals` darf die Instanzvariablen nicht verändern
- Zwei als gleich geltenden Objekte sollten den gleichen Hashcode liefern

API contract

- reflexiv
  - `x.equals(x)` ergibt true
- symetrisch
  - `x.equals(y)` ergibt nur dann true, wenn `y.equals(x)` auch true ergibt
- transitiv
  - wenn `x.equals(y)` true ergibt und `y.equals(z)` true ergibt, dann ergibt `x.equals(z)` auch true
- konsistent

Kontrollstrukturen

IF

- `if ( Bedingung ) { Anweisungen }`
- `if ( Bedingung ) { Anweisungen } else { Anweisungen }`
- `Erg = Bedingung ? Ausdruck1 : Ausdruck2;`
- Bedingung muss ein boolisches Ergebnis haben

Switch

```
switch (zeichen) {
  case '+': //add
  break;

  case '-': //sub
  break;

  case '/':
  case '*': System.out.println("cannot");
  break;

  default: System.out.println("illegal");
}
```

Schleifen

- ```
  for(int i=0; i < 10; i++) {
    //Anweisungen;
  }
  ```
- Initialisierung Zählervariable,
  Startbedingung,
  Zählen
  ```

Schleifen

- ```
  int[] arr  = {1, 2, 3};
  for(int i : arg) {
    //Anweisungen;
  }
  ```
  ```

Schleifen

- `while ( Bedingung )  { Anweisungen }`
- ```
  do { Anweisungen }
  while ( Bedingung ) ;
  ```
  ```

Vererbung

Klassen können...

- Eigenschaften und Methoden erben
- mehrere Interfaces implementieren

Vererbung

- In Java nur einfach
- Privates Inventar einer Klasse wird nicht vererbt
- Inventar mit default Zugriff kann nur geerbt werden, wenn beide Klassen im gleichen Paket liegen

Vererbung

- Subklasse kann auf ein Objekt der Superklasse zugreifen
- Superklasse kann nicht auf ein Objekt der Subklasse zugreifen
- Über eine Referenz der Superklasse können nur Eigenschaften und Methoden genutzt werden, die in der Superklasse definiert wurden
- Über eine Referenz eines Interfaces können nur Eigenschaften und Methoden genutzt werden, die im Interface definiert wurden

Vererbung

- Eine Referenz von Typ einer Superklasse oder eines Interface kann mit Objekten verschiedener abgeleiteter Klassen belegt werden

super und this

- sind Objektreferenzen
- this muss genutzt werden, wenn lokale Variablen die Instanzvariablen überdecken
- Konstruktoren können andere Konstruktoren über this() aufrufen
- super ist die Referenz auf ein Objekt der Superklasse
- super erlaubt den Zugriff auf Methoden oder Eigenschaften der Superklasse die in der Subklasse überschrieben wurden

super und this

- super() kann genutzt werden um in der Subklasse den Konstruktor der Superklasse aufzurufen

Interfaces

- Ein Interface muss vollständig implementiert werden, oder die Klasse muss abstract werden
- Ein interface kann beliebig viele Interfaces erweitern

Überschreiben von Methoden

- Zugrifsmodifikator darf nicht restriktiver sein
- Darf keine Checked-Exception definieren, die nicht in der Originalmethode definiert war
- Argumentenliste muss gleich bleiben
- Rückgabetyp muss gleich bleiben oder kompatibel sein

Interfacedefinition

```
interface Lesbar {
    //…
}
```

Interfacedefinition

- per default abstract
- alle Methoden per default public
- alle Variablen sind Konstanten,
  per default public static final
- Methoden dürfen nicht final sein
- kann nur Interfaces erweitern
- default und statische Methoden möglich

Abstrakte Klassen

- können nicht instanziert werden
- können abstrakte Methoden enthalten
- Interface ist abstract by default

Abstrakte Methoden

- haben keinen Body

Abstrakte Variablen

- gibt es nicht

Final Klassen

- Können nicht durch andere Klassen erweitert werden
- Interfaces können nicht final sein

Final Methoden

- Können in erbenden Klassen nicht überschrieben werden

Polymorphismus

- Existiert dort, wo eine Erb-Beziehung existiert und Methoden in der Super- und Subklasse gleiche Signatur haben
- polymorphische Methoden nennt man *overridden methods*

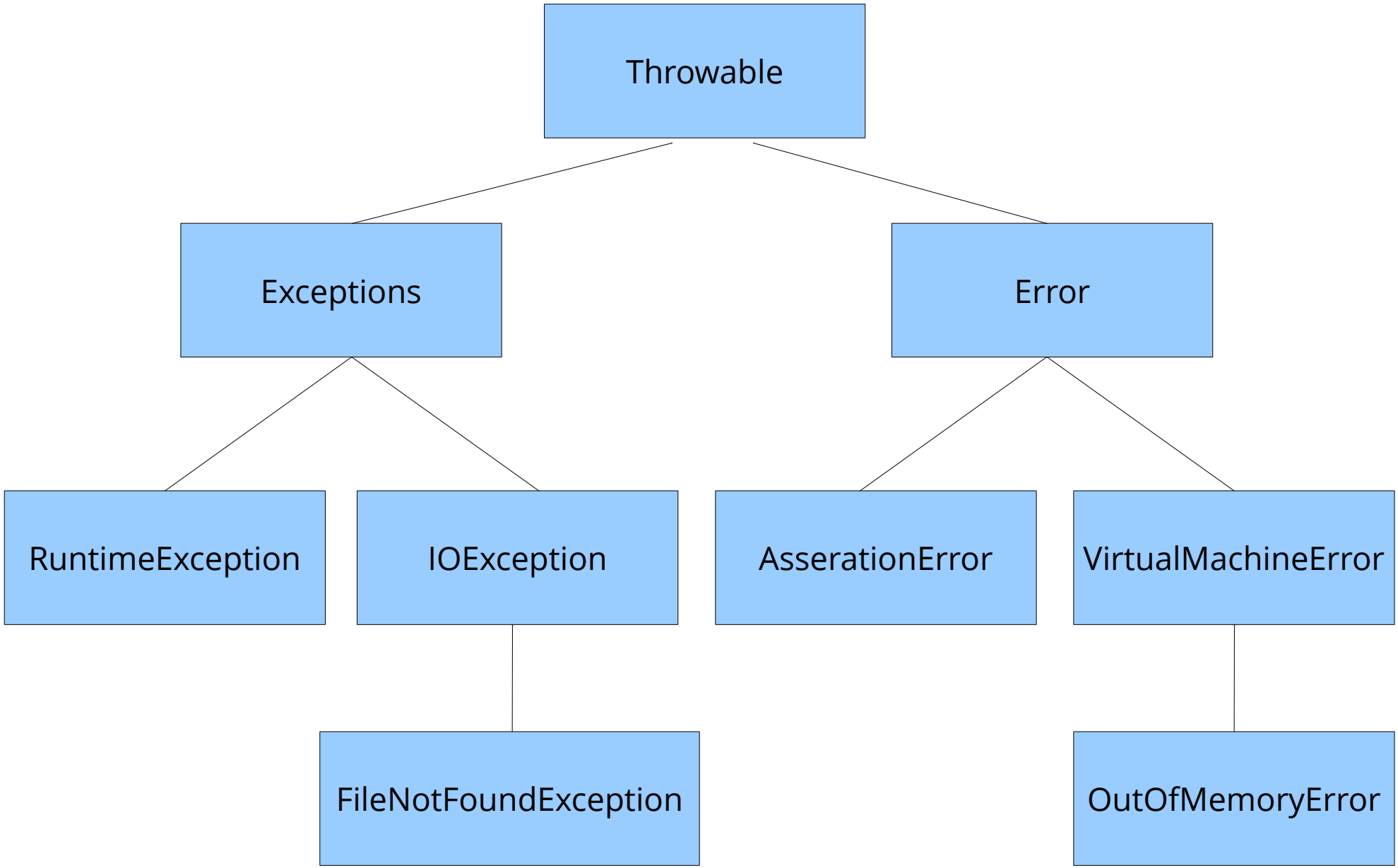
overridden methods

- die Methoden haben
  - gleichen Namen
  - gleiche Parameterliste
- return-Typ kann gleich oder eine Subklasse der original return-Typs sein (Kovariant)
- überladen reicht nicht
- Original-Methode kann `abstract` sein

Exceptions

Exceptions

- Ausnahmen
- Informationen über einen die Ausnahme
- Klassen, Basisklasse java.lang.Throwable
- 2 Arten:
  - checked (java.lang.Exceptions)
  - unchecked (java.lang.RuntimeException)


Throwable
Exceptions
Error
RuntimeException
IOException
AsserationError
VirtualMachineError
FileNotFoundException
OutOfMemoryError

Exceptions

- Nach Konstruktoren und Methoden einsetzbar

```
throws IOException [,.....] { }
```

Ausnahmen werfen

- Schlüsselwort `throw`

```
throw new InstanzDerException();
```

Exceptions behandeln

- try / catch

```
try {
    new FileInputStream(„xyz.txt“);
}
catch(FileNotFoundException e) {
    // Meldung
}
```

Mehrere behandeln

```
try {
   new FileInputStream(„xyz.txt“);
}
catch(FileNotFoundException e) {
   // Meldung
}
catch(IOException) {
   // Meldung
}
```

Multicatch

```
try {
   new FileInputStream(„xyz.txt“);
}
catch(FileNotFoundException | SQLException e) {
  // Meldung
}
```

...letzte Amtshandlung

```
try {
   new FileInputStream(„xyz.txt“);
}
catch(FileNotFoundException e) {
   throw new SpecialException();
}
finally {
   // wird ausgeführt
}
```

Exception weitergeben

- Rethrow
  - Fangen, behandeln und weitergeben

```
catch(FileNotFoundException e) {
   throw e;
}
```

Eigene Exceptions

- Von java.lang.Exception ableiten

```
public class SpecialException extends Exception{

  public SpecialException(String msg) {
    super(msg);
  }
}
```

```
throw new SpecialException(„Fehler“);
```