Angewandte Softwareentwicklung Java

WS 2014/2015

Markus Berg
Hochschule Wismar
Fakultät für Ingenieurwissenschaften
Bereich Elektrotechnik und Informatik

markus.berg@hs-wismar.de
http://mmberg.net

Vorwort

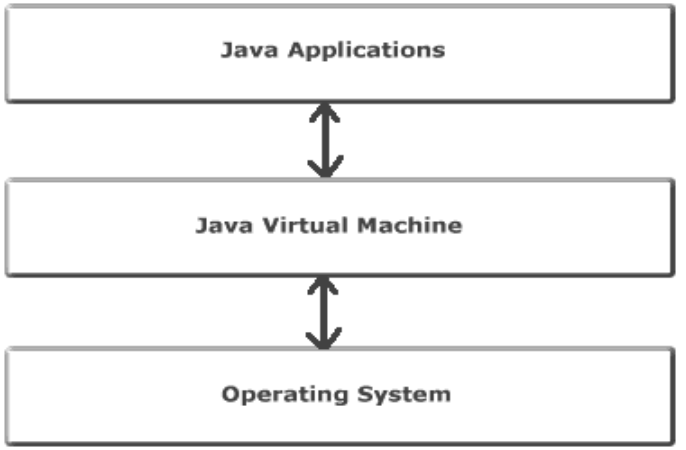
- Kurzes Kennenlernen der Sprache Java bzw. der Eigenschaften und Unterschiede zu anderen Sprachen
- Annahme: Sie kennen/können bereits:
  - Imperatives Programmieren
    - Datentypen, Fallunterscheidungen, Schleifen,…
  - Konzepte der Objektorientierung
    - Klassen, Vererbung,…
  - C/C++ oder Pascal/Delphi

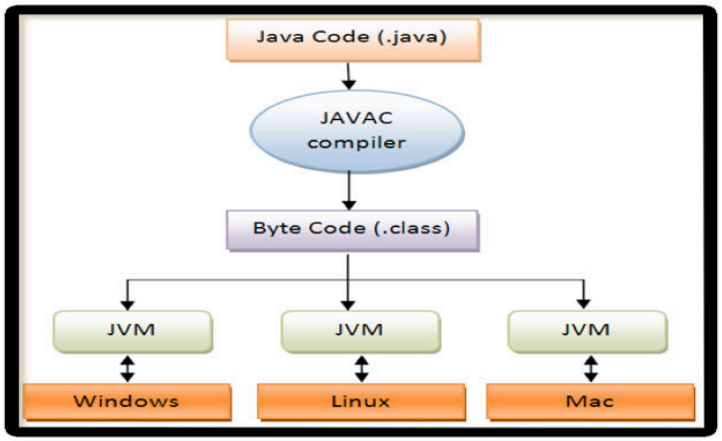
Eigenschaften

- Ehemals Sun, nun Oracle
- Objektorientiert
- Keine Pointer(nur intern)
- Erzeugt (kompiliert) keinen direkten Maschinencode (wie C) für eine Plattform (z.B. x86) und Betriebssystem (z.B. Linux) sondern Bytecode, der von einer VM interpretiert wird
  - vgl. MS .NET und CLR (Common Language Runtime)
- Diese JVM selbst ist nativ programmiert
- Plattformunabhängig und betriebssystemunabhängig durch JVM

Die Laufzeitumgebung

- Virtuelle Maschine + Bibliotheken
- VM führt den Bytecode (.class) aus
  - Classloader
  - Garbage Collection
- Just in Time Compilation (~Interpreter)
  - Kompiliert zur Laufzeit und puffert erzeugten Maschinencode
  - Arbeitet block- statt wie Interpreter zeilenweise
- Kapselt Zugriff auf das Betriebssystem durch APIs
- Kompiliertes Programm ist ohne Änderung auf Windows, Linux, Mac,... lauffähig

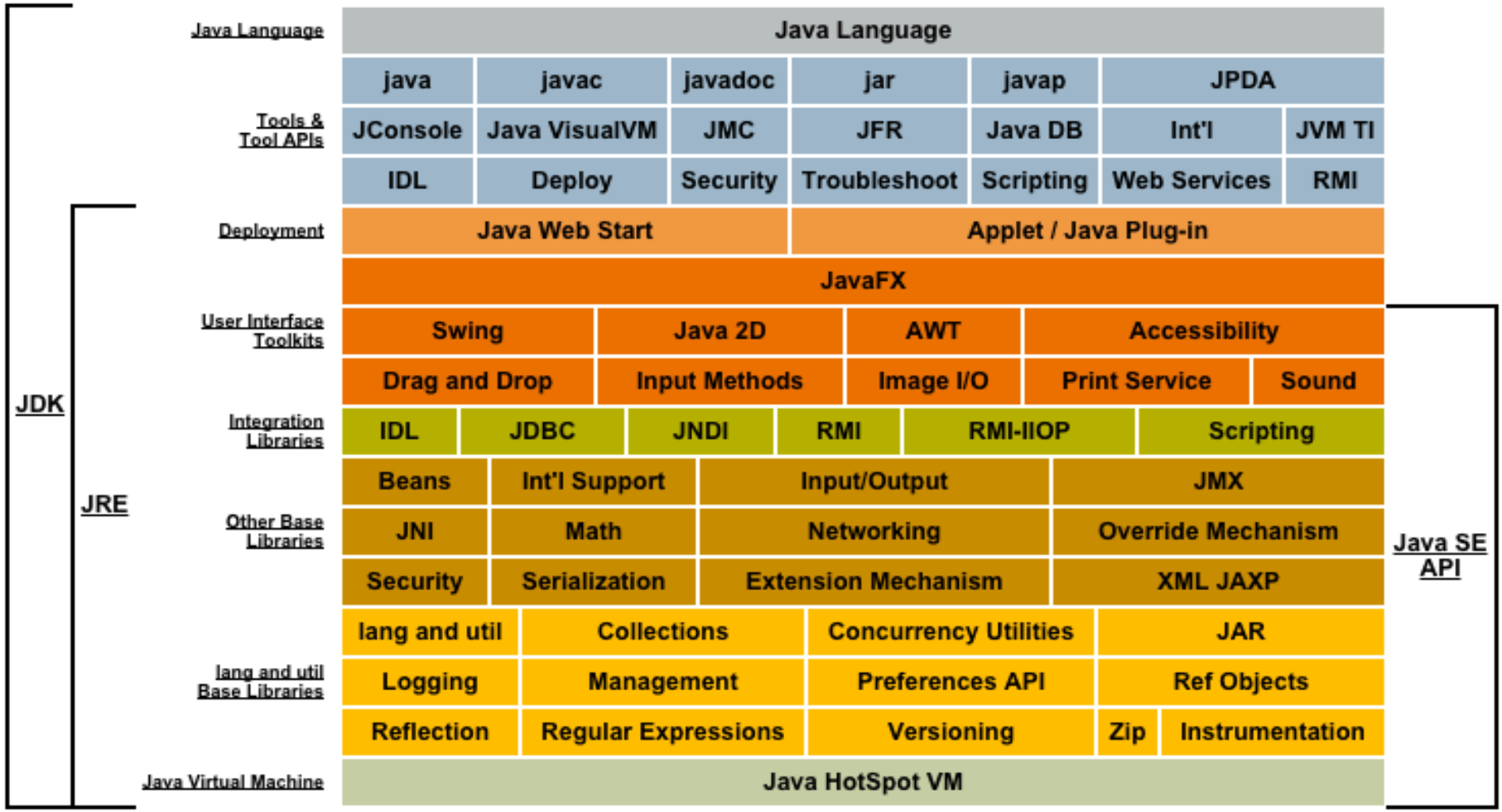




Garbage Collection

- Verwaltung von Objekten durch die Laufzeitumgebung
- Nicht mehr genutzte Speicherbereiche (wenn keine Referenz mehr existiert) werden automatisch freigegeben
- Objekte werden explizit angelegt (Konstruktor) und automatisch freigegeben

Java ist eine... Plattform!



Java Varianten

- Java SE (Standard Edition)
- Java EE (Enterprise Edition)
  - Servlets
  - Java Server Pages (JSP)
- Java Applets (Java im Browser, d.h. Client)
- Java ME (Micro Edition): kaum Bedeutung
  - Stattdessen: Android mit Dalvik VM
- ~~Java Script~~ (Skriptsprache von Netscape)

Datentypen

- Einfache (feste Länge!)
  - boolean (true, false, kein numerisches Äquivalent)
  - Ganzzahlen
    - byte (8 bit, vorzeichenbehaftet, -128..127 statt 0..255)
    - short (16 bit)
    - [char (16 bit, vorzeichenlos, z.B. Unicode-Zeichen)]
    - int (32 bit)
    - long (64 bit)
  - Fließkomma
    - float (32 bit)
    - double (64 bit)

Datentypen

- Komplexe (Referenztypen, Objekte)
  - String
  - Boolean
  - Integer
  - Float
  - ...
  - Eigene Klassen

Kontrollstrukturen

- Schleifen
  - Kopfgesteuert: `while(x) do{...}`
  - Fußgesteuert: `do{...} while(x)`
    = Pascals `repeat until (!x)`
  - For: `for(int i=0; i<bedingung; i++){...}`
    - Foreach: `for(Element e : list){...}`

- Bedingungen
  - ```
    if(condition){
    ...
    }
    else if(condition) {
    ...
    }
    else{
    ...
    }
    ```

Dangling else:

```
if (condition1)
  if (condition2)
      instruction;
else
  instruction;
```

→ Klammern!
```

Methoden

- Signatur
  - Rückgabewert (Datentyp bzw. void)
  - Name
  - Parameterliste
- Java hat keine Header-Files o.Ä.
- Parameterübergabe immer call-by-value (ABER… später mehr)
- Methoden überladen: gleicher Name, aber:
  - Unterschiedliche Anzahl an Argumenten oder
  - Unterschiedliche Datentypen der Argumente

Main-Methode

- Wird beim Starten der Klasse aufgerufen
- Meist eine main-Methode in gesamter Applikation
- Muss statisch sein, um vor Instantiierung geladen werden zu können

```
public static void main( String[] args )
{
  System.out.println(“Moin Welt“);
}
```

Packages und Imports

- Thematisch zusammengehörige Klassen werden in Paketen zusammengefasst
  - Vollständige Qualifizierung: Paketname + Klassenname
  - Erlaubt gleiche Klassennamen in verschiedenen Paketen
  - Erste Zeile jeder Klasse (`package de.mmberg.ase;`)
- Namenskonvention: Domains rückwärts
  - z.B. de.mmberg.ase (statt ase.mmberg.de)
  - Hieraus ergibt sich i.d.R. Kleinschreibung
  - Jeder durch Punkt getrennte Namensbestandteil ist im Dateisystem ein Ordner, d.h. de/mmberg/ase
- Um auf Klassen eines Paketes zugreifen zu können, muss das Paket importiert werden
  - Einzelne Klasse: `import paket.subpaket.klasse`
  - Alle Klassen des Pakets: `import paket.subpaket.*`
  - Erleichtert die Nutzung von Klassen, da auf wiederholte volle Qualifizierung verzichtet werden kann (solange die Klasse eindeutig ist)

Classpath

- Führt alle Pfade auf, in denen nach Klassen gesucht wird, die im Programm benötigt werden
  - Bibliotheken (*.jar) müssen explizit angegeben werden
- Genutzt vom Compiler

Kommandozeile

- Kompilieren
  - `javac meineKlasse.java`
  - Ergebnis: meineKlasse.class
  - mit Bibliotheken
    - `javac –classpath lib/mylib.jar meineKlasse.java`
  - bei Packages
    - `javac -classpath „.“ de.mmberg.ase.MoinWelt.java`
- Ausführen
  - Starten der Main-Methode
  - Datei
    - Ohne Dateiendung!
    - `java meineKlasse`
  - Archiv
    - `java –jar meinArchiv.jar`

Modifier

- Sichtbarkeitsmodifikatoren (für Methoden und Variablen)
  - Public: alle
  - Private: nur innerhalb der Klasse
  - Protected: nur innerhalb des Paketes bzw. bei abgeleiteten Klassen
  - Kein Modifier: nur innerhalb des Paketes
- static
  - Klassenvariablen vs. Instanzvariablen
  - Statische Eigenschaften gelten für alle Instanzen einer Klasse
  - Es ist keine Instanziierung erforderlich, direkt über Klassennamen ansprechbar
    - z.B. `Math.xxx();`
    - statt `Math m =new Math(); m.xxx();`
  - Genutzt für Variablen und Methoden die unabhängig von einer konkreten Instanz sind, z.B. `Math.max(int a, int b)`

Modifier

- final
  - unveränderliche Variable (verbietet weitere Zuweisungen, initiale Zuweisung kann auch NACH Deklaration stattfinden)

```
final int a;
...
a=3;
```

  - unveränderliche Methode (darf nicht in Subklasse überschrieben werden)
- static final
  - Wird zur Deklaration von Konstanten benutzt

>> *http://twbk.de/ASE14*

Sollte eine Methode moveForward() der Klasse Player in einem Multiplayer-Spiel static sein?

A	Ja
B	Nein

Ausgabekanäle

- System.in
  - Stdin
  - Vereinfachtes Einlesen durch Scanner

```
Scanner scanner = new Scanner(System.in);
String s = scanner.next();
```

- System.out
  - Stdout
  - `System.out.println(„Moin Welt“);`
- System.err
  - stderr

Projektstruktur

- src (Quellcode)
  - mit Packages
- test (später mehr dazu)
- bin (Kompilat)
- lib (Bibliotheken)

Auffrischung: Objektorientierte Programmierung

- Grundgedanke
  - Entitäten der Welt: Klasse („ich bin“)
    - Auto, Frucht, Mediengerät
    - Konkrete Ausprägung einer Entität: Instanz/Objekt
      - Auto von Peter, Apfel der hier liegt, Mediengerät von Susi
  - Aktionen/Operationen: Methoden („ich kann“)
    - Beschleunigen, reifen, abspielen
  - Attribute („ich habe“)
- Zweck
  - Wiederverwendbarkeit

OO-Konzepte (Ausschnitt)

- Abstraktion
  - Unterscheidung zwischen Konzept (Bauplan/Implementierung → Klasse) und Nutzung des Konzeptes (Objekt)
    - Bei Nutzung muss ich Bauplan nicht kennen
- Kapselung
  - Verhalten und Daten/Eigenschaften zu einem Objekt mit Methoden und Attributen kombinieren
- Vererbung
  - Spezialisierung (is-a)
    - Auto > Audi, Frucht > Apfel, Mediengerät > BluRayPlayer
- Polymorphismus
  - Überladen
    - Gleicher Methodenname, unterschiedliche Parameter
    - Teilweise Unterscheidung nur im Datentyp
  - Überschreiben
    - Geerbte Methode lokal überschreiben und anders implementieren

Klassen

- Klassen definieren Typen
- Instanzen eines Types => Objekte
  - Erzeugt mit dem new-Operator
    - Lädt den Konstruktor
      - Signatur: Klassenname (Parameter1,..,Pn)
        - Kein Rückgabetyp erforderlich, da immer eine Instanz der Klasse zurückgegeben wird
        - Bzw. genauer eine „Referenz“ auf das neue Objekt
        - Standardkonstruktor hat keine Argumente
- Datei muss so heißen wie Klasse
  - public MeineKlasse{...}
    - MeineKlasse.java
      - MeineKlasse.class

Vererbung

- Hierarchie von thematisch ähnlichen Klassen, die mit zunehmender Hierarchietiefe konkreter werden
  - Spezialisierung
  - Fortbewegungsmittel > Fahrzeug > Auto > 4Tuerer > Audi > A6
    - Instanz
      - `Fortbewegungsmittel meinA6 = new A6();`
      - `A6 meinA6 = new A6();`
- Schlüsselwort `extends`
  - `public class Fahrzeug extends Fortbewegungsmittel{...}`
- Keine Mehrfachvererbung
  - Wenn zwei Oberklassen die gleiche Methode implementieren, weiß die Subklasse nicht welche gemeint ist

Interfaces

- Möglichkeit, um trotz fehlender Mehrfachvererbung einer Klasse mehrere Typen zuweisen zu können
  - Triangle extends GeomObject implements 2D
  - Audi extends Car implements RentalCar
    - getPricePerMile()
    - getFreeMiles()
- Gibt Liste von Methoden bzw. deren Signatur vor
  - sgn. Abstrakte Methoden
  - Implementierung erfolgt in der aktuellen Klasse und wird nicht aus der Superklasse geerbt
- Keine Instanziierung möglich
- Interfaces werden über das Schlüsselwort `implements` eingebunden (nicht `extends`)
- Eine Klasse kann beliebig viele Interfaces haben
- Interface ohne Methoden: Marker Interface (Prüfung des Typs mit `instanceof`)

Abstrakte Klassen

- Eine Klasse ist abstrakt sobald sie eine abstrakte Methode besitzt
- Abstrakte Klassen können nicht instanziiert werden
- Abstrakte Methode
  - Definition der Signatur
  - Keine Implementierung
    - Erfolgt in Subklasse je nach konkretem Typ
    - Klasse GeomObj definiert abstrakte Methode getSurface()
      - Triangle extends GeomObject
      - Square extends GeomObject
      - Beide Klassen müssen die Methode besitzen, implementieren sie jedoch unterschiedlich

>> *http://twbk.de/ASE14*

Ist es problematisch wenn eine Klasse mehrere Interfaces implementiert, die die gleiche Methode anbieten?

A	Ja
B	Nein

Tücken bei der Parameterübergabe

- Aussage: „alles call-by-value“
- ABER: Kopie eines Wertes (primitiver Datentyp) vs. Kopie einer Referenz (Objekt)
  - Java kennt tatsächlich kein call-by-reference
  - D.h. alle Parametervariablen sind immer lokale Kopien
  - Man könnte aber unterscheiden zwischen call-by-value-with-a-primitive und call-by-value-with-a-reference

```
static void main(){
  int a=3;
  addOne(a);
  System.out.println(a); //3
}

private void addOne(int a){
  a++;
}
```

```
static void main(){
  Integer a=new Integer(3);
  addOne(a);
  System.out.println(a); //4
}

private void addOne(Integer a){
  a++;
}
```

```
static void main(){
  Integer a=new Integer(3);
  doCrazy(a);
  System.out.println(a); //3
}

private void doCrazy(Integer a){
  a=new Integer(8);
}
```

Referenzen

- this
  - Referenziert die eigene Instanz
  - z.B. um globale Variablen (Objektvariablen) aus der eigenen Klasse bei Namensgleichheit mit lokalen Variablen referenzieren zu können

```
Integer a;

void doSth(Integer a){
  if(this.a!=null) this.a=a;
}
```

- super
  - Referenziert die Instanz der Oberklasse

Getter und Setter (Java Beans)

- Definiertes Design einer Klasse
  - Lokale Variablen als *private* deklarieren
  - Getter und Setter für lokale Variablen
    - Zugriff beschränken
      - Andere Objekte können nicht auf Variable direkt zugreifen
      - Ändern von Variablen durch andere Objekte verhindern durch *private Setter*
    - Zusätzliche Funktionalitäten zur Konsistenzwahrung
      - z.B. Prüfen auf null oder leeren String
    - Benennung: z.B. für Variable *varname*
      - **set**Varname
      - **get**Varname bzw. **is**Varname (wenn boolean)

```
private String name;

public String getName(){
  return this.name;
}

public String setName(String
name){
  this.name=name;
}
```

Exceptions

- In C sieht man oft, dass Rückgabewerte als Fehlerindikator missbraucht werden
  - eigentlicher Rückgabewert über Pointer

```
#include <stdio.h>

int divide(int *result, int
 value1,int value2){
 if (value2==0) return -1;
 else{
  *result=value1/value2;
  return 1;
 }
}

int main(){
 int res;
 divide(&res,4,2);
 printf("%d",res);
 return 0;
}
```

Exceptions

- In Java werden Fehler über Exceptions gemeldet
  - Rückgabewerte sind immer „echte“ Rückgabewerte
- Exceptions werden „geworfen“
  - `throw new CustomException(„Das war wohl nix“);`
- Exceptions können „gefangen“ werden durch catch-Blöcke
  - Genauer gesagt try..catch
  - Try: Block in dem potentiell ein Fehler auftreten kann
  - Catch: Reaktion auf den Fehler
    - Im einfachsten Fall Ausgabe der Fehlermeldung

Exceptions werfen und fangen

```
doFunnyStuff(Integer a){ throws NegativeArgumentException
  if(a<0) throw new NegativeArgumentException();
}

public class NegativeArgumentException extends Exception{
  public NegativeArgumentException(){
    super(„Das Argument ist zu klein“)
  }
}
```

```
try{
  doFunnyStuff(-2);
}
catch(NegativeArgumentException ex){
 System.out.println(ex.getMessage()); //Das Argument ist zu klein
 //oder: ex.printStackTrace(); //mehr Infos über Fehlerquelle
}
```

→ Wenn Exceptions nicht lokal gefangen werden, müssen sie von der aufrufenden Methode gefangen werden („nach oben reichen“) oder erneut weitergereicht werden

Generics

- Java ist typisiert, d.h. jede Variable hat einen bestimmten Datentyp
  - z.B. ein Name (String), eine Geburtsdatum (Date),...
  - Oder auch eine Einkaufsliste (List), Liste mit Matrikelnummern (List) oder Liste mit Büchern (List)
- Listen
  - Spezielle Funktionen wie Elemente zählen oder Elemente hinzufügen etc.
  - Ein Element der Liste ist aber vom Typ Object
    - Somit gehen sämtliche Eigenschaften verloren (ohne expliziten Cast)
    - Die Liste weiß nichts von Integer, Date, String, Buch,...
  - Daher Inhalt der Liste typisieren: List<String>, List<Buch>,...
- Typen die ihre Inhaltstypen typisieren können nennt man Generics
  - Nicht nur bei Listen
  - z.B. Maps,...

Maps & Lists

- Über Listen iterieren (for each e in liste)

```
List<Elem> liste;
for (Elem e : liste){
 ...
}
```

- Zwei wichtige Vertreter
  - ArrayList
  - HashMap

Annotations

- Benutzerdefinierte Metadaten
- Annotationen beginnen mit @
- Jeweils vor die Signatur einer Methode bzw. vor einem Parameter oder vor einer Klasse
- z.B.
  - @Override
    - Zeigt an, dass diese Methode eine Methode der Oberklasse überschreibt
  - @WebService
    - Zeigt an, dass die Methode als Webservice-Methode veröffentlicht wird
    - Weitere Attribute geben den Namen, Endpunkt etc an
    - @WebService(Name=„getWeather“)
  - @Deprecated
    - Zeigt an, dass die Methode veraltet ist und bald aus der API entfernt wird
  - @Test
    - Zeigt an, dass es sich um einen Testcase handelt
  - @XmlRoot
    - Zeigt an, dass die Klasse das Root-Element bei der XML-Serialisierung ist

Hinweise

- Kleinschreibung bei main-Methode wichtig!
- String-Vergleiche mit .equals statt „==“
- String-Verkettung mit „+“ („a“+“b“=„ab“)
- Seit Java 7 endlich switch mit Strings möglich
  - ```
    switch(stringVar)
    {
     case „a“: doSthFunny(); break;
     case „b“: ...
    }
    ```
- Mehrere Rückgabewerte müssen über eigene Klassen gekapselt werden
- Methoden mit einer variablen Anzahl an Parametern sind möglich
  - ```
    void doSth(int... params){ //die Punkte sind ernst gemeint
      for (int param:params){/* Anweisungen */}
    }
    ```

IDEs

- Eclipse
- Netbeans (nutzen wir im Praktikum)
- JDeveloper
- …

Quellen und weiterführende Literatur

- http://openbook.galileocomputing.de/javainsel/
- http://www.applicationserverinfo.com/images/jvm-intro1.jpg
- http://www.nosid.org/java-call-by-value-reference.html
- http://docs.oracle.com/javase/7/docs/api/overview-summary.html

Quiz

>> *http://twbk.de/ASE14*

Was zeichnet eine abstrakte Klasse aus

A	Keine Methode ist implementiert
B	Mindestens eine Methode existiert nur als Signatur
C	Von ihr lässt sich nicht erben

>> *http://twbk.de/ASE14*

Java wird direkt in nativen Code kompiliert. Richtig oder falsch?

A	Richtig
B	Falsch, denn Java-Code wird interpretiert und nicht kompiliert.
C	Falsch, denn Java-Code wird zunächst in Bytecode kompiliert

>> *http://twbk.de/ASE14*

In Java muss ich explizit Speicher wieder freigeben.

A	Richtig
B	Falsch

>> *http://twbk.de/ASE14*

Wie lautet die Dateiendung einer kompilierten Java-Datei?

A	.exe
B	.out
C	.java
D	.class

>> *http://twbk.de/ASE14*

Ist das möglich?

```
int i=2;
System.out.println(i.toString());
```

A	Ja
B	Nein

>> *http://twbk.de/ASE14*

Ist a in KlasseB sichtbar?

```
public class KlasseA{
        public int i;
        protected int a;
}

public class KlasseB extends KlasseA{
}
```

A	**Ja**
B	**Nein**

>> *http://twbk.de/ASE14*

Was ist das Ergebnis des Vergleiches `a.toLowerCase()==b` ?

```
String a=„HALLO“;
String b=„hallo“;
```

A	**true**
B	**false**
C	**Fehler**

>> *http://twbk.de/ASE14*

Was wird ausgegeben?

```
public class Car{
  int speed=0;

  public int accelerate(){
    return ++speed;
  }
}

public class Race{
 public void start(){
  Object o = new Car();
  System.out.println(o.accelerate());
 }
}
```

A	0
B	1
C	Fehler

>> *http://twbk.de/ASE14*

Was ist das Ergebnis von a+b ?

```
String a=„3“;
String b=„4“;
```

A	**7**
B	**34**
C	**Fehler**

>> *http://twbk.de/ASE14*

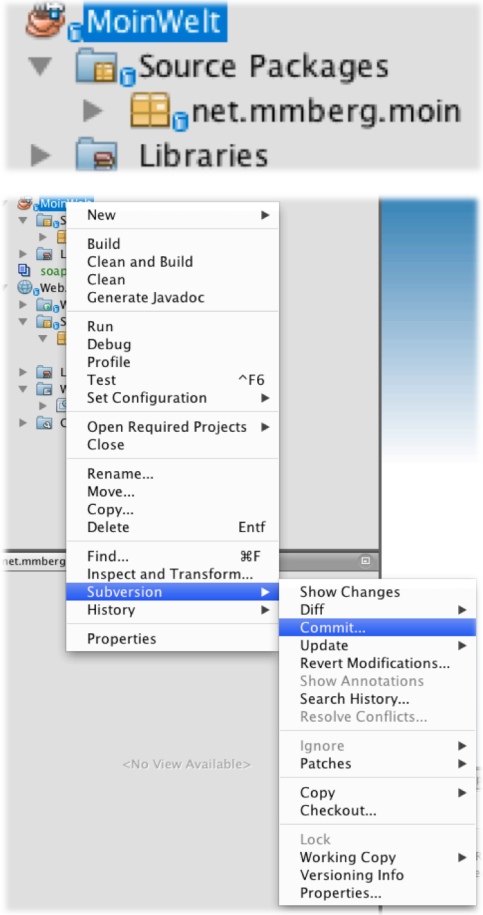
Kompiliert der Code?

```
Object o = new Boat();
Car c = (Car) o;
```

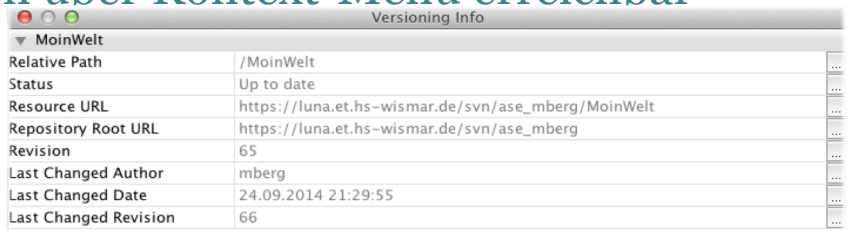
A	**ja**
B	**nein**

Übungen

Exkurs: Anlegen eines Projektes und Anbindung an SVN mit Netbeans



- Neues Projekt anlegen (Java / Java Application)
  - Namen und Pfad eingeben (z.B. MoinWelt)
  - Legen Sie das Projekt in dem Ordner an, wo sie Ihr Repository ausgecheckt haben
    - Es wird automatisch ein neuer Unterordner mit dem Projektnamen ausgewählt
- Netbeans sollte automatisch erkennen, dass der Ordner ein Repository ist
  - Dies ist zu erkennen an dem blauen „Datenbank-Symbol“
  - Auf Nachfrage Login eingeben
  - SVN Funktionalitäten über Kontext-Menü erreichbar
  - Kontrolle: Versioning Info



Übung: Moin Welt (und etwas mehr)

```java
package net.mmberg.moin;

import java.util.Scanner;


/**
 *
 * @author Markus
 */
public class MoinWelt {

    public static void main(String[] args) {
        System.out.println("Moin Welt! Sag was!");
        Scanner scanner = new Scanner(System.in);
        String input = scanner.next();
        System.out.println("Du hast "+input+ " gesagt.");
    }

}
```

Übung: Vererbung

- Wir wollen folgenden Sachverhalt modellieren
- Es gibt verschiedene Formen von Speichermedien
  - Harddisks und Compact Discs
- Speichermedien haben ein Dateisystem und sind portabel/importabel
- Außerdem kann von jedem Speichermedium die Kapazität erfragt werden

Übung: Vererbung

- Abstrakte Klasse Speichermedium
  - `enum MOBILITY{portable, importable}`
  - Attribute `String dateisystem` und `MOBILITY mobility` mit Gettern und Settern
  - Abstrakte Methode `getCapacity()`
  - Konstruktor mit Parameter Filesystem
- Klasse CompactDisc erbt von Speichermedium
  - Attribut `capacity`
  - Konstruktor mit Attribut `capacity`
  - Konstruktor ruft super auf mit Filesystem ISO9660 und setzt Mobility auf portable
- Klasse Harddisk erbt von Speichermedium
  - Attribute cylinders, sectors und heads
  - Getter und Setter
  - Größe errechnet sich dynamisch aus deren Produkt
  - Konstruktor mit Filesystem du den o.g. Attributen

Übung: Vererbung

- Wir können nun aus der main-Methode, die in einer weiteren Klasse (hier: Kasse) implementiert ist, Speichermedien erzeugen

```
Speichermedium meineNeuePlatte = new Harddisk("NTFS",300,20,2);
Speichermedium meineCD = new CompactDisc(700);

System.out.println(meineNeuePlatte.getCapacity());
System.out.println(meineCD.getCapacity());
```

- `meineNeuePlatte` und `meineCD` sind Instanzen verschiedener Klassen, jedoch sind beide vom Typ Speichermedium und bieten somit die Methode getCapacity an
- Wenn wir explizit eine Harddisk anlegen, sehen wir mehr Methoden (z.B. getSectors)

```
Harddisk meineNeuePlatte = new Harddisk("NTFS",300,20,2);
```

Übung: Vererbung

- Wir legen jetzt eine Klasse an, die nichts mit Speichermedien zu tun hat
  - z.B. Apfel
  - Diese Klasse hat lediglich ein Attribut zur Speicherung der Apfelsorte
- Nun möchten wir Produkte verkaufen, z.B. Speichermedien und Äpfel. Jedes Produkt hat einen Preis und eine Bezeichnung.
- Wir legen das Interface Produkt mit den Methoden getPrice und getName an und weisen es sowohl der Apfelklasse als auch der Klasse Speichermedium zu
- Hier müssen nun die abstrakten Methoden implementiert werden
  - Dies erfordert das Anpassen der Konstruktoren
  - ...und das Anlegen neuer Attribute

Übung: Vererbung

- Wir simulieren nun ein Geschäft und einen Kassenzettel
- Der Einkaufskorb wird über `HashMap<Product,Integer> products;` modelliert
- Das Einkaufen wird über das Hinzufügen eines Produktes zur Map abgebildet

```
private void einkaufen(Product p, int quantity){
        products.put(p, quantity);
    }
```

- Wir kaufen ein:

```
products=new HashMap<>();
einkaufen(meineNeuePlatte,1);
einkaufen(meineCD,3);
einkaufen(meinApfel,5);
```

Übung: Vererbung

- Nun soll ein Kassenzettel gedruckt werden
- Da alle unsere Produkte das Interface `Product` implementiert haben, müssen sie auch die Methoden `getPrice` und `getName` besitzen
- Die Menge beziehen wir aus der Map
- Somit stehen alle Daten bereit um den Kassenzettel zu drucken:

```
private void printKassenzettel(HashMap<Product,Integer> products){
        for(Map.Entry e : products.entrySet()){
            Product p = (Product)e.getKey();
            Integer q = (Integer)e.getValue();
            System.out.println(p.getName() + ": "+q+"*"+p.getPrice()
                  +"="+q*p.getPrice());
        }
    }
```