ISW: Software Engineering WS 2015/16

Java: Grundlagen und Beispiele

Marcus Seiler
Institute of Computer Science
Chair of Software Engineering
Im Neuenheimer Feld 326
69120 Heidelberg, Germany
http://se.ifi.uni-heidelberg.de
marcus.seiler@informatik.uni-heidelberg.de

Inhalt

- Teil 1: Überblick
  - Erstkontakt
  - Java Standard Edition
  - Objektorientierte Programmierung (OOP)
  - Unterschiede zwischen C++ und Java
- Teil 2: Java Basics
  - Klassen
  - Felder
  - Operationen
  - Konstruktoren
  - Vererbung
  - Objekte
  - Ausnahmen
  - Java API
  - Collections

Erstkontakt

- Java?
  - Ist mit der Hauptstadt Jakarta die bedeutendste der indonesischen Inseln
  - Tätige und erloschene Vulkane bilden die zentrale Längsachse
  - Eines der wichtigen Exportprodukte der im 17. Jahrhundert entstandenen Plantagenwirtschaft ist der Kaffee

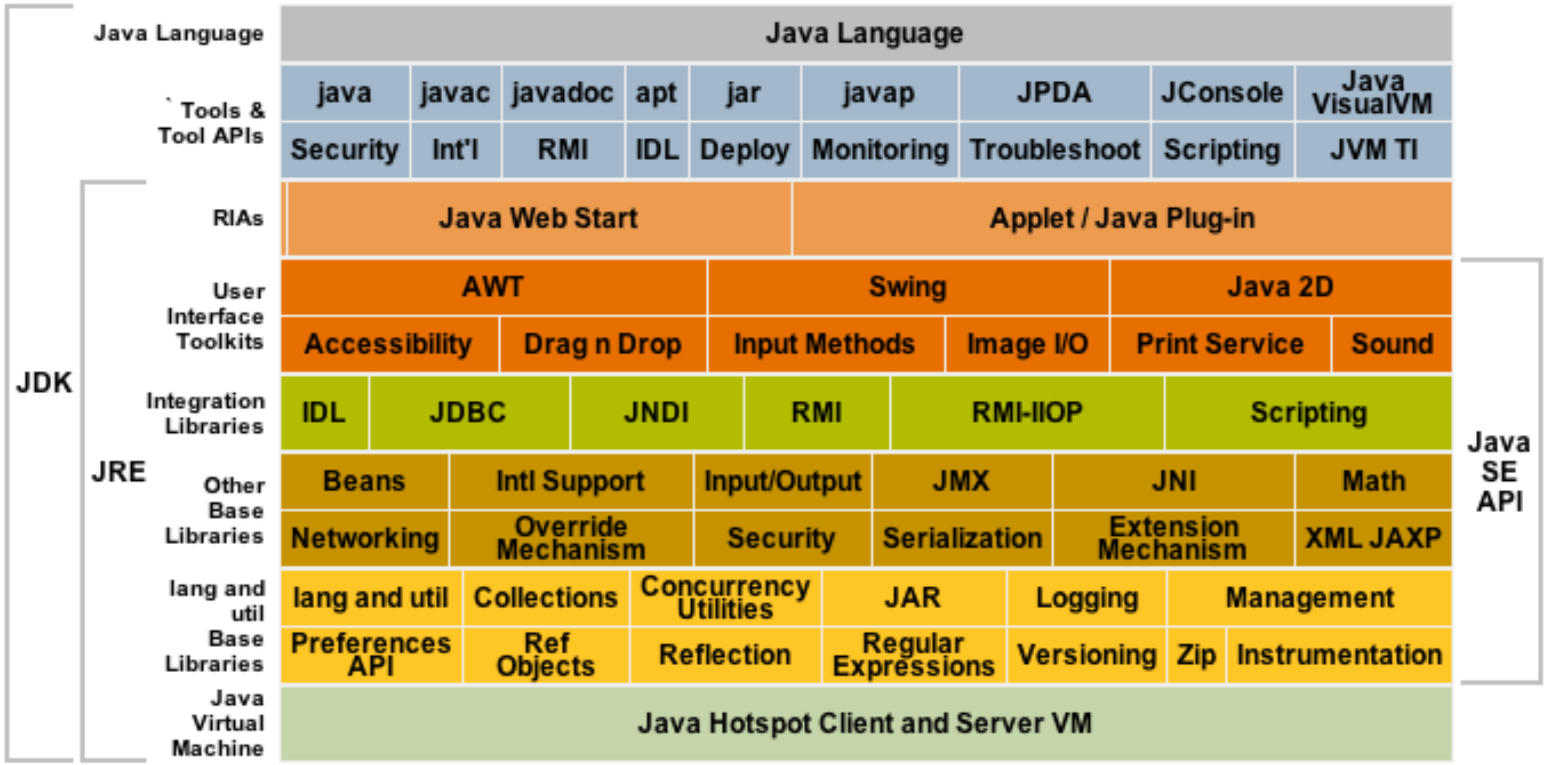
Erstkontakt

- Was ist Java(tm) noch?
  - Eine Plattform
  - Eine Programmiersprache
- Wer ist der Hersteller?
  - Ursprünglich Sun Microsystems
  - Inzwischen von Oracle aufgekauft
- Website?
  - http://www.java.com/
  - http://www.oracle.com/technetwork/java/index.html
  - …

Erstkontakt

- Was bedeutet Plattform?
  - Eine hardware-spezifische Umgebung in der Anwendungen ausgeführt werden
- Welche Java 2 Plattformen gibt es?
  - Enterprise Edition (JEE)
    → Anwendungen für Server
  - Standard Edition (JSE)
    → Anwendungen für Desktops
  - Micro Edition (JME)
    → Anwendungen für Mobile Kommunikation
  - Java Card
    → Anwendungen für Smartcards
- Was kennzeichnet die Plattformen?
  - Plattform-spezifische Klassenbibliothek
  - Plattform-spezifischer Interpreter (Java Virtual Machine)
- Allerdings: plattform-übergreifende Programmiersprache und Compiler

Components and Technologies of Java



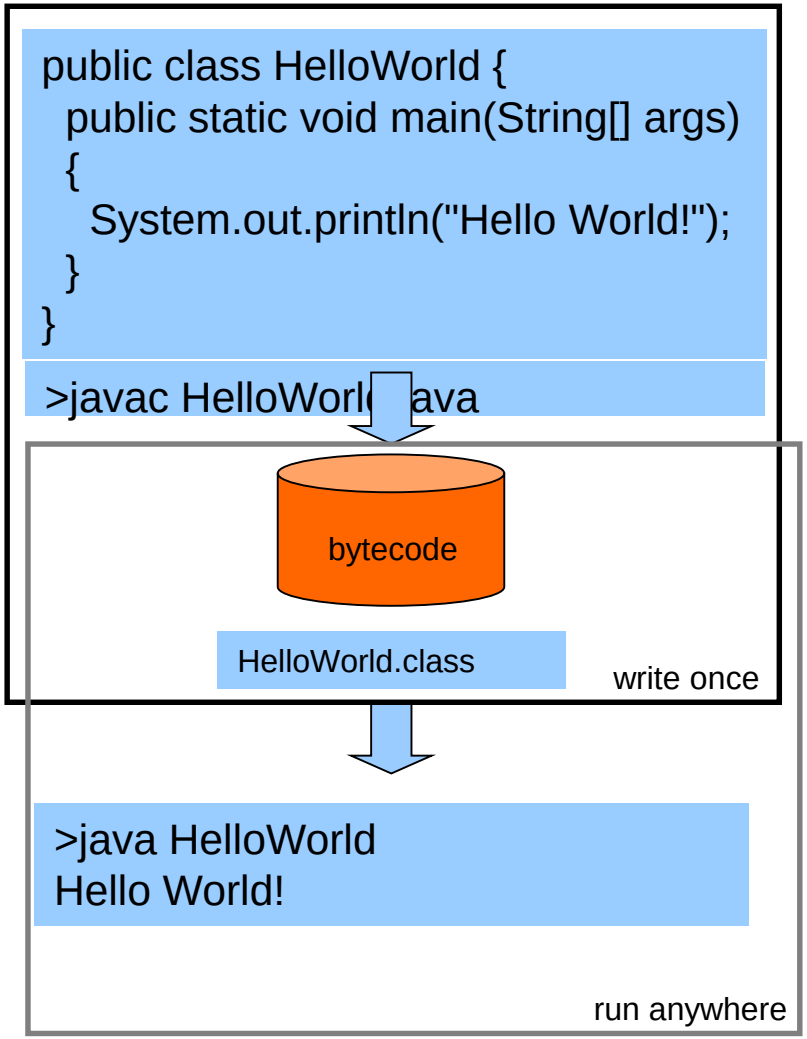
Quelle: Java 6 Documentation

Erstkontakt

- Geschichte der Plattform und Programmiersprache
  - Anfang 1996: JDK 1.0
  - Anfang 1997: JDK 1.1
  - Ende 1998: J2SE 1.2
    - Swing, Collections
  - Mitte 2000: J2SE 1.3
    - JNDI, RMI, Sound
  - Anfang 2002: J2SE 1.4
    - 64-bit Architektur, NIO, reguläre Ausdrücke, assertions, Schnittstelle für XML-Parser
  - Mitte 2004: J2SE 5.0
    - Generische Klassen, typsichere Aufzählungen, erweiterte for-Schleife, Boxing und Unboxing
  - Ende 2006: Java SE 6
  - Mitte 2011: Java SE 7
  - März 2014: Java SE 8

Die erste Java Anwendung

- Entwicklungsschritte
  - Programmieren
  - Kompilieren mit javac(.exe)
  - Ausführen mit java(.exe)
- Optional
  - Generieren einer Dokumentation mit javadoc(.exe)
  - Verpacken mit jar(.exe)
  - Verfügbar machen mit Java Web Start



Objektorientierte Programmierung

- Bildet Objekte der realen Welt ab

- Objekte
  - reagieren nur auf Botschaften, die sie verstehen
  - vereinigen Daten und Methoden
- Methoden
  - Botschaften werden durch sogenannte Methoden realisiert, d.h. durch Aufruf einer Methode sendet man eine Botschaft
  - Methoden sind Prozeduren sehr ähnlich
- Klassen bzw. Objekttypen
  - Objekte, die dieselben Botschaften verstehen und dieselben Merkmale besitzen, bilden eine Klasse von Objekten

Objektorientierte Programmierung

- Kapselung
  - Interna eines Objekts werden unsichtbar nach außen gemacht bzw. gekapselt
- Vererbung
  - Enger Zusammenhang mit Wiederverwendbarkeit und Anpassungsfähigkeit
  - Neues Objekt wird von vorhandenem Objekt abgeleitet
  - Das neue Objekt erhält dann neue charakteristische Merkmale
- Überschreiben
  - Enger Zusammenhang mit Wiederverwendbarkeit und Anpassungsfähigkeit
  - Verschiedene Unterklassen verstehen dieselbe Botschaft, obwohl die technische Umsetzung der Reaktion völlig unterschiedlich sein kann
    → Polymorphie
- Überladen
  - Klasse stellt verschiedene Varianten derselben Methode zur Verfügung, d.h. unterschiedliche Parameterstruktur

Unterschiede zwischen C++ und Java

- In Java keine Mehrfachvererbung
- In Java keine public, protected und private Vererbung

- Java hat kein Präprozessor → kein #defines oder macros
- Bei Java ist kein Linkprozess notwendig

- Java kennt keine Zeiger → dafür Referenzen zu Objekten

- In Java werden Objekte immer mit `new` erzeugt
- Speicherverwaltung in Java ist automatisch → dennoch Speicherverbrauch berücksichtigen
- Java kennt keinen Destruktor → Methode finalize()

Unterschiede zwischen C++ und Java

- Felder in Java sind durchnummerierte Listen von Referenzen
  - Ausnahme bei Aufrufen eines nicht existierenden Index
  - mehrdimensionale Felder sind Listen von Feldern
- In Java keine Definition von Operatoren möglich (Operatorenüberladug in C++)
- Copy-Konstruktor ersetzt durch Operation clone()
- In Java kein struct, union, typedef statement → man deklariert neue Klasse
- Java kennt keine header files → *import* Anweisungen und *Interface Definition*

Unterschiede zwischen C++ und Java

- Java kennt kein Schlüsselwort virtual → Schnittstellenklasse
- Java kennt keine reine virtuellen Operationen → Schlüsselwort abstract
- Java kennt keine globalen Variablen → ersetzen durch static
- In Java kein Schlüsselwort const → Schlüsselwort final

- Java bietet kein goto (Sprünge sind aber trotzdem mit Continue und Sprungmarke)

- Fallunterscheidungen: im Gegensatz zu C darf ein Ausdruck kein int zurückliefern, sondern immer nur boolean true oder false

Inhalt

- Teil 1: Überblick
  - Erstkontakt
  - Java Standard Edition
  - Objektorientierte Programmierung (OOP)
  - Unterschiede zwischen C++ und Java
- Teil 2: Java Basics
  - Klassen
  - Felder
  - Operationen
  - Konstruktoren
  - Vererbung
  - Objekte
  - Ausnahmen
  - Java API
  - Collections

Klassen

- Aufbau einer Klasse

```
package <PaketName>;

import <PaketOderEinzeln>;

<modifiers> class <KlassenName>
  extends <KlassenName>
  implements <InterfaceName>, …
{
  <Abschnitt Variablen-Definitionen>

  <Abschnitt Konstruktoren-Defintionen>

  <Abschnitt Operationen-Definitionen>
}
```

- Beispiel

```
package com.my.graphics;

import java.lang.Object;
import java.util.*;

public class Point {

public int x, y, color;

public void moveTo(int newX, int newY) {
    x = newX;
    y = newY;
}

public void moveRel(int dx, int dy) {
      x += dx;
      y += dy;
}

}
```

Schnittstellenklassen

- Schnittstellenklasse
  - Kennzeichen ist Schlüsselwort interface
  - Instanzmethoden dürfen nur public und default als Zugriffs-Modifier besitzen
  - Instanzmethoden dürfen nicht implementiert werden
  - Klassenvariablen kompilieren, können aber nicht von der implementierenden Klasse geändert werden => Konstanten
  - Klassenmethoden sind nicht erlaubt

- Beispiel

```
public interface Person {
  public String sayHello();
  Integer tellAge() throws Exception;
}

public interface Autofahrer {
  public void gibGas();
}
```

Abstrakte Klassen

- Abstrakte Klassen
  - Unterschied zu Schnittstellenklasse?

```
public abstract class Human {

  public abstract void think();
  // no implementation

  public void walk() {
    // let's assume that
    // this operation is
    // implemented
  }

}
```

Felder

- Felder von Klassen und von primitiven Datentypen
  - sind selbst Objekte, auch wenn die Elemente primitive Typen (wie int) sind
  - Drei Dinge sind notwendig um Felder zu füllen:
    - Die Deklaration einer Feld-Referenz-Variable
    - Das Konstruieren eines Feld-Objekts
    - Das Belegen der Elemente des Feld-Objekts
- Beispiel
  - siehe nächste Folie

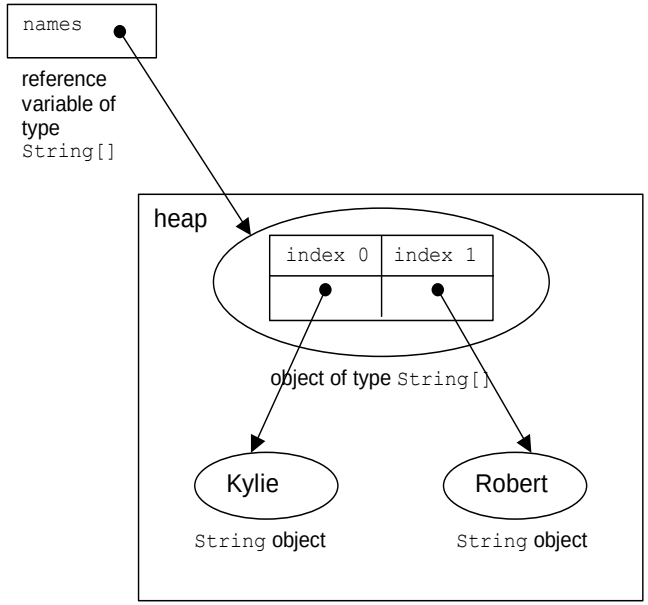
Felder

- Beispiel

```
public class ArrayTester {
  int[] key; // array of primitives
  String[] names; // array of objects
  names = new String[2];
  // or in one line
  int[] scores = new int[10];
  // or in shortcut notation
  String[] chiles = {"jalapeno",
"serrano"};
  // or in shortcut notation
  int x = 9;
  int[] yArray = {3, 6, x, 12};

  public void main(String[] args) {
    // Werte zuweisen
    names[0] = "Kylie";
    names[1] = "Robert";
    // Werte zuweisen
    for (int i=0; i<5; i++) {
      scores[i] = i;
    }
  }
}
```

- Visualisierung



Wichtige Schlüsselwörter

- package
  - Die Klasse oder die Schnittstellenklasse einer Datei gehören zu einem Paket
  - Unterläßt man die Definition, befindet sich die Klasse oder die Schnittstellenklasse im Paket „default“
  - Zusammenhang mit Verzeichnisstruktur
  - Die Definition hat Auswirkung auf die Sichtbarkeit einer Klasse für Klassen in demselben oder einem anderen Paket
  - Kategorisierung
- **import**
  - von einzelnen Klassen
  - von einzelnen Schnittstellenklassen
  - von ganzen Paketen
  - CLASSPATH
- **public**
  - einer der 3 access modifier
- **class** oder **interface**
- **extends**
  - erbt von Klasse
- **implements**
  - implementiert Schnittstellenklasse

Kommentare

- Vier Möglichkeiten
  - // Einzeiler oder am Zeilenende
  - /* ... */ Einzeiler oder am Zeilenende
  - /* ... * ... */ Mehrzeiler
  - /** ... * ... */ javadoc
- Beispiel

```
 /**
  * Prints out a welcome message
  * @version 1.0
  */
class HelloWorld {
  /*
   * Say Hello
   */
  public static void main(String[] args) {
    // output text
    System.out.println("Hello World!");
  }
}
```

- Javadoc Tags:
  - @author
    - classes and interfaces only, required
  - @version
    - classes and interfaces only, required
  - @param
    - methods and constructors only
  - @return
    - methods only
  - @exception
  - @throws
  - @see
  - @since
  - @deprecated

Variablen

- Variablen
  - zu verwenden nur bei Klassen, nicht bei Schnittstellenklassen
  - gehören entweder einer Instanz → Instanzvariable
  - oder einer Klasse → Klassenvariable bzw. statische Variable (modifier static)
  - oder befinden sich innerhalb einer Methode → lokale Variable
  - Instanzvariablen und Klassenvariablen werden dann Members oder Attribute genannt

```
public class APerson {
  // Beispiel für?
  public String lastName = "";
  private String firstName;

  // Beispiel für?
  public static final boolean
     eachPersonHasAName = true;
  private static int age;

  public void doSomething() {
    int i=5;
  }
}
```

- Ein Wort zu Attributen
  - Immer mit getX- und setX-Operationen kapseln
  - Kapselung
  - Einschränkung des Zugriffs
  - „Getters and Setters“ oder JavaBeans oder „Accessors and Mutators“

Operationen

- Operationen
  - zu verwenden bei Klassen und Schnittstellenklassen

```
<modifiers> <return type>
<MethodenNamen> ([Parameter1,
  Parameter2, …])
[throws Exception1, …] {
  // Methodenkörper
}
```

- Beispiel

```
public void doSomething() {
  // implement here
}

public Integer calculate(Integer
  i1, String s1) throws Exception {
  // implement here
}
```

Operationen

- Was tut man hier?

```
public class DoClass {

public void doSomething(Integer i) {
}

// OK?
public void doSomething(String s) {
}

// OK?
public String doSomething(Integer i)
{
}

// OK?
public String doSomething(int i) {
}

} // end of class
```

- Ist das erlaubt?

```
public class ClassWhatsThis {

// Methode?
public ClassWhatsThis() { }

} // end of class
```

- Wann ist eine Methode „eindeutig“?

Konstruktoren

- Aufgerufen beim Erzeugen eines Objekts => nur bei Klassen
- Überladen möglich
- Im Unterschied zu Methoden haben Konstruktoren keinen Ergebnistyp
- Jede Klasse, die keinen Konstruktor implementiert erhält einen parameterlosen default Konstruktor

```
// eigener default constructor
public <KlassenName> { }
```

- Es ist möglich den default Konstruktor zu überschreiben
- Denselben Namen wie die Klasse
- Mehrere Konstruktoren mit unterschiedlichen Parametern sind möglich
  - Wozu?
- Aufruf von anderen Konstruktoren in einem Konstruktor?

Konstruktoren

- Welchen Konstruktor hat die folgende Klasse?

```
public class ClassWithoutConstructor {
}
```

- Warum ist es nicht möglich zu kompilieren?

```
public class DoesNotCompile {
  public String DoesNotCompile(int i) {
  }
}
```

- Frage: Wieviele Konstruktoren hat diese Klasse?

```
public class ClassHowMany {
  public ClassHowMany(String s) {
  }
}
```

- Wie nennt man dies?

```
public class ClassTwoCons {
  public ClassTwoCons(Integer i) {
  }

  public ClassTwoCons(String s) {
    this(new Integer(5));
  }
}
```

Fragen Sie jetzt!

Oder haben Sie eigene Beiträge?

Erben

- Erben von einer Schnittstellenklasse
  - Eine Schnittstellenklasse kann von einer Schnittstellenklasse erben
- Wieviele Methoden muss eine Klasse implementieren, die die Schnittstellenklasse Yourself implementiert?

```
public interface Person {

  public String sayHello();

  Integer tellAge() throws Exception;
}


public interface Yourself extends Person {
  public boolean isAwake();
}
```

Erben

- Erben von einer Klasse
  - Klassen können vererben
    - Attribute
    - Konstruktoren
    - Operationen

```
public class ParentClass {

  public String text = "ParentClass";

  public ParentClass(String s) {
    System.out.println("ParentClass");
  }

  public String saySomething() {
    System.out.println("Hallo");
    return ""; // macht keinen Sinn
  }

}
```

- Fortsetzung:

```
public class ChildClass
extends ParentClass {

 public ChildClass() {
    System.out.println("ChildClass");

    // Warum wird dies
    // NICHT kompilieren?

    // Was muss ergänzt werden?
  }

  // Welche Operation gibt es gratis?

  // erlaubt? was ist denn das hier?
  public String saySomething() {
    System.out.println("Hallo erstmal");
  }

}
```

this und super

- Schlüsselwörter this und super
  - Beides referenziert Objekte
  - this(…) und super(…) zum Aufrufen von Konstruktoren
  - this.<name> und super.<name> zum Aufrufen von Methoden und zum Referenzieren von Attributen

```
public class ChildClass
extends ParentClass {

public ChildClass(String s) {
  new ParentClass();
  // erlaubt? siehe Folie zuvor?

  super(""); // erlaubt?

  System.out.println("ChildClass");

  super.saySomething();
}

public static void main(String[] args)
{
  //new ChildClass();
  // warum kompiliert dies nicht?

  new ChildClass("");
  // Ausgabe?
}

} // end of class
```

Objekte erzeugen

- Objekte erzeugen
  - geschieht immer unter Verwendung des Schlüsselwortes new und eines der Konstruktoren
  - Java allokiert Speicherplatz für das Objekt
- Konstruktoren
  - überschreiben und überladen möglich

- Beispiel
  - Was wurde vorausgesetzt, damit die zweite Zeile kompiliert?
  - Welches sind die Objekte?

```
new Person();

Person p1 = new Person();

Person p2 = new Person("Hans", "Meier");

p1 = p2;
```

  - Referenz = Zeiger?

Objekte verwenden

- Aufruf von Attributen und Operationen an Objekten
  - dot-Operator .

```
String s1 = p.mNachname;

p.mNachname = "Meier";

p.sayHello();

String vornameUndNachname =
p.toString();


String s3 = p.getClass().getName();
// p.getClass() liefert ein Objekt
vom Typ Class, Stichwort Reflection
```

- Aufruf von Attributen und Operationen an Klassen
  - Wie müssen diese nochmal deklariert sein?

```
int i = Person.AnzahlArme; // 2

Person.sayHello();
// alle Personen können das
```

Übergeben von Objekten

- Objekte als Parameter von Methoden
  - Gibt man den Wert eines primitiven Datentyps an eine Methode, dann spricht man von „pass by value“ → Kopie des Wertes
  - Gibt man ein Objekt an eine Methode, dann erhält diese keine Kopie des Objekts, sondern IMMER eine KOPIE der Referenz → eine Art „pass by reference“

```
// Ausgabe "hello" oder "bleibt so"?
public class PassByReference {

  public static void main(String[] args) {
    String str = "bleibt so";
    System.out.println("str="+str);

    Person p = new Person(„Vanessa");

    PassByReference pbr =
      new PassByReference();
    pbr.sayHello(str, p);
    System.out.println("str="+str);
    System.out.println("p="+p.getName());
  }


 public void sayHello(String s, Person p) {
   s = "hello";
   p.setName(„Denise");
 }
} // end of class
```

Objekte löschen

- Objekte löschen
  - Das Löschen übernimmt Java's Garbage Collector, wenn keine Referenzen mehr auf das Objekt mehr vorhanden sind
  - Man löscht eine Referenz explizit mit Schlüsselwort null
  - Operation finalize() in Object zum Aufräumen

- Beispiel
  - Welches Objekt wäre ein Kandidat zur Garbage Collection?

```
Person p = new Person();
Person p2 = p;
p = null;
```

Sichtbarkeit

- Sichtbarkeit von Attributen und Operationen
  - public Operation und Attribute:
    - Operationen und Attribute sind von allen anderen Klassen verwendbar
  - default Operation und Attribute:
    - Operationen und Attribute sind nur von anderen Klassen in demselben package verwendbar, sonst compile error
  - protected Operation und Attribute:
    - Operationen und Attribute sind nur von VERERBTEN Klassen in demselben package verwendbar, sonst compile error
  - private Operation und Attribute:
    - Operationen und Attribute sind nur von der eigenen Klasse verwendbar

Ausnahmen

- Ausnahmen
  - Ausnahmen sind Subklassen der Klasse java.lang.Exception
  - Von vielen Operationen der Java API werden Ausnahmen geworfen
  - Zusätzlich gibt es Ausnahmen zur Laufzeit. (z.B. bei der Division durch 0)
  - Diese Ausnahmen muss man entweder auffangen (try/catch) oder weiter werfen (throws)
  - Im Rahmen der eigenen Anwendung definiert und verwendet man eigene Ausnahmen zwecks Robustheit von Operationen
  - Häufig verwendetes Konzept

- Ausnahmen in einer eigenen Operation werfen => Design-Frage

```
public class ClassWithException {
  public void anExceptionalMethod()
  throws Exception {
    if (someThingUnusualHasHappened())
    {
      throw new Exception(„something wrong“);
      // execution never reaches here
    }
  }
} // end of class
```

- Ausnahmen auffangen und werfen

```
public void responsibleMethod()
throws Exception {
  ClassWithExceptiion cwe =
    new ClassWithException();
  try {
    cwe.anExceptionalMethod();
  } catch (Exception e) {
    // do something responsible
    // re-throw the exception
    throw e;
    // or another exception!
}
}
```

Collections

- Arten
  - sorted
  - unsorted
- Typsicherheit geht verloren
  - Beim Einstellen impliziter up-cast von beliebigem Typ zu Typ Object
  - Bei der Entnahme expliziter down-cast von Object auf Ursprungstyp
  - Beim Konvertierungsversuch ClassCastException
  - instanceof

- Beispiel ArrayList

```
ArrayList mixed = new ArrayList();

mixed.add("Foo");
mixed.add(new Complex(2, 3));
mixed.add(new int[] {1, 2, 4, 19});
mixed.add(new ArrayList());

for (int i=0; i<mixed.size()) {
  Object o = mixed.get(i);

  System.out.println(o.hashCode());
}
```

- Beispiel HashMap

```
Map phonebook = new HashMap();

phonebook.put("Lara", new Long(1225100));
phonebook.put("Schiedermeier", new Long(22222));
phonebook.put("Sekretariat", new Long(111111));

Long l = (Long) phonebook.get("Lara");
```

Literatur

- http://java.oracle.com
- http://download.oracle.com/javase/7/docs/

- Stefan Middendorf, Reiner Singer, Jörn Heid, *Java*, Dpunkt Verlag, Oktober 2002
- David Flanagan, *Java in a Nutshell*, O'Reilly, November 2005
- Christian Ullenboom, *Java ist auch eine Insel*, Galileo Computing, 8. Auflage, 2009, http://www.tutego.de/javabuch/
- Guido Krüger, *Handbuch der Java-Programmierung*, Addison-Wesley, 2009, http://www.javabuch.de
- Joshua Bloch, *Effective Java, 2nd Edition ,* O'Reilly, 2008
- Podcast: Chaosradio Express, *CRE090: Java – Ein Überblick über die Java-Plattform*, http://chaosradio.ccc.de/cre090.html

Marcus Seiler
Institute of Computer Science
Chair of Software Engineering
Im Neuenheimer Feld 326
69120 Heidelberg, Germany
http://se.ifi.uni-heidelberg.de
marcus.seiler@informatik.uni-heidelberg.de

RUPRECHT-KARLS-UNIVERSITÄT HEIDELBERG