JAVA

Datentypen

byte
short
int
long
float
double
char
boolean

zugehörige Klassen:
Byte
Short
Integer
Long
Float
Double
Character
Boolean

weitere “Datentypen”:

String
StringBuffer

Array

Referenz

Operatoren

== vs. equals

Ablaufstrukturen

Sequenz:

Block {}
Definition lokaler Variablen für Block!

Selektion:

if ... else ...
switch

Iteration:

while
for
do ... while

break
continue
return

Objektorientierung

- ⇨ **zentrale Konzepte sind**
  - **Klasse und Instanz (Objekt)**
  - **Kapselung**
  - **Vererbung**
  - **Polymorphie**

Klasse

- ⇨ **definiert durch**

  - **Variablen (Attribute)**
    - **Instanzvariablen**
    - **Klassenvariablen**

  - **Konstruktor(en)**
    - **finalizer**
    - **Garbage Collection**

  - **Methoden (Operationen)**
    - **Instanzmethoden**
      - **this**
    - **Klassenmethoden**

Überladen von Methoden möglich

Vererbung

Vererbung in Java ist Einfachvererbung:

- **Subclass extends Superclass**

Polymorphie (Überschreiben von Methoden)

- **late binding!**

super => Aufruf einer Methode der Superclass

Konstruktor(en) können eigentlich nicht überschrieben werden, aber der Konstruktor der Superclass kann im Konstruktor der Subclass über den Namen super aufgerufen werden!

Alle Klassen erben von java.lang.Object!

Pakete

Ziel:

- **Organisation von Klassen**
- **Identifikation von Klassen**
- **Reduktion von Namenskonflikten**
- **Zugriffsschutz (private Klassen ...)**

Pakete anlegen

- **package Anweisung**
- **Verzeichnisstruktur beachten!**

Pakete verwenden

- **Explizite Verwendung vs. import Anweisung**
- **Namenskonflikte bedingen explizite Verwendung!**

CLASSPATH beachten!

Modifier

Zugriffsschutz

Standard
public
protected
private

(Kein stärkerer Zugriffsschutz in der Subclass möglich – Ausnahme Standard!)

Betrifft Variablen, Konstruktoren, Methoden, Klassen, Interfaces

static

Klassenvariablen und –methoden

final

Klasse: nicht ableitbar
Methode: nicht überschreibbar
Variable: nicht veränderbar => Konstante

abstract

Klasse: keine Instantiierung möglich
Methode: keine Implementierung
Abstrakte Klassen können auch nicht abstrakte Methoden haben!

Schnittstellen (Interfaces):

Zweite Vererbungshierarchie parallel zur Klassenhierarchie

- **kein Gegenstück zur “Object“ Klasse**
- **Mehrfachvererbung möglich**

Klassen implementieren (ggf. mehrere) Schnittstellen!

Mehrfachvererbung?

Schnittstellen können wie Klassen als “Datentypen“ verwendet werden!

Variablen eines public Interface: public, static, final
Methoden eines public Interface: public, abstract, nicht static!
Nichtöffentliche Schnittstellen => Variablen und Methoden nichtöffentlich

Programmierung im Hinblick auf Schnittstellen, nicht Klassen!

Beispiel:

```
package org.tit02agr.artikel;

import java.util.*;

public class Artikel {

	// Instanzvariablen
	protected String artikelnummer;
	protected String artikelbezeichnung;

	// Klassenvariablen
	private static Map artikelListe = new HashMap();

	// Konstruktor(en)
	Artikel () {
	}
	Artikel (String artikelnummer, String artikelbezeichnung) {
		this.artikelnummer = artikelnummer;
		this.artikelbezeichnung = artikelbezeichnung;
	}

	// Instanzmethoden
	public String getArtikelnummer () {
		return artikelnummer;
	}
	public String getArtikelbezeichnung () {
		return artikelbezeichnung;
	}
	public void setArtikelnummer (String artikelnummer) {
		this.artikelnummer = artikelnummer;
	}
	public void setArtikelbezeichnung (String artikelbezeichnung) {
		this.artikelbezeichnung = artikelbezeichnung;
	}
	public String toString() {
		return "Artikelnummer: "+ artikelnummer +
		"\nArtikelbezeichnung: " + artikelbezeichnung;
	}


	// Klassenmethoden
	public static Artikel createArtikel
		(String artikelnummer, String artikelbezeichnung) {
		Artikel artikel = new Artikel ( artikelnummer, artikelbezeichnung);
		artikelListe.put(artikelnummer, artikel);
		return artikel;
	}
```

```
        public Artikel retrieveArtikel (String artikelnummer) {
                return (Artikel) artikelListe.get(artikelnummer);
         }

        public static Map getArtikelListe() {
                return artikelListe;
        }

        public static void setArtikelListe(Map map) {
                artikelListe = map;
        }

}
```

```
package org.tit02agr.artikel;

import java.util.*;

public class KategorisierterArtikel extends Artikel {

        // (zusätzliche) Instanzvariablen
        Map artikelKategorien = new HashMap();

         // Konstruktor(en)
        KategorisierterArtikel () {
        }
        KategorisierterArtikel
        (String artikelnummer, String artikelbezeichnung, Map kategorien) {
                super(artikelnummer, artikelbezeichnung);
                if (kategorien != null)
                        this.artikelKategorien = kategorien;
         }

         // (zusätzliche) Instanzmethoden
         public String toString() {
                String str = super.toString();
                if ((artikelKategorien == null) || artikelKategorien.isEmpty())
                        return str;
                str = str + "\nKategorien: ";
                Iterator iter = artikelKategorien.values().iterator();
                while (iter.hasNext()) {
                        String kategorie = (String) iter.next();
                        str = str + "" + kategorie;
                }
                return str;
         }

        public static Artikel createArtikel
                (String artikelnummer, String artikelbezeichnung) {
                KategosisierterArtikel artikel =
                        new KategorisierterArtikel ( artikelnummer,
                artikelbezeichnung, null);
                Artikel.getArtikelListe().put(artikelnummer, artikel);
                return artikel;
        }

        public Map getArtikelKategorien() {
                return artikelKategorien;
        }

        public void setArtikelKategorien(Map map) {
                artikelKategorien = map;
        }

}
```

Exceptions:

Ausnahmebehandlung in JAVA:

Trennung Kontrollfluß – Ausnahme

Exception erbt von Throwable

- ⇨ **Error (keine Steuerung via Applikation)**
- ⇨ **Exception**
  - o **RuntimeException (sollten nicht auftreten!)**
  - o **Sonstige Exceptions (in Applikation zu behandeln!)**

Methoden können so definiert werden, dass sie eine Exception werfen
... throws *Exception*

(Überschriebene Methoden können so deklariert werden, dass sie weniger Exceptions werfen!)

Aufrufende Methoden müssen diese Exception behandeln oder so definiert werden, dass sie diese Exception werfen.

Erfolgt keine Behandlung => Abbruch der Applikation!

Exception Handling

Übliche Behandlung:

```
try {
   ...
} catch (Exception e) {
   …
} …
   finally [
   …
}
```

Pro try Block sind mehrere catch Blöcke zulässig, von denen der erste zutreffende ausgeführt wird!

finally Block wird auf jeden Fall nach Ausführung des Exception Handling ausgeführt und erlaubt z.B. die Freigabe von Ressourcen!

Methoden können Exceptions werfen:

throw *Exception*();

Eigene Exception Classes (erben von Exception) können definiert werden!

Exception Classes sollten zwei Konstruktoren haben:

- ⇨ **ohne Parameter**
- ⇨ **mit dem String Parameter *Exception Message***