UV Objektorientierte Programmierung

SS 2018

H.Hagenauer
FB Computerwissenschaften

UNIVERSITÄT SALZBURG

Übersicht zur LV

Ziel: Basiskonzepte der objektorientierten Programmierung besser verstehen und damit vertiefende Techniken kennenlernen und gezielt anwenden können.

Inhaltliche Voraussetzung: *Einführung in die Programmierung*

Programmiersprache: Java

V-Teil: kurze Einführung zu verschiedenen Konzepten

U-Teil: Programmieraufgaben, Ausarbeitung ergänzender Themen zum V-Teil

Prüfungsmodus:

- 2 Tests (gleichwertig)
  – positiv wenn 50% der Gesamtpunkte erreicht werden
- Aufgaben bearbeiten und abgeben (maximal 3 Punkte pro Aufgabe)
  – positiv wenn 50% der Gesamtpunkte erreicht werden

→ *jeder der beiden Teile muss positiv sein - Gesamtnote 50:50!*

UNIVERSITÄT SALZBURG

Terminvorschau

Termin	V	U
01.03.2018	2	
08.03.2018		3
15.03.2018		
22.03.2018	2	
29.03.2018	LV-frei	
05.04.2018	LV-frei	
12.04.2018		3
19.04.2018		3
26.04.2018	2	
03.05.2018		3
10.05.2018	Feiertag	
17.05.2018	1	
24.05.2018		3
31.05.2018	Feiertag	
07.06.2018	1	
14.06.2018		3
21.06.2018		3
28.06.2018		

V-Teil: jeweils 11:30 Uhr, 1 oder 2 SSt

U-Teil: jeweils 1 SSt pro Gruppe

- Gruppe 1: 11:30 Uhr
- Gruppe 2: 12:20 Uhr
- Gruppe 3: 13:10 Uhr

Testtermine

- 26.April 2018
- 7.Juni 2018

UNIVERSITÄT SALZBURG

Abgabe von Programmen (und Dateien)

- Abgabe mittels Abgabesystem: https://abgaben.cosy.sbg.ac.at/
- Abgabedatum und -zeitpunkt wird bei der Aufgabenstellung angegeben.
- Alle abzugebenden Dateien sind in einer zip-Dateil zusammenzufassen und abzugeben.
- Alle zur Kompilierung notwendigen Dateien sind abzugeben.
- Bearbeitung der Aufgabe im Team möglich:
  - ein Team darf maximal aus 3 Studierenden bestehen
  - jedes Teammitglied hat selbst abzugben
  - jede Datei muss die Namen aller Teammitglieder enhalten (z.B. Java-Dateien als Kommentar in der ersten Zeile)
  - jedes Teammitglied muss die Abgabe erklären können.

UNIVERSITÄT SALZBURG

Abgabe von Programmen (und Dateien) (2)

- Programmcode muss
  - korrekt eingerückt sein,
  - strukturiert und lesbar sein,
  - sinnvoll kommentiert werden.
- Nur **kompilierbare** Programme werden berücksichtigt!
- Abgabesystem
  - bei der ersten Anmeldung muss das Passwort neu gesetzt werden. Es wird eine E-Mail mit einem Link zur Passwortänderung versendet.
  - Sollten Sie in diesem Semester an weiteren LV teilnehmen, die auch dieses Abgabesystem nutzen, dann sind die selben Zugangsdaten zu verwenden.

UNIVERSITÄT SALZBURG

Literaturhinweise und Webseiten

Bücher

- W.Savitch: *Java – An Introduction to Problem Solving & Programming*, 7th edition, Pearson, 2014
- ... weitere umfangreiche Auswahl

Webseiten

- `http://www.oracle.com/technetwork/java/index.html`
  vieles zu Java vom Softwarehersteller Oracle (u.a. JDK)
- `http://docs.oracle.com/javase/8/docs/api/index.html`
  API-Spezifikation (Application Programming Interface)
- `http://docs.oracle.com/javase/tutorial/`
  Tutorien zu verschiedenen Themen rund um Java
- ... große Auswahl mit verschiedenen Schwerpunkten (Tutorien, Tipps, Regelwerk, Fragen, ...)

UNIVERSITÄT SALZBURG

Kapitel 1

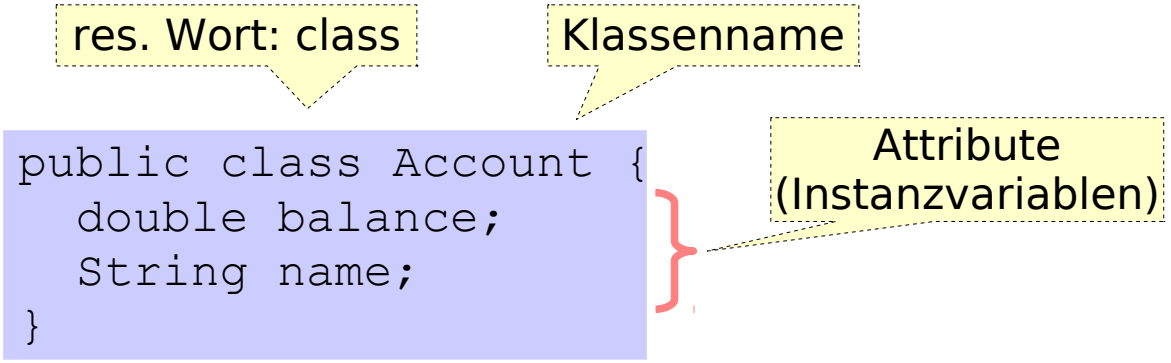
Wiederholdung
Klassen, Objekte, Methoden

UNIVERSITÄT SALZBURG

Klasse, Objekte

Eine ***Klasse (class)*** fasst mehrere Variablen verschiedener Typen zu einem neuen Typ zusammen.

- Variablen der Klasse heißen ***Attribute*** oder ***Instanzvariablen*** *(instance variables, fields)*



- Objekte werden als ***Instanzen*** einer Klasse bezeichnet (müssen explizit mittels `new` erzeugt werden)
- Variablen von Klassentypen sind Zeiger (Referenzen, Pointer) auf Objekte
- Zugriff auf Instanzvariable mittels „Dot-Notation“

UNIVERSITÄT SALZBURG

Methoden

Daten haben meist auch Operationen, die auf sie zugreifen und Änderungen vornehmen

⇨ Daten und „ihre“ Operationen gehören zusammen - sie werden in *einem* Sprachkonstrukt definiert - ***Klasse (class)***

Operationen werden in OO-Prog. ***Methoden (methods)*** genannt.

⇨ eine ***Klasse*** definiert ***Daten*** und die dazugehörenden ***Methoden.***

- Methodenaufruf (method invocation, method call) i.A. mittels Objekt/Objektvariable und „Dot-Notation“
- Methoden ohne/mit Rückgabewert
- formale/aktuelle Parameter, call-by-value
- globale/lokale Variablen

UNIVERSITÄT SALZBURG

Datenkapselung

Zugriffsrechte (auf Instanzvariable, Methoden) mittels *Modifikatoren (modifiers)* in Java:

`private` : Zugriff nur von innerhalb der Klassendefinition

`protected` : Zugriff von allen Unterklassen sowie allen Klassen des deklarierenden Pakets (i.A. des selben Verzeichnisses)

keine Angabe: Zugriff von allen Klassen des deklarierenden Pakets (des selben Verzeichnisses)

`public` : freier Zugriff von beliebigen Klassen aus

Damit übliche Vorgehensweise für **Datenkapselung (data encapsulation)**:

- Instanzvariablen als `private` deklarieren
- Zugriffsmöglichkeiten über `public` Methoden anbieten
- public-Teile werden als **Schnittstelle (interface)** bezeichnet

UNIVERSITÄT SALZBURG

Konstruktoren, static, ...

- **Konstruktoren (constructors)** sind spezielle Methoden, die bei der Erzeugung von Objekten einer Klasse (mit `new`) aufgerufen werden (default-Konstruktor, mehrere Konstruktoren)
- **static**-Methoden und Variablen gehören zu einer Klasse und sind nicht auf ein konkretes Objekt bezogen
- Mehrere Methoden in einer Klasse mit dem selben Namen sind möglich – **überladen von Methoden (overloading)** - Unterscheidung mittels Parameterliste

UNIVERSITÄT SALZBURG

Oberklasse, Unterklasse - Vererbung

Spezialisierung/Generalisierung von Klassen
→ zu bestehenden Klassen werden Unterklassen gebildet.

Eine **Unterklasse (subclass, child class)**

- *erbt (inherits)* Eigenschaften der **Oberklasse (base class, superclass)**,
- kann weitere Eigenschaften (das sind Instanzvariable, Methoden) hinzufügen (d.h. es wird erweitert)
- und vorhandene Eigenschaften (i.A. Methoden) *überschreiben (overriding)* – d.h. semantisch anpassen.

Dieses Konzept wird **Vererbung (inheritance)** genannt.

Damit können Klassenhierarchien erstellt werden
- Spezialisierung/Generalisierung.

UNIVERSITÄT SALZBURG

Polymorphismus

Polymorphismus (polymorphism) bedeutet „Vielgestaltigkeit“

⇨ Programmierung: mehrere Typen sind im selben Kontext erlaubt, jedoch mit möglicherweise unterschiedlicher Wirkung!

In *Java* beinhaltet *Polymorphismus* 2 Bedeutungen:

1. Typfamilien

Sei A eine Klasse und B eine Unterklasse von A – dann gilt:
wo ein A-Objekt erwartet wird (Zuweisung, Parameter), darf ein B-Objekt eingesetzt werden.

Sinnvoll, da das B-Objekt alle Eigenschaften des A-Objekts besitzt.

2. Methodenaufruf und dynamische Bindung

Sei wieder B eine Unterklasse von A, und die Methode m aus A wird in B überschrieben – dann gilt (wenn r Variable vom Typ A ist):
beim Aufruf der Methode mittels r.m(...) entscheidet der Typ des von r referenzierten Objekts welche Version der Methode ausgeführt wird (und nicht der Typ von r).

entsprechende Ausprägung der Methode erst zur Laufzeit ermittelbar

→ **dynamische Bindung (dynamic binding, late binding)**

UNIVERSITÄT SALZBURG

Abstrakte Klassen

Methoden ohne Implementierung (d.h. ohne Rumpf) werden **abstrakte Methoden** genannt. Enthält eine Klasse mindestens eine abstrakte Methode, wird sie zu einer **abstrakte Klasse**.

- von abstrakten Klassen können keine Objekte erzeugt werden!
- Unterklassen von abstrakten Klassen müssen zu allen abstrakten Methoden die Rümpfe implementieren oder sind wieder abstrakt.
- eine abstrakte Klasse definiert eine *Schnittstelle* für spätere Unterklassen – legt fest, welche Methoden mindestens implementiert werden müssen (sonst Compilerfehler!).
- abstrakte Klasse definiert einen Typ – d.h. Variable, Parameter, ... deklarierbar.
- mittels Variablen/Parameter einer abstrakten Klasse sind nur die Methoden dieser Klasse aufrufbar (nicht weitere einer Unterklasse)!

UNIVERSITÄT SALZBURG

Kapitel 2

Interfaces

UNIVERSITÄT SALZBURG

Motivation - Beispiel

- Menge von Bankkonten
  – durchschnittlicher Kontostand
- Menge von (einfachen) Graphikobjekten
  – durchschnittliche Fläche
- .....

1. Σ bilden
2. durch Anzahl der Elemente dividieren

D.h. der Basisalgorithmus ist immer der selbe – nur eine Methode nötig!

Was wird dazu benötigt – was sollen so verschiedene *Dinge* gemeinsam haben?

⇨ etwas messbares – ein Maß

z.B. Bankkonto – Kontostand, Graphikobjekt – Fläche, ...

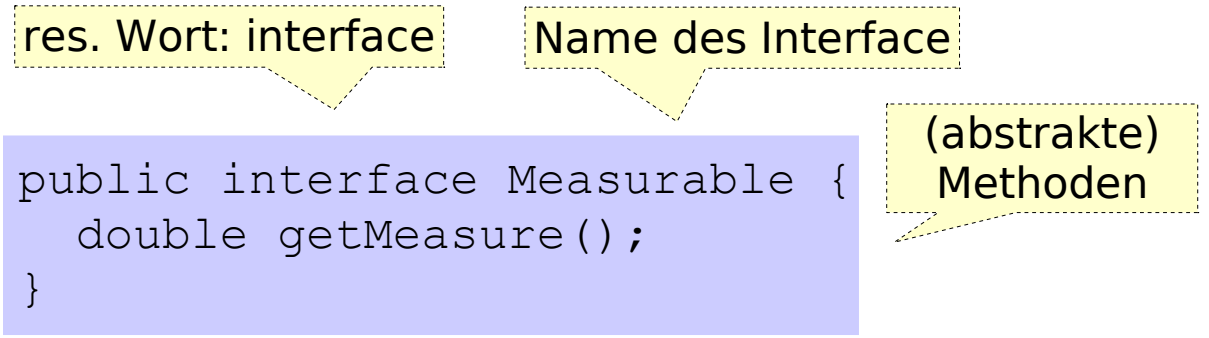
Umsetzung mittels einer Methode, die von *allen* betroffenen Klassen zur Verfügung gestellt wird (z.B. `getMeasure()`).

UNIVERSITÄT SALZBURG

Interface - Deklaration

Wenn mehrere Klassen, auch aus verschiedenen Klassenhierarchien, die selben Eigenschaften aufweisen sollen

⇨ Interface definieren



- Interfaces enthalten
  - abstrakte Methoden (sind automatisch auch public!)
  - static Konstante
  - static und default Methoden (ab Java 8)
- Interfaces in eigener Datei implementieren

UNIVERSITÄT SALZBURG

Klassen implementieren Interfaces

Interfaces erzeugen keine Klassenhierarchie

– Klassen **implementieren** Interfaces

implements-Klausel (res. Wort)

Interface

```
public class Account implements Measurable {
   ...
   public double getMeasure(){
     return balance;
   }
   ...
}
```

Methode aus Interface – jetzt mit Rumpf

beliebig viele Klassen können ein Interfaces implementieren

```
public class Circle implements Measurable {
   ...
   public double getMeasure(){
     return getArea();
   }
   ...
}
```

UNIVERSITÄT SALZBURG

Interfacetyp

Interfaces definieren einen Typ – **Interfacetyp**

- d.h. ein Interface definiert, welche Eigenschaften alle Objekte dieses Typs aufweisen
- es können keine Objekte eines Interfacetyps erzeugt werden – jedoch Referenzen (etwa Variable oder Parameter) auf Objekte einer Klasse, die das Interface implementiert
- über Referenzen eines Interfacetyps kann nur auf die im Interface definierten Eigenschaften zugegriffen werden

(siehe `Measurable, Account, MeasurableDemo, ...`)