Westfälische
Wilhelms-Universität
Münster

Programmieren in Java

objektorientierte Programmierung 2

wissen.leben
WWU Münster

Patrick Förster, Thorsten Hovenga

Zentrum für
Informations
Verarbeitung

Zusammenhang Klasse-Datei

- In jeder *.java – Datei kann es genau eine public-Klasse geben wobei Klassen- und Dateiname übereinstimmen.
- Es können weitere Klassen in einer Java-Datei enthalten sein wobei die Konvention gilt, dass möglichst nur eine Klasse pro Datei definiert wird damit die Dateistruktur weitgehend deckungsgleich mit der Klassenstruktur ist
- Ausnahme sind innere Klassen. Diese sind sowieso nur innerhalb der Klasse in der Sie definiert werden sichtbar.

Beispiel.java:

```
public class Beispiel{
…
}
```

Paketstruktur

- Jede Java-Klasse hat ein sog. Package in dem Klassen die zu einem Projekt oder einer Programmieraufgabe gehören abgelegt werden
- Wird kein package definiert gehört die Klasse zum default-package
- Die Definition erfolgt am Anfang einer Java-Datei mit dem Schlüsselwort `package`
- Beispiel: `package de.java.vorlesung;`
- Die Punktnotation gibt hier eine Hierarchie wieder.
- Konvention ist, dass Klassen in deutscher Sprache unterhalb des de-Package abgelegt werden

Beispiel Paketstruktur

- Die Klassen packagetest.java und importtest.java

```
package de.java.vorlesung;
public class packagetest {
    public void hallo(){
        System.out.println("Hallo");
    }
}
```

```
import de.java.vorlesung.packagetest;
public class importtest {
    public static void main(String[] args) {
        packagetest test = new packagetest();
        test.hallo();
    }
}
```

Imports

- Importiert werden können Klassen und Pakete
- Schlüsselwort ist hier `import`
- Soll ein ganzes Paket importiert werden setzt man hinter den Paketnamen ein `.*`
- Zugriff auf statische Methoden anderer Klassen in anderen Paketen erfolgen mit der Punktnotation des packages.Klassenname.Methodenname
- Beispiel: `de.java.vorlesung.packagetest.hallo();`
- Wenn die Klasse bereits vorher importiert wurde reicht der Klassenname

This

- `this` ist ein weiteres Schlüsselwort und liefert das aktuelle Objekt einer Klasse.
- Wird immer benötigt wenn in Methoden auf die Attribute eines Objekts zugegriffen werden soll und mehrere Instanzen einer Klasse involviert sind.
- Wird häufig bei Überdeckungsproblemen eingesetzt
- Beispiel:

```
class test {
 int wert;
test(int wert)
{
wert = wert; //Das kann nicht funktionieren
this.wert = wert;
}
}
```

Designregeln get- und set-Methoden

- Prinzip einer Klasse ist es, dass die Methoden der Klasse benutzt werden können ohne besondere Rücksicht auf die Implementierung zu nehmen
- Wenn in einer Methode die Attribute ihres Objekts manipuliert werden ist häufig eine Beschränkung des Zugriffs auf die Attribute erforderlich

```
public class Bruch {
int zaehler;
int nenner;

double ausrechnen(){
return (double)zaehler /
nenner;
}
}
```

```
 public class Bruchsafe {
int zaehler;
private int nenner;
int getNenner(){
return nenner;
}
void setNenner(int arg){
if(arg != 0)
nenner = arg;
else
System.out.println("Fehler! Nenner ist 0");
}
double ausrechnen(){
return (double)zaehler / nenner;
}
}
```

Vererbung bei Java-Klassen

- Eine Vererbung der Attribute und Methoden einer Klasse an eine abgeleitete Klasse erfolgt mittels `extends`

```
class Basis{
int wert;
void ausgeben(){
System.out.println("Wert = "+
wert);
}
}
```

```
class Abgeleitet extends Basis{

void erhoehen(){
          wert+=20;
          }
}
```

```
public class Vererbung {
public static void main(String[] args) {
Abgeleitet obj = new Abgeleitet();
obj.wert = 5;
obj.erhoehen();
obj.ausgeben();
}
}
```

Ausgabe(Vererbung):
25

Fakten zur Vererbung

- Die Basisklasse vererbt alle in Ihr enthaltenden Elemente mit Ausnahme der Konstruktoren und der private-Elemente
- Die Basisklasse wird durch Vererbung nicht geändert
- Eine Java-Klasse kann nicht kann nicht gleichzeitig von zwei oder mehr Klassen abgeleitet werden
- Klassen, die als Final deklariert sind können nicht als Basisklasse verwendet werden
- In den Mehtoden der abgeleiteten Klasse können die ererbten Elemente direkt verwendet werden
- Der Zugriff auf geerbte Attribute und Methoden erfolgt wie üblich über ein Objekt(bei statischen Elementen über den Klassennamen)
- Ererbte Elemente behalten ihren Zugriffsschutz
- Ererbte Attribute werden mit dem Standardkonstruktor der Basisklasse initialisiert

super()

- Will man steuern ob der Standardkonstruktor der Basisklasse aufgerufen wird oder ein anderer Konstruktor der Basisklasse wird die Methode super() benutzt mit der man die Konstruktoren der Basisklasse aufrufen kann.

```
class BasisK2{
int b_wert;
BasisK2(int arg){
b_wert = arg;
}
}
```

```
class AbgeleitetK2 extends
BasisK2{
int abc_wert;
AbgeleitetK2(int arg){
super(12);
abc_wert = 9;
// b_wert = 5;
}
}
```

```
public static void main(String[] args) {
AbgeleitetK2 obj = new AbgeleitetK2();
System.out.println("b-wert = "+obj.b_wert);
System.out.println("abc-wert = "+obj.abc_wert);
}
```

Interfaces

- Interfaces oder Schnittstellen sind vorstellbar als abstrakte Klassen, die abstrakte public-Methoden als Elemente enthalten

```
public interface Umkehrbar {
void umkehren();
}

public class Textpassage implements Umkehrbar {
public void umkehren(){}
}
```

- Interfaces werden über `implements` ererbt. In der erbenden Klasse müssen alle Methoden des Interface implementiert werden.
- Klassen können mehrere Interfaces implementieren

Elemente mit gleichem Namen

- Wie schon zuvor erwähnt kann es durchaus Methoden geben die denselben Methodennamen besitzen aber eine unterschiedliche Funktionalität darstellen
- **Überladung:** Die Methode wird mit verschiedenen Parametern mehrfach implementiert (ähnlich zu mehreren Konstruktoren). Auch wenn der Name einer Methode in der Basisklasse schonmal verwendet wurde ist dies eine Überladung.
- **Verdeckung**: Wenn in einer abgeleiteten Klasse ein Attribut definiert wird , dass denselben Namen wie ein Attribut der Basisklasse trägt, so wird das Attribut der Basisklasse verdeckt. Der Zugriff auf verdeckte Attribute kann mit einem Cast zu einem Objekt der Basisklasse erfolgen
- **Überschreibung:** Das selbe gilt für Methoden. Hier muss allerdings die Signatur gleich der Signatur der Basisklasse sein.(Beim Überschreiben darf lediglich der Zugriff gelockert werden)

Innere Klassen

- Innere Klassen werden meist als Hilfsklasse der übergeordneten Klasse benötigt
- Sinnvoll ist eine Realisierung als inner Klasse wenn diese nur von einer einzigen Klasse benötigt wird.
- Die innere Klasse kann durch `protected` und `private` von anderen Klassen des Pakets verborgen werden
- Die innere Klasse kann auf alle Elemente der äußeren Klasse zugreifen und umgekehrt ist das genauso möglich
- Der Zugriff auf eine innere Klasse ist auch von Außen möglich wenn diese nicht als `private` deklariert wurde

```
Aussen.Innen innenobj = aussenObj.new Innen();
```

Vielen Dank für die Aufmerksamkeit!