Programmieren 1 – Live Übung

Vererbung

***Aufgabe 1: Generalisieren – Gemeinsamkeiten entdecken***

Für die Immobilienfirma Exklusiv & Teuer soll ein Softwaresystem entwickelt werden. Eine erste Analyseaktivität ergab folgende zwei Klassen:





1. Vergleichen Sie die beiden Klassen. Was fällt Ihnen auf? Achten Sie dabei nicht nur auf die Namen, sondern auf die Semantik, die dahinter steckt.  
   Attribut adresse tritt bei beiden Klassen auf

eigentümer und besitzer  
erstellungsjahr und baujahr  
verkaufspreis und preis  
getVerkaufspreis() und getPreis() haben zwar unterschiedliche Bezeichnungen aber sagen eigentlich dasselbe aus

1. Entwerfen Sie eine Vererbungsstruktur, indem Sie eine Oberklasse für die beiden Klassen mit geeigneten Namen finden. Finden Sie dazu sowohl für die Oberklasse als auch für die beiden entstehenden Unterklassen geeignete Namen für die Attribute und Methoden.



UML: man spricht von Gerneralisierungs-/Spezialierungshierarchie

Objektorientierten Programmierung: Vererbung

***Aufgabe 2: Umsetzung der Klassen***

Implementieren Sie die die drei Klassen Ihrer Vererbungsstruktur aus Aufgabe 1. Berücksichtigen Sie dabei nur eine Auswahl der oben angeführten Attribute. Folgender Programmablauf sollte möglich sein:

Daten zum Einfamilienhaus

Haustyp: Stadthaus

Besitzer: Müller

Verkaufspreis: 300000

Daten zum Geschäftshaus

Anzahl Büroräume: 10

Besitzer: Winter

Verkaufspreis: 1500000

Lösung:

**public** **abstract** **class** Immobilie {

// Attribute

**protected** String besitzer = " ";

**private** **int** verkaufspreis = 0;

// Konstruktor

**protected** Immobilie(String besitzer, **int** verkaufspreis) {

**this**.besitzer = besitzer;

**this**.verkaufspreis = verkaufspreis;

}

**public** String getBesitzer(){

**return** besitzer;

}

**public** **int** getVerkaufspreis(){

**return** verkaufspreis;

}

}

**public** **class** Einfamilienhaus **extends** Immobilie {

//Attribute

**private** String haustyp = "Stadthaus";

//Konstruktor

**public** Einfamilienhaus(String besitzer, **int** verkaufspreis, String haustyp){

**super**(besitzer, verkaufspreis);

**this**.haustyp = haustyp;

}

**public** String getHaustyp(){

**return** haustyp;

}

}

**public** **class** Geschaeftshaus **extends** Immobilie {

//Attribute

**private** **int** anzahlBueroraeume = 0;

//Konstruktor

**public** Geschaeftshaus(String besitzer, **int** verkaufspreis, **int** anzahlBueroraeume){

**super**(besitzer, verkaufspreis);

**this**.anzahlBueroraeume = anzahlBueroraeume;

}

**public** **int** getAnzahlBueroraeume(){

**return** anzahlBueroraeume;

}

}

**import** java.util.\*;

**public** **class** ImmobilienUI {

**public** **static** **void** main(String[] args) {

// **TODO** Auto-generated method stub

Einfamilienhaus einHaus = **new** Einfamilienhaus("Müller", 3000000, "Stadthaus");

Geschaeftshaus nocheinHaus = **new** Geschaeftshaus("Winter", 1500000, 10);

System.***out***.println("Daten zum Einfamilienhaus:");

System.***out***.println("Haustyp: " + einHaus.getHaustyp());

System.***out***.println("Besitzer: " + einHaus.getBesitzer());

System.***out***.println("Verkaufspreis: " + einHaus.getVerkaufspreis());

System.***out***.println("\nDaten zum Geschäftshaus:");

System.***out***.println("Anzahl Büroräume: " + nocheinHaus.getAnzahlBueroraeume());

System.***out***.println("Besitzer: " + nocheinHaus.getBesitzer());

System.***out***.println("Verkaufspreis: " + nocheinHaus.getVerkaufspreis());

}

}

***Aufgabe 3: Konstruktoren***

Schreiben Sie zu den drei Klassen die jeweils passenden Konstruktoren. Berücksichtigen Sie dabei folgende Aspekte:

* Der Konstruktor der Oberklasse soll zwei Aufrufparameter besitzen, nämlich der Besitzer sowie der Verkaufspreis.
* Dem Konstruktor der Unterklasse Einfamilienhaus soll drei Aufrufparameter haben, nämlich der Besitzer, der Verkaufspreis sowie der Haustyp.
* Der Konstruktor der Unterklasse Geschäftshaus soll drei Aufrufparameter besitzen, nämlich der Besitzer, der Verkaufspreis sowie die Anzahl der Büroräume.

Lösung:

// Konstruktor

**protected** Immobilie(String besitzer, **int** verkaufspreis) {

**this**.besitzer = besitzer;

**this**.verkaufspreis = verkaufspreis;

}

//Konstruktor

**public** Einfamilienhaus(String besitzer, **int** verkaufspreis, String haustyp){

**super**(besitzer, verkaufspreis);

**this**.haustyp = haustyp;

}

//Konstruktor

**public** Geschaeftshaus(String besitzer, **int** verkaufspreis, **int** anzahlBueroraeume){

**super**(besitzer, verkaufspreis);

**this**.anzahlBueroraeume = anzahlBueroraeume;

}

***Aufgabe 4: Abstrakte Klasse***

Berücksichtigen Sie folgenden Aspekt in der Implementierung: Bei der Immobilienfirma Exklusiv & Teuer ist jede Immobilie entweder ein Einfamilienhaus oder ein Geschäftshaus. Daher ist es nicht sinnvoll von der Oberklasse Objekte zu erzeugen. Sie soll als abstrakte Klasse gekennzeichnet werden.

**public** **abstract** **class** Immobilie {

// Attribute

**protected** String besitzer = " ";

**private** **int** verkaufspreis = 0;

// Konstruktor

**protected** Immobilie(String besitzer, **int** verkaufspreis) {

**this**.besitzer = besitzer;

**this**.verkaufspreis = verkaufspreis;

}

**public** String getBesitzer(){

**return** besitzer;

}

**public** **int** getVerkaufspreis(){

**return** verkaufspreis;

}

}

***Aufgabe 5: Anwendungsklasse schreiben***

Schreiben Sie abschließend eine Anwendungsklasse, die den in Aufgabe 2 beschriebenen Testfall simuliert.

**import** java.util.\*;

**public** **class** ImmobilienUI {

**public** **static** **void** main(String[] args) {

// **TODO** Auto-generated method stub

Einfamilienhaus einHaus = **new** Einfamilienhaus("Müller", 3000000, "Stadthaus");

Geschaeftshaus nocheinHaus = **new** Geschaeftshaus("Winter", 1500000, 10);

System.***out***.println("Daten zum Einfamilienhaus:");

System.***out***.println("Haustyp: " + einHaus.getHaustyp());

System.***out***.println("Besitzer: " + einHaus.getBesitzer());

System.***out***.println("Verkaufspreis: " + einHaus.getVerkaufspreis());

System.***out***.println("\nDaten zum Geschäftshaus:");

System.***out***.println("Anzahl Büroräume: " + nocheinHaus.getAnzahlBueroraeume());

System.***out***.println("Besitzer: " + nocheinHaus.getBesitzer());

System.***out***.println("Verkaufspreis: " + nocheinHaus.getVerkaufspreis());

}

}

***Aufgabe 6: Methoden redefinieren***

Eine Bank verwaltet bisher ihre Kunden mit einem Programm. Jeder Kunde kann ein Konto besitzen. Ein Konto ist dadurch gekennzeichnet, dass der Kunden sowohl einen positiven als auch negativen Kontostand besitzen kann. Bei der Realisierung wurde folgendes Klassendiagramm umgesetzt:



Es sollen jetzt mit demselben Programm zusätzlich Sparkonten verwaltet werden. Jeder Kunde kann neben einem Konto auch ein Sparkonto besitzen – oder auch nur ein Sparkonto. Eine Analyse der Klasse Konto ergibt, dass Sparkonto über dieselben Attribute und Operationen wie ein Konto verfügt. Allerdings darf der Kontostand eines Sparkontos nie negativ sein. Deshalb muss die Operation buchen() redefiniert bzw. überschrieben werden. Folgendes Klassendiagramm veranschaulicht die Zusammenhänge:



Schreiben Sie die Klasse Sparkonto.

Lösung:

**public** **class** Sparkonto **extends** Konto {

//Konstruktor

**public** Sparkonto(Kunde meinKunde){

**super**(meinKunde);

}

//Methode buchen redefinieren

**public** **void** buchen(**int** betrag){

**if** (getKontostand() + betrag > 0){

**super**.buchen(betrag);

}

}

}

***Aufgabe7: Schnittstellen***

Es soll ein Programm zur Sprachauswahl geschrieben werden, bei dem die Benutzungsoberfläche auf verschiedene Sprachen eingestellt werden kann. Zunächst sollen die Sprachen Deutsch und Englisch wähbar sein. Später sollen aber noch weitere Sprachen hinzugefügt werden, ohne das Programm neu schreiben zu müssen. Das Klassendiagramm zeigt das konzipierte Schnittstellenkonzept:



Setzen Sie die folgenden Klassen um:

1. Klasse, die das Interface DialogI repräsentiert

Lösung:

**public** **interface** DialogI {

//abstrakte Operation

**void** zeigeDialog();

}

1. Klasse DialogDeutsch  
   (Methode zeigeDialog gibt den Text „Herzlich willkommen“ auf dem Bildschirm aus.  
   Methode zeigeImpressum gibt den Text „Impressum“ auf dem Bildschirm aus.)

Lösung:

**public** **class** DialogDeutsch **implements** DialogI {

**public** **void** zeigeDialog(){

System.***out***.println("Herzlich willkommen!");

}

}

1. Klasse DialogEnglisch  
   (Methode zeigeDialog gibt den Text „Welcome“ auf dem Bildschirm aus.)

Lösung:

**public** **class** DialogEnglisch **implements** DialogI {

**public** **void** zeigeDialog() {

System.***out***.println("Welcome!");

}

}

**Anwendungsklasse:**

**import** java.util.Scanner;

**public** **class** Sprachauswahl {

**public** **static** **void** main(String[] args) {

System.***out***.println("Bitte Spache waehlen." + "Please choose your language.");

System.***out***.println("Deutsch: 1 Englisch: 2");

Scanner eingabe = **new** Scanner(System.***in***);

**int** sprache = eingabe.nextInt();

DialogI einDialog;

**if** (sprache == 1){

einDialog = **new** DialogDeutsch();

}

**else** {

einDialog = **new** DialogEnglisch();

}

//ab hier ist die Sprache eingestellt

einDialog.zeigeDialog();

}

}