**JURUSAN TEKNIK INFORMATIKA**

**UNIVERSITAS SANATA DHARMA**

**PRAKTIKUM PBO I**

**MODUL XII**

**Abstract Class dan Interface**

1. **TUJUAN PRAKTIKUM**
2. Mahasiswa mampu memahami konsep abstract class dan interface.
3. Mahasiswa mampu mengimplementasikan abstract class dan interface.
4. **ABSTRACT CLASS**

public class GraduateStudent extends Student {

private int jumlah;

public int getJumlah()

{

return jumlah;

}

public String getCourseGrade() {

int total = 0;

for (int i = 0; i < NUM\_OF\_TESTS; i++) {

total += test[i];

}

if (total / NUM\_OF\_TESTS >= 85) {

courseGrade = "Pass";

} else {

courseGrade = "No Pass";

}

return courseGrade;

}

}

public class UndergraduateStudent extends Student {

@Override

public String getCourseGrade() {

int total = 0;

for (int i = 0; i < NUM\_OF\_TESTS; i++) {

total += test[i];

}

if (total / NUM\_OF\_TESTS >= 80) {

courseGrade = "Pass";

} else {

courseGrade = "No Pass";

}

return courseGrade;

}

}

public class Student {

protected final static int NUM\_OF\_TESTS = 3;

protected String name;

protected int[] test;

protected String courseGrade;

public Student( ) {

this("No Name");

}

public Student(String studentName) {

name = studentName;

test = new int[NUM\_OF\_TESTS];

courseGrade = "\*\*\*\*";

}

public String getCourseGrade( ) {

return courseGrade;

}

.........

}

public class StudentTest {

public static void main(String[] args)

{

Scanner sc=new Scanner(System.in);

Student[] roster = new Student[40];

...........................

for (int i = 0; i < (ju+jg); i++)

{

System.out.println(roster[i].getName());

System.out.println(roster[i].getCourseGrade());

....................................

}

Kelas UndergraduateStudent dan GraduateStudent meng-override method getCourseGrade() pada kelas Student. Implementasi method getCourseGrade berbeda pada setiap sub kelasnya. Dengan kelas-kelas pewarisan seperti pada contoh di bawah kita pastikan bahwa semua Student adalah Undergradute atau Gradute, dan tidak ada instansiasi dari kelas Student, maka isi dari method getCourseGrade() pada kelas Student tidak pernah akan dikerjakan.

Jika begitu, bagaimana jika method getCourseGrade pada kelas Student dihilangkan ?. Hal ini akan menyebabkan kita tidak dapat mengakses method tersebut jika menggunakan prinsip polimorfisme, seperti pada contoh kode program :

System.out.println(roster[i].getCourseGrade());

Java menyediakan konsep **abstract class**, yang memungkinkan kita untuk membuat method abstract. **Method abstract** adalah method yang tidak memiliki badan method, tetapi memiliki header method dengan nama method, tipe pengembalian method dan parameter method yang ditentukan.

Sebuah **abstract class** adalah sebuah kelas yang memerlukan definisi method yang pasti tetapi tidak perlu mendefinisikan implementasi method secara eksplisit. Sebuah **abstract class** memiliki syarat :

* minimal satu abstract method
* kelas tersebut tidak dapat diinstansiasi

Sehingga kelas Student pada contoh diatas menjadi :

abstract class Student {

protected final static int NUM\_OF\_TESTS = 3;

protected String name;

protected int[] test;

protected String courseGrade;

public Student( ) {

this("No Name");

}

public Student1(String studentName) {

name = studentName;

test = new int[NUM\_OF\_TESTS];

courseGrade = "\*\*\*\*";

}

public abstract String getCourseGrade();

.................

}

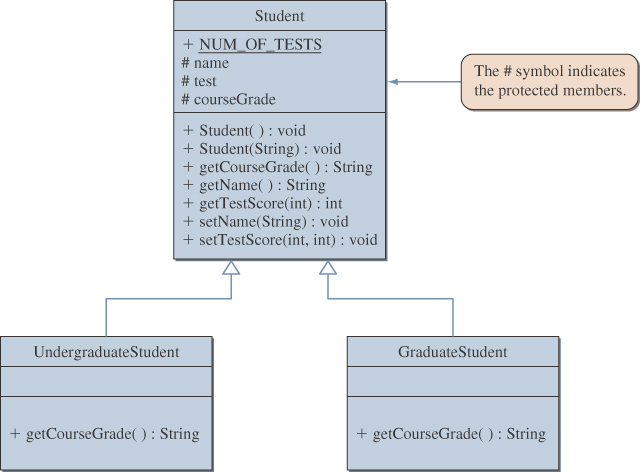
Implementasi kelas abstract diatas tidak akan mengubah apapun pada implementasi kelas StudentTest. Tetapi untuk semua subclass dari kelas Student, harus mengimplementasikan method getCourseGrade() atau membuat method getCourseGrade() dengan badan atau isi methodnya.

1. **INTERFACE**

Dengan abstract class, maka kita dapat mendefinisikan perilaku kelas tanpa memperhatikan implementasi method. Dalam suatu kasus, kita perlu memastikan bahwa suatu kelas harus memiliki perilaku tertentu, tanpa harus memperhatikan atribut yang dipunyai. Sebagai contoh, kita menginginkan bahwa yang namanya Learner pasti memiliki status kelulusan berdasar tes yang diambil. Karena hal ini, maka kita dapat memiliki interface Learner yang memiliki method abstract :

void getCourseGrade()

Semua kelas yang memiliki perilaku seorang Learner akan menjadi sub kelas dari interface Learner tersebut. Sehingga kelas tersebut akan mengimplementasikan semua method abstract



<<interface>>Learner

+ getCourseGrade():String

Sehingga kelas Student akan berubah, sedangkan kelas lain akan tetap.

public interface Learner {

public String getCourseGrade();

}

public class Student implements Learner {

protected final static int NUM\_OF\_TESTS = 3;

protected String name;

protected int[] test;

protected String courseGrade;

public Student( ) {

this("No Name");

}

public Student(String studentName) {

name = studentName;

test = new int[NUM\_OF\_TESTS];

courseGrade = "\*\*\*\*";

}

@Override

public String getCourseGrade() {

return courseGrade;

}

................

}

1. **LANGKAH PERCOBAAN**

1. Ketik dan jalankan program dibawah ini. Jika terjadi kesalahan, berikan alasan dan betulkan

//Interface Leaner

public interface Learner {

public String getCourseGrade();

}

//Class abstract Student yang mewariskan ke class GraduateStudent dan UndergraduateStudent

public abstract class Student {

protected final static int NUM\_OF\_TESTS = 3;

protected String name;

protected int[] test;

protected String courseGrade;

public Student( ) {

this("No Name");

}

public Student(String studentName) {

name = studentName;

test = new int[NUM\_OF\_TESTS];

courseGrade = "\*\*\*\*";

}

//public abstract String getCourseGrade();

public String getName( ) {

return name;

}

public int getTestScore(int testNumber) {

return test[testNumber-1];

}

public void setName(String newName) {

name = newName;

}

public void setTestScore(int testNumber, int testScore) {

test[testNumber-1] = testScore;

}

}

//Class GraduateStudent yang mewarisi Student dan mengiplement Learner

public class GraduateStudent extends Student implements Learner{

private int jumlah;

public int getJumlah()

{

return jumlah;

}

@Override

public String getCourseGrade() {

int total = 0;

for (int i = 0; i < NUM\_OF\_TESTS; i++) {

total += test[i];

}

if (total / NUM\_OF\_TESTS >= 85) {

courseGrade = "Pass";

} else {

courseGrade = "No Pass";

}

return courseGrade;

}

}

//Class UndergraduateStudent yang mewarisi Student dan mengiplement Learner

public class UndergraduateStudent extends Student implements Learner{

public UndergraduateStudent(){

};

@Override

public String getCourseGrade() {

int total = 0;

for (int i = 0; i < NUM\_OF\_TESTS; i++) {

total += test[i];

}

if (total / NUM\_OF\_TESTS >= 80) {

courseGrade = "Pass";

} else {

courseGrade = "No Pass";

}

return courseGrade;

}

}

//Class yang memauat metode Main

import java.util.Scanner;

public class StudentTest {

public static void main(String[] args){

Scanner sc=new Scanner(System.in);

Student[] roster = new Student[40];

System.out.print("Berapa banyak Undergraduate Students ? ");

int ju=sc.nextInt();

System.out.print("Berapa banyak Graduate Students? ");

int jg=sc.nextInt();

System.out.println("Masukkan data Under");

for (int i=0;i<ju;i++){

System.out.print("Masukkan nama Under ke-"+(i+1)+" ");

roster[i] = new UndergraduateStudent();

roster[i].setName(sc.next());

System.out.print("\nMasukkan skor test ke 1 data under-"+(i+1)+" ");

roster[i].setTestScore(1, sc.nextInt());

System.out.print("\nMasukkan skor test ke 2 data under-"+(i+1)+" ");

roster[i].setTestScore(2, sc.nextInt());

System.out.print("\nMasukkan skor test ke 3 data under-"+(i+1)+" ");

roster[i].setTestScore(3, sc.nextInt());

}

System.out.println("\nMasukkan data Grad");

for (int i=ju;i<ju+jg;i++)

{

System.out.println("Masukkan nama Grad ke-"+(i+1) +" ");

roster[i] = new GraduateStudent();

roster[i].setName(sc.next());

System.out.print("\nMasukkan skor test ke 1 data grad-"+(i+1)+" ");

roster[i].setTestScore(1, sc.nextInt());

System.out.print("\nMasukkan skor test ke 2 data grad-"+(i+1)+" ");

roster[i].setTestScore(2, sc.nextInt());

System.out.print("\nMasukkan skor test ke 3 data grad-"+(i+1)+" ");

roster[i].setTestScore(3, sc.nextInt());

}

//Membentuk obyek dari interface Leaner

Learner murid = null;

for (int i = 0; i < (ju+jg); i++){

if(roster[i] instanceof GraduateStudent) {

murid = new GraduateStudent();

murid = (GraduateStudent)roster[i];

}

else {

murid = new UndergraduateStudent();

murid = (UndergraduateStudent)roster[i];

}

System.out.println(roster[i].getName());

System.out.println(murid.getCourseGrade());

}

}

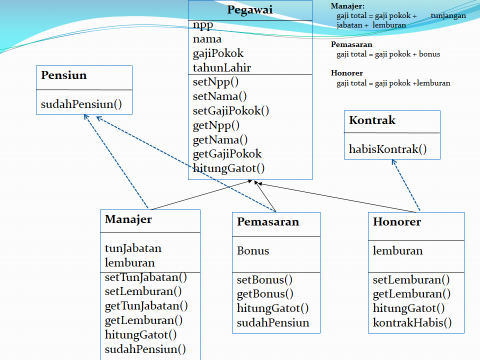
}

2. Ujilah pembuatan interface pada contoh interface Learner di atas, dan ujilah kembali StudentTest pada soal langkah percobaan 1 diatas. Jelaskan pembuatan interface, dan perubahan pada kelas Student.

1. **TUGAS**

1. Kasus yang anda buat tetap menggunakan kelas Pegawai, Marketing, Manajer dan Honorer pada pertemuan sebelumnya.

* Ubahlah kelas pegawai menjadi kelas abstrak. (carilah apa yang menjadi method abstrak)
* Tambahkan interface seperti pada diagram class berikut ini
* Implementasikan interface tersebut pada seluruh pegawai



Perhatikan bahwa di diagram class di atas ada 2 buah interface yang diimplementasi oleh class yang terhubung. Selian itu ada class abstrak Pegawai yang mewariskan ke 3 class lain. Interface Pensiun akan menentukan kapan tahun akan pensiun berdasar usia yakni di atas 55 dan interface Kontrak menentukan kapan habis kontrak berdasar tahun lahir ditambah 10 tahun.

1. **LAPORAN PRAKTIKUM**

Buatlah laporan dari hasil menjalankan dan menganalisa program Anda dalam modul 13, yang mencakup bagian **LANGKAH PERCOBAAN** dan **TUGAS**. Laporan memuat:

1. Tujuan praktikum (5%)
2. Listing program (hasil *capture screen*) (10%)
3. Output program (hasil *capture screen*) (10%)
4. Hasil dan analisa dari Langkah Percobaan 1 (15%)
5. Hasil dan analisa dari Langkah Percobaan 2 (10%)
6. Analisa tugas yang berisi:
   * Maksud/tujuan umum program (10%)
   * Diagram kelas beserta struktur hubungan antar kelas (10%)
   * Penjelasan atas tiap baris perintah program dalam tiap kelas (20%)
7. Daftar pustaka yang dipakai untuk membuat laporan (10%)

**Daftar Pustaka**

Wu, C. Thomas. 2006. *An Introduction to Object-Oriented Programming with Java*. Mc Graw-Hill International Edition. 4th edition.

👍👍 Seize the day !!👍👍