Nadila Jannatul Ma'wa

20220040199

TI22C

SESI 3

PERCOBAAN 1

```
class Parent {
    public int x = 5;
}

class Child extends Parent {
    public int x = 10;
    public void Info(int x) {
    System.out.println("Nilai x sebagai parameter = " + x);
    System.out.println("Data member x di class Child = " + this.x);
    System.out.println("Data member x di class Parent = " +
super.x);
    }
}

public class NilaiX {
    public static void main(String args[]) {
        Child tes = new Child();
        tes.Info(20);
    }
}
```

Kode di atas mendefinisikan dua kelas, yaitu kelas **Parent** dan kelas **Child**, serta sebuah kelas **NilaiX** yang memiliki metode **main** sebagai entry point program. Kelas **Parent** memiliki sebuah atribut publik **x** dengan nilai awal 5. Kelas **Child** merupakan turunan dari kelas **Parent** dan memiliki atribut publik **x** sendiri dengan nilai awal 10. Kelas **Child** juga memiliki metode **info** yang menerima parameter **x**.

Dalam metode **info**, terdapat tiga pernyataan cetak yang menampilkan nilai **x**: nilai **x** sebagai parameter, nilai **x** dari atribut **x** di kelas **Child**, dan nilai **x** dari atribut **x** di kelas **Parent**. Dalam metode **main**, sebuah objek dari kelas **Child** dibuat dan metode **info** dipanggil dengan argumen 20.

Output :

```
Nilai x sebagai parameter = 20
Data member x di class Child = 10
Data member x di class Parent = 5
```

this.x merujuk ke nilai **x** yang ada di dalam kelas **Child**, sedangkan **super.x** merujuk ke nilai **x** yang ada di dalam kelas **Parent**. Dalam kasus ini, nilai **x** yang diambil dari kelas **Child** adalah 10, sementara nilai **x** yang diambil dari kelas **Parent** adalah 5.

PERCOBAAN 2

```
public class Pegawai {
   private String nama;
   public double gaji;
}

public class Manajer extends Pegawai {
   public String departemen;

   public void IsiData(String n, String d) {
      nama=n;
      departemen=d;
   }
}
```

- Kode ini mencoba menggambarkan hubungan antara kelas **Pegawai** (sebagai superclass atau kelas induk) dan **Manajer** (sebagai subclass atau turunan).
- Dengan mewarisi atribut **nama** dan **gaji**, kelas **Manajer** dapat mengakses dan menggunakan atribut-atribut tersebut.
- Atribut **nama** dari kelas **Pegawai** dideklarasikan sebagai private, yang mengikuti prinsip enkapsulasi untuk melindungi data dari akses langsung.
- Penggunaan konstruktor dan metode setter untuk mengatur nilai atribut memberikan fleksibilitas dalam inisialisasi objek dan memastikan integritas data.
- Namun, kode tersebut masih terbatas dan belum optimal. Sebagai contoh, tidak ada metode untuk mengambil nilai atribut **nama** dan **gaji**, yang dapat membuat akses ke informasi tersebut sulit dari luar kelas **Pegawai**. Selain itu, tidak ada validasi atau pemrosesan tambahan yang dilakukan pada data yang dimasukkan ke dalam kelas. Selanjutnya, perlu dipertimbangkan untuk menambahkan metode getter untuk mengakses nilai atribut secara aman.

```
// PERCOBAAN 2
class Pegawai {
    public String nama;
    public double gaji;
}

public class Manajer extends Pegawai {
    public String departemen;

    public void IsiData(String n, String d) {
        nama = n;
        departemen = d;
    }
}
```

#Modifier private pada attribute nama di ubah menjadi public agar dapat digunakan pada class Manajer

#Modifier pada attribute nama masih bisa menggunakan private apabila di tambahkan dengan method setter untuk memungkinkan class Manajer mengubah nilai atribut nama.

PERCOBAAN 3

```
public class Parent {
     // kosong
}

public class Child extends Parent {
    int x;
    public Child() {
        x = 5;
    }
}
```

- Kode ini menunjukkan konsep dasar pewarisan dalam pemrograman berorientasi objek (OOP), di mana kelas **Child** mewarisi dari kelas **Parent**.
- Meskipun kelas **Parent** kosong, namun ini mengilustrasikan bahwa pewarisan dapat terjadi bahkan jika tidak ada atribut atau metode yang diwarisi.
- Kode ini dapat dianggap sebagai contoh dasar dari penggunaan pewarisan dalam Java, namun dalam praktiknya, pewarisan biasanya digunakan untuk mengambil atau memodifikasi perilaku dan data dari kelas induk.

```java
// PERCOBAAN 3
class Parent {
    // kosong

}

public class Child extends Parent {
    int x;
    public Child() {
        x = 5;
    }
}
```

PERCOBAAN 4

```
class Employee {
     private static final double BASE_SALARY = 15000.00;
     private String Name = ""; private
     double Salary = 0.0; private Date
     birthDate;

     public Employee() {}
     public Employee(String name, double salary, Date DoB){
          this.Name=name;
          this.Salary=salary;
          this.birthDate=DoB;
     }
     public Employee(String name,double salary){
          this(name,salary,null);
     }
     public Employee(String name, Date DoB){
          this(name,BASE_SALARY,DoB);
     }
     public Employee(String name){
          this(name,BASE_SALARY);
     }

     public String GetName(){ return Name;}
     public double GetSalary(){ return Salary; }
}

      class Manager extends Employee {

     //tambahan attribrute untuk kelas manager private
     String department;

     public Manager(String name,double salary,String dept){
          super(name,salary);
          department=dept;
     }
     public Manager(String n,String dept){
          super(n);
          department=dept;
     }
     public Manager(String dept){
          super();
          department=dept;
     }
     public String GetDept(){
          return department;
     }
}

public class TestManager {

     public static void main(String[] args) {
          Manager Utama = new Manager("John",5000000,"Financial");
          System.out.println("Name:"+ Utama.GetName());
          System.out.println("Salary:"+ Utama.GetSalary());
          System.out.println("Department:"+ Utama.GetDept());

          Utama = new Manager("Michael","Accounting");
          System.out.println("Name:"+ Utama.GetName());
          System.out.println("Salary:"+ Utama.GetSalary());
          System.out.println("Department:"+ Utama.GetDept());
     }
}
```

- Pewarisan antara kelas Manager dan Employee. Sebagai turunan dari kelas Employee, kelas Manager dapat mengakses atribut dan metode yang ada pada kelas Employee.
- Konstruktor kelas Manager memanggil konstruktor superclass Employee menggunakan kata kunci super().
- Penggunaan metode getter pada kelas Employee dan Manager memungkinkan untuk mengakses atribut yang bersifat private secara aman.
- Dengan penggunaan kelas TestManager, kita dapat menguji fungsionalitas dari kelas Manager dan melihat bagaimana pewarisan dan konstruktor bekerja dalam praktiknya.

```
// PERCOBAAN 4
import java.util.Date;

class Employee {
    private static final double BASE_SALARY = 15000.00;
    private String Name = "";
    private double Salary = 0.0;
    private Date birthDate;

    public Employee() {}
    public Employee (String name, double salary, Date Dob){
        this.Name = name;
        this.Salary = salary;
        this.birthDate = Dob;
    }
    public Employee(String name, double salary) {
        this(name,salary,null);
    }
    public Employee(String name, Date DoB) {
        this(name,BASE_SALARY,DoB);
    }
    public Employee(String name) {
        this(name,BASE_SALARY);
    }

    public String GetName() {
        return Name;
    }
    public double GetSalary() {
        return Salary;
    }
}
```

```java
class Manager extends Employee {
    private String department;

    public Manager(String name, double salary, String dept) {
        super(name,salary);
        department = dept;
    }
    public Manager (String n, String dept) {
        super(n);
        department = dept;
    }
    public Manager (String dept) {
        super();
        department = dept;
    }
    public String GetDept() {
        return department;
    }
}

public class TestManager {

    public static void main(String[] args) {
        Manager Utama = new Manager("john",5000000,"Financial");
        System.out.println("Name: "+Utama.GetName());
        System.out.println("Salary: "+Utama.GetSalary());
        System.out.println("Department: "+Utama.GetDept());

        Utama = new Manager("Michael","Accounting");
        System.out.println("Name: "+Utama.GetName());
        // System.out.println("Salary: "+Utama.GetSalary()); // Tidak ada metode
GetSalary di kelas Manager
        System.out.println("Department: "+Utama.GetDept());
    }
}
```

#Menambahkan java.util.Date; agar type data date dapat di gunakan pada class file tersebut

Output :

```
Name: john
Salary: 5000000.0
Department: Financial
Name: Michael
Department: Accounting
```

PERCOBAAN 5

```
public class MoodyObject {

    protected String getMood(){
        return "moody";
    }
    public void speak(){
        System.out.println("I am"+getMood());
    }
    void laugh() {}
    void cry() {}
}

public class SadObject extends MoodyObject{
    protected String getMood(){
        return "sad";
    }
    public void cry(){
        System.out.println("Hoo hoo");
    }
}

public class HappyObject extends MoodyObject{
    protected String getMood(){
        return"happy";
    }
    public void laugh(){
    System.out.println("Hahaha");
    }
}

public class MoodyTest {
    public static void main(String[] args) {

        MoodyObject m = new MoodyObject();

        //test perent class
        m.speak();
```

- Kode ini mengilustrasikan konsep polimorfisme di mana objek dari kelas turunan dapat diakses melalui referensi kelas dasarnya.
- Karena metode **speak()** dipanggil pada objek yang diinisialisasi sebagai **MoodyObject**, maka metode **getMood()** yang dipanggil adalah versi yang ditentukan dalam kelas turunannya (**SadObject** atau **HappyObject**).
- Dengan menggunakan polimorfisme, kita dapat membuat kode yang lebih fleksibel dan dapat diubah tanpa perlu memodifikasi implementasi di banyak tempat.
- Selain itu, penggunaan pewarisan memungkinkan kita untuk mengelompokkan dan mengorganisir kode dengan lebih baik, dengan menghindari duplikasi kode dan meningkatkan keterbacaan dan pemeliharaan.

```java
//PERCOBAAN 5
class MoodyObject {
    protected String getMood(){
        return "Moody";
    }
    public void speak() {
        System.out.println("I am "+ getMood());
    }
    void laugh() {

    }
    void cry() {

    }
}

class SadObject extends MoodyObject {
    protected String getMood() {
        return "sad";
    }
    public void cry(){
        System.out.println("Hoo hoo");
    }
}

class HappyObject extends MoodyObject {
    protected String getMood() {
        return "happy";
    }
    public void laugh(){
        System.out.println("Hahaha");
    }
}

public class MoodyTest {
    public static void main(String[] args) {
        MoodyObject m = new MoodyObject();
        m.speak();

        m = new HappyObject();
        m.speak();
        m.laugh();

        m = new SadObject();
        m.speak();
```

```
        m.cry();
    }
}
```

#Modifier public digunakan pada class MoodyTest saja, karna class MoodyTest digunakan untuk penamaan file

Output :

```
I am Moody
I am happy
Hahaha
I am sad
Hoo hoo
```

PERCOBAAN 6

```
class A {
    String var_a = "Variabel A";
    String var_b = "Variabel B";
    String var_c = "Variabel C";
    String var_d = "Variabel D";

    A(){
        System.out.println("Konstruktor A dijalankan");
    }
}

class B extends A{
    B(){
        System.out.println("Konstruktor B dijalankan ");
        var_a = "Var_a dari class B";
        var_b = "Var_a dari class B";
    }

    public static void main(String args[]){

        System.out.println("Objek A dibuat");
        A aa= new A();
        System.out.println("menampilkan nama variabel obyek aa");
        System.out.println(aa.var_a);
        System.out.println(aa.var_b);
        System.out.println(aa.var_c);
        System.out.println(aa.var_d);
        System.out.println("");

        System.out.println("Objek B dibuat");
        B bb= new B();
        System.out.println("menampilkan nama variabel obyek bb");
        System.out.println(bb.var_a);
        System.out.println(bb.var_b);
        System.out.println(bb.var_c);
        System.out.println(bb.var_d);
    }
}
```

- Pewarisan atau inheritance digunakan di sini dengan membuat kelas **B** yang merupakan turunan dari kelas **A**. Dengan demikian, kelas **B** mewarisi atribut dan metode dari kelas **A**.
- Konstruktor kelas **B** dapat mengakses atribut yang diwarisi dari kelas **A** dan mengubah nilainya.
- Metode **main()** digunakan untuk menguji fungsionalitas kelas-kelas tersebut dengan membuat objek dari masing-masing kelas dan mencetak nilai atributnya.
- Dengan pewarisan dan penggunaan konstruktor di dalam kelas turunan, kita dapat membuat kode yang lebih efisien dan mudah di-maintain.

```
// PERCOBAAN 6
class A {
    String var_a = "Variabel A";
    String var_b = "Variabel B";
    String var_c = "Variabel C";
    String var_d = "Variabel D";

    A(){
        System.out.println("Konstruktor A dijalankan");
    }
}
```

```java
class B extends A{
    B() {
        System.out.println("Konstruktor B dijalankan");
        var_a = "Var_a dari class B";
        var_b = "Var_b dari class B";
    }

    public static void main(String[] args) {

        System.out.println("Objek A dibuat");
        A aa = new A();
        System.out.println("Menampilkan proyek nama dari aa");
        System.out.println(aa.var_a);
        System.out.println(aa.var_b);
        System.out.println(aa.var_c);
        System.out.println(aa.var_d);
        System.out.println("");

        System.out.println("Objek B dibuat");
        B bb = new B();
        System.out.println("Menampilkan proyek nama dari bb");
        System.out.println(bb.var_a);
        System.out.println(bb.var_b);
        System.out.println(bb.var_c);
        System.out.println(bb.var_d);


    }
}
```

Output :

```
Objek A dibuat
Konstruktor A dijalankan
Menampilkan proyek nama dari aa
Variabel A
Variabel B
Variabel C
Variabel D

Objek B dibuat
Konstruktor A dijalankan
Konstruktor B dijalankan
Menampilkan proyek nama dari bb
Var_a dari class B
Var_b dari class B
Variabel C
Variabel D
```

PERCOBAAN 7

```
class Bapak {
    int a;
    int b;

    void show_variabel(){
        System.out.println("NIlai a="+ a);
        System.out.println("NIlai b="+ b);
    }
}

class Anak extends Bapak{
    int c;
    void show_variabel(){
        System.out.println("NIlai a="+ a);
        System.out.println("NIlai b="+ b);
        System.out.println("NIlai c="+ c);
    }
}

public class InheritExample {

    public static void main(String[] args) {

        Bapak objectBapak = new Bapak();
        Anak objectAnak = new Anak();

        objectBapak.a=1;
        objectBapak.b=1;
        System.out.println("Object Bapak (Superclass):");

        objectBapak.show_variabel();
        objectAnak.c=5;
        System.out.println("Object Anak (Superclass dari Bapak):");
        objectAnak.show_variabel();
    }
}
```

- Dengan menggunakan pewarisan, kelas **Anak** dapat mewarisi atribut dan metode dari kelas **Bapak**, sehingga mengurangi duplikasi kode.
- Dalam kasus ini, metode **show_variabel()** di kelas **Anak** di-override untuk menambahkan perilaku tambahan.
- Dalam konsep pewarisan, kelas turunan (**Anak**) dapat mengakses atribut dan metode yang diwarisi dari kelas induk (**Bapak**), tetapi kelas induk (**Bapak**) tidak dapat mengakses atribut dan metode yang ditambahkan oleh kelas turunannya (**Anak**).
- Dalam metode **main()**, objek **objectBapak** dan **objectAnak** dapat dipanggil dan digunakan secara terpisah, menunjukkan fleksibilitas dan modularitas yang diberikan oleh pewarisan dalam OOP.

```java
//PERCOBAAN 7
class Bapak {
    int a;
    int b;

    void show_variabel() {
        System.out.println("Nilai a = " + a );
        System.out.println("Nilai b = " + b );
    }
}

class Anak extends Bapak {
    int c;
    void show_variabel() {
        System.out.println("Nilai a = " + a);
        System.out.println("Nilai b = " + b);
        System.out.println("Nilai c = " + c);
    }
}

public class InheritExample {
    public static void main(String[] args) {
        Bapak objectBapak = new Bapak();
        Anak objectAnak = new Anak();

        objectBapak.a = 1;
        objectBapak.b = 1;
        System.out.println("Object Bapak (Superclass):");

        objectBapak.show_variabel();
        objectAnak.c=5;
        System.out.println("Object Anak (Superclass dari Bapak):");
        objectAnak.show_variabel();

    }
}
```

Output:

```
Object Bapak (Superclass):
Nilai a = 1
Nilai b = 1
Object Anak (Superclass dari Bapak):
Nilai a = 0
Nilai b = 0
Nilai c = 5
```

PERCOBAAN 8

```
public class Parent {
    String parentName;
    Parent(){}

    Parent(String parentName){
        this.parentName = parentName;
        System.out.println("Konstruktor parent");
    }

}

class Baby extends Parent {
    String babyName;

    Baby(String babyName){
        super();
        this.babyName = babyName;
        System.out.println("Konstruktor Baby");
        System.out.println(babyName);
    }

    public void Cry(){
        System.out.println("Owek owek");
    }
```

- Dengan menggunakan pewarisan, kelas **Baby** dapat mewarisi atribut dan metode yang ada di kelas **Parent**.
- Penggunaan constructor kelas induk dalam kelas turunan memungkinkan untuk menginisialisasi state yang relevan dari kelas induk sebelum melakukan inisialisasi tambahan yang spesifik untuk kelas turunan.
- Dalam hal ini, kelas turunan **Baby** memiliki perilaku tambahan dengan memiliki metode **Cry()**, yang tidak dimiliki oleh kelas induk.
- Kode ini menggambarkan konsep pewarisan dalam pemrograman berorientasi objek (OOP), yang memungkinkan pengorganisasian dan abstraksi yang lebih baik dari kode.

```
// PERCOBAAN 8
class Parent {
    String parentName;
    Parent () {

    }
    Parent (String parentName) {
        this.parentName =parentName;
        System.out.println("Konstruktor parent");
        System.out.println(parentName);
    }
```

```
}

class Baby extends Parent {
    String babyName;

    Baby(String babyName) {
        super();
        this.babyName = babyName;
        System.out.println("Konstruktor Baby");
        System.out.println(babyName);
    }

    public void Cry(){
        System.out.println("Owek owek");
    }
}

public class Test {
    public static void main(String[] args) {
        Parent parent = new Parent("Jordan");
        Baby baby = new Baby("Steve");
        baby.Cry();
    }
}
```

Output :

```
Konstruktor parent
Jordan
Konstruktor Baby
Steve
Owek owek
```