

Nama : LASMARIA KATRINA ANGELINA NAINGGOLAN

NIM : 201402110

KOM : B

Matkul: PEMROGRAMAN BERIORENTASI OBJEK

## 1. PENGERTIAN INHERITANCE

Pewarisan (Inheritance) adalah penurunan sifat yang ada pada suatu kelas kepada kelas baru yang menjadi turunannya. Sifat-sifat suatu kelas ditentukan oleh data anggota dan metode, sehingga yang dimaksud penurunan sifat adalah penurunan sifat adalah penurunan data anggota atau metode. Kelas yang menurunkan sifat disebut kelas dasar disebut kelas turunan (derived class).

Dengan pewarisan dapat diciptakan suatu kelas baru yang mana kelas tersebut mewarisi seluruh sifat kelas dasar kelas yang mempunyai akses public atau protected ditambah sifat khusus dari kelas yang bersangkutan. Ketentuan utama dengan adanya pewarisan yaitu memungkinkan suatu kode yang telah ditulis mudah sekali untuk digunakan kembali. Kita telah membuat suatu kelas yang telah diuji. dan suatu hari nanti, kita ingin menerapkan kode tersebut pada sesuatu yang memiliki sifat-sifat kelas tersebut, tinggal mewariskan kelas yang telah ada di kelas baru (kelas turunan). Kita tidak perlu mengotak ngatik kelas yang teruji, sehingga efek samping yang tidak diharapkan tidak terjadi. Dengan cara ini pengembangan program menjadi lebih efisien dan hemat waktu. Selain itu, kita dapat menambahkan sifat-sifat baru yang tidak ada pada kelas dasar atau bahkan dapat mengganti sifat-sifat pada kelas turunan, yang berbeda dengan sifat kelas dasar.

Pengertian Inheritance:

- Inheritance merupakan salah satu dari tiga konsep dasar OOP.
- Konsep inheritance ini mengadopsi dunia riil dimana suatu entitas/obyek / y dapat mempunyai entitas/obyek turunan.
- Dengan konsep inheritance, sebuah class dapat mempunyai class turunan.
- Suatu class yang mempunyai class turunan dinamakan parent class atau base class.
- Sedangkan class turunan itu sendiri seringkali disebut subclass atau child class.

Deklarasi Inheritance

- Dengan menambahkan kata kunci extends setelah deklarasi nama class, kemudian diikuti dengan nama parent class-nya..

## 2. IMPLEMENTASI INHERITANCE

### A. Implementasi inheritance pada C++:

```
#include <iostream>

using namespace std;

// class induk
class BangunDatar {
protected:
    int lebar, tinggi;

public:
    void setValue(int a, int b){
        lebar = a;
        tinggi = b;
    }
};

// class turunan
class Persegi: public BangunDatar {
public:
    int getValue(){
        return (lebar * tinggi);
    }
};

// class turunan
class Segitiga: public BangunDatar {
public:
    int getValue(){
        return (lebar * tinggi / 2);
    }
};

// program utama
int main () {

    // Inisiasi Object
    Persegi persegi;
    Segitiga segitiga;

    // Inisiasi Nilai
    persegi.setValue(4,5);
    segitiga.setValue(4,5);

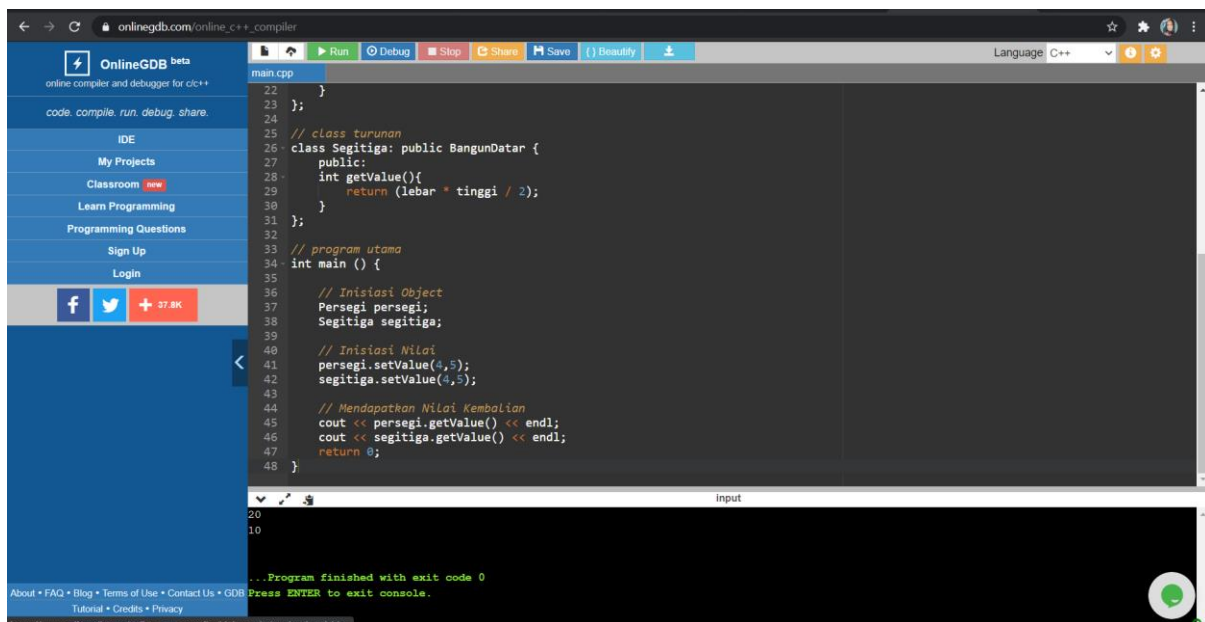
    // Mendapatkan Nilai Kembalian
    cout << persegi.getValue() << endl;
```

```
cout << segitiga.getValue() << endl;
return 0; }
```

Output:

20

10



The screenshot shows the OnlineGDB website interface. On the left is a sidebar with navigation links like 'My Projects', 'Classroom', and 'Sign Up'. The main area displays a C++ code editor with the following code:

```
22 }
23 };
24
25 // class turunan
26 class Segitiga: public BangunDatar {
27 public:
28     int getValue(){
29         return (lebar * tinggi / 2);
30     }
31 };
32
33 // program utama
34 int main () {
35
36     // Inisiasi Object
37     Persegi persegi;
38     Segitiga segitiga;
39
40     // Inisiasi Nilai
41     persegi.setValue(4,5);
42     segitiga.setValue(4,5);
43
44     // Mendapatkan Nilai Kembali
45     cout << persegi.getValue() << endl;
46     cout << segitiga.getValue() << endl;
47     return 0;
48 }
```

Below the code editor, the 'Input' field is empty, and the 'Output' field shows the results of the program execution:

```
20
10
...Program finished with exit code 0
Press ENTER to exit console.
```

## B. Implementasi Inheritance dalam java:

```
class mahasiswa {
    private int npm;
    private String nama;
    private buku bukunya[];
    private int jumlah;
    public mahasiswa(int n, String m){
        npm = n;
        nama = m;
        jumlah = 0;
        bukunya = new buku [3];
        System.out.println("Constructor mahasiswa "+nama);
    }
    public void pinjam_buku(buku b){
        if (jumlah < 3){
            bukunya[jumlah] = b;
            jumlah++;
            System.out.println(nama+" pinjam buku "+b.get_judul()+" karangan "+b.get_pengarang());
        }
    }
}
```

```

        else
            System.out.println("Sudah Penuh!");
    }
    public void daftar_buku(){
        System.out.println("Daftar Bukunya "+nama+":");
        for(int i=0; i<jumlah; i++){
            System.out.println(i+1+" "+bukunya[i].get_judul());
        }
    }
}
class buku {
    public String judul;
    public String pengarang;
    public String penerbit;
    public buku(){ };
    public buku(String j, String p){
        judul = j;
        pengarang = p;
        System.out.println("Constructor buku "+judul);
    }
    public void milik_buku(){
        System.out.println("Method ini milik class induk");
    }
    public String get_judul(){
        return judul;
    }
    public String get_pengarang(){
        return pengarang;
    }
}
class buku_sejarah extends buku {
    public void view_buku_sejarah(){
        milik_buku();
    }
    public void set_buku(String j, String p){
        judul = j;
        pengarang = p;
        System.out.println(judul+" "+pengarang);
    }
}
public class Main {
    public static void main(String[] args) {
        mahasiswa si = new mahasiswa(123,"Ruby");
        buku bp = new buku("Java","Arie");
        si.pinjam_buku(bp);
        buku pti = new buku("Pengantar IT","Rudi");
    }
}

```

```
        si.pinjam_buku(pti);
        buku logika = new buku("Logika","Pulung");
        si.pinjam_buku(logika);
        buku komik = new buku("Naruto","One Piece");
        si.pinjam_buku(komik);
        si.daftar_buku();
        buku_sejarah bukuku = new buku_sejarah();
        bukuku.view_buku_sejarah();
        bukuku.set_buku("Sejarah Kita","Top Cerr");
    }
}
```

#### OUTPUT:

init:

deps-jar:

Compiling 1 source file to C:\Documents and Settings\user\My Documents\NetBeansProjects\JavaApplication21\build\classes

compile:

run:

Constructor mahasiswa Ruby

Constructor buku Java

Ruby pinjam buku Java karangan Arie

Constructor buku Pengantar IT

Ruby pinjam buku Pengantar IT karangan Rudi

Constructor buku Logika

Ruby pinjam buku Logika karangan Pulung

Constructor buku Naruto

Sudah Penuh!

Daftar Bukunya Ruby:

1. Java

2. Pengantar IT

3. Logika

Method ini milik class induk

Sejarah Kita Top Cerr

BUILD SUCCESSFUL (total time: 8 seconds)

### 3. ACCESS SPECIFIER

*Access Modifier / Access Specifier* adalah salah satu fitur penting dalam *Object Oriented Programming* (OOP) untuk melakukan *Data Hiding* (Menyembunyikan Data). Fitur ini memungkinkan kita untuk mengatur hak akses dari *member class*, digunakan agar tidak sembarangan perintah dapat mengakses, atau tidak bisa di akses secara langsung.

Fitur ini memiliki 3 tipe *Access Modifier*, yaitu:

1. *Public*
2. *Private*
3. *Protected*

Ketiga *Access Modifier* mempunyai sifat mereka masing-masing . Semua baris deklarasi sebagai *member class* akan mengikuti sifat dari label di atasnya hingga label berikutnya atau tanda penutup *class* `};`.

#### 1. PUBLIC

*Public* adalah label yang berfungsi untuk menentukan sifat akses ke semua *member* yang mengikutinya (di bawahnya), sehingga memiliki sifat dapat di akses dari manapun. Dapat di akses dari dalam *class* itu sendiri, dari anak *class* (*derived class*) dan juga dari luar *class*.

Contoh program:

```
include <iostream>
using namespace std;

class siswa{
public: //Akses Specifier
    int id;
};

int main (){
    siswa budi;
    budi.id = 10;
    cout<<budi.id<<endl;

    return 0;
}
```

Di atas adalah contoh program untuk membuktikan sifat dari *public*. *Member* yang mengikuti label *public* adalah `id`. dan di *main function* kita membuat *object* `budi`, untuk mencoba mengakses *member* variabel `id` dari *instance* `budi`. Dan membuktikan bahwa label *public* akan bersifat dapat diakses dari manapun, dari dalam *class*, anak *class* dan diluar *class*.

## 2. PRIVATE

*Private* adalah label yang berfungsi untuk menentukan sifat akses ke semua *member* yang mengikutinya menjadi memiliki sifat yang tidak dapat di akses dari manapun kecuali melalui *friend function* dan dari dalam *class* itu sendiri.

Contoh program:

```
#include <iostream>
using namespace std;

class siswa{
private:
    int id;
};

class orangtua : public siswa{
public:
    int getId(){
        return id;
    }
};

int main (){
    siswa budi;
    budi.id = 10;
    cout<<budi.id<<endl;
    return 0;
}
```

Di atas adalah program yang sama seperti contoh program pada penjelasan *public*, penulis hanya menambahkan anak *class* (*derived class*), dan mengganti label *public* dengan *private*.

Dari program di atas akan menghasilkan pesan *Error* pada *compiler*, yang berbunyi:

**Error : 'int siswa::id' is private**

Pesan tersebut berasal dari 3 tempat yaitu pesan pertama dari perintah yang berada di dalam *derived class*, kedua dan ketiga berasal dari perintah pemanggilan *member id* yang berada pada *main function*.

Pesan itu membuktikan bahwa *member id* mengikuti sifat label *private*. Yang tidak dapat diakses dari luar *class* dan tidak bisa diakses dari *derived class*. *Private* memiliki sifat hanya dapat di akses dari dalam *class* itu sendiri.

Tapi masih ada cara untuk mengakses *member id* tersebut dari luar *class*, dengan cara membuat perantara di dalam *class* tersebut, yaitu dengan *function* yang bersifat *public*, cara ini disebut *setter* dan *getter*.

Contoh program:

```
#include <iostream>
using namespace std;

class siswa{
private:
    int id;

public:
    int getId(){
        return id;
    }

    void setId(int i){
        id = i;
    }
};

int main (){
    siswa budi;
    budi.setId(10);
    cout<<budi.getId()<<endl;
    return 0;
}
```

Bukan hanya dengan cara itu saja, kita juga bisa menggunakan *friend function* untuk dapat mengakses *member class* yang bersifat *private*, yang akan penulis jelaskan pada artikel berikutnya.



### C. PROTECTED

*Protected* adalah label yang berfungsi untuk menentukan sifat akses semua *member* yang mengikutinya, sehingga memiliki sifat yang tidak dapat diakses dari luar *class* tapi masih dapat di akses dari dalam *class* maupun anak *class* (*derived class*).

Contoh program:

```
#include <iostream>
using namespace std;

class siswa{
protected:
    int id;
};

class orangtua : public siswa{
public:
    int getId(){
        return id;
    }
};

int main (){
    siswa budi;
    budi.id = 10;
    cout<<budi.id<<endl;
    return 0;
}
```

Di atas adalah contoh program yang masih sama seperti contoh program pada contoh sebelumnya dari penjelasan *private*. Penulis hanya mengubah label *access specifier* dari *private* menjadi *protected*.

Dari program di atas akan menghasilkan pesan *Error* pada *compiler*, yang berbunyi.

**Error : 'int siswa::id' is protected**

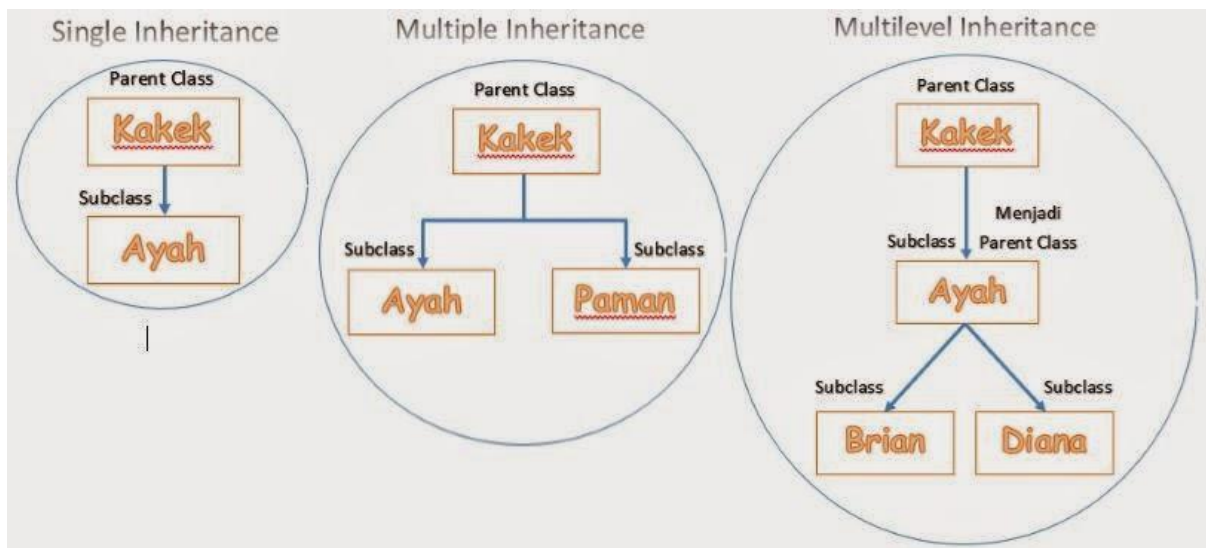
Pesan tersebut hanya berasal dari perintah pemanggilan *member id* yang berada pada *main function*, membuktikan sifat akses dari *protected*. *Member id* tidak bisa diakses dari luar *class* tapi masih bisa diakses dari *derived class*.

Sama seperti sebelumnya, untuk dapat mengakses *member* yang bersifat *protected* dari luar *class* kita juga bisa menggunakan cara *setter getter* atau *friend function*.

#### 4. JENIS JENIS INHERITANCE:

Ada 5 JENIS *inheritance*, yaitu :

1. *Single Level Inheritance* : yaitu inheritance yang hanya satu level, dari satu *Superclass* ke satu *Subclass*
2. *Multiple Inheritance* : yaitu class yang memiliki 2 *Superclass*.
3. *Multilevel Inheritance* : seperti gambar diatas.
4. *Hierarchical Inheritance* : yaitu saat ada satu class yang dijadikan *Superclass* dan class lainnya/lebihnya adalah *Subclass*.
5. *Hybrid Inheritance* : yaitu kombinasi dari 2 atau lebih type inheritance diatas.



**RESOURCE:**

<http://pborubyjava.blogspot.com/2012/04/inheritance.html>

<https://kodedasar.com/inheritance-cpp/>

<https://www.belajarcpp.com/tutorial/cpp/access-modifier/>

<http://sudahkahtahu.blogspot.com/2015/02/konsep-oop-inheritance.html>