

PEMROGRAMAN BERORIENTASI OBJEK

Inheritance dan Polimorfisme

PENDAHULUAN

- Apa itu *Inheritance*, *Derived Class* , dan *Base class* ? Pasti untuk seorang yang baru mengenal bahasa *pemrograman berorientasi objek* akan sangat membingungkan.
- *Inheritance* adalah sebuah keistimewaan bahasa yang berorientasi objek (PBO) yang tidak hanya C++ tetapi juga bahasa yang lain

KOMPONEN INHERITANCE

- **Inheritance** = Pewarisan
- **Derived class** = Kelas Turunan
- **Base Class** = Kelas Dasar

Contoh :

```
class Segitiga : public BangunDatar{ };
```

Artinya :

- **Class Segitiga** adalah **derived class** (kelas yang diturunkan).
- **Class BangunDatar** adalah **base class** (kelas dasar).
- Semua yang dimiliki oleh **kelas dasar** akan dimiliki juga oleh **kelas turunannya**

KOMPONEN INHERITANCE

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

class BangunDatar {
};

class Segitiga : public BangunDatar{
}

int main () {
    Segitiga segitigasiku;
    return 0;
}
```

INHERITANCE

- Di dalam *inheritance* tentu akan muncul pertanyaan mengenai, siapa yang mewariskan? siapa yang di warisi ? dan warisannya itu apa ?
- Sama seperti manusia, yang *mewariskan* adalah orang tua, yang diwarisi adalah anak, *warisannya* itu bisa berupa *sifat dan tampak fisik* dari kedua orang tua

BASE CLASS DAN DERIVATED CLASS

- Dalam **PBO** yang mewariskan disebut sebagai **Base Class**, yang diwarisi disebut sebagai **Derived Class**, dan **warisannya** itu berupa '**members**' dari **class** yaitu **variable** dan **function**.
- Untuk lebih mudah di pahami, ditunjukkan beberapa baris kode, dengan nama '**Segitiga**' dan '**Kotak**'
- '**Segitiga**' dan '**Kotak**' ini sama-sama bangun '**datar**' yang memiliki perbedaan bentuk tetapi sama-sama punya '**lebar**' dan '**tinggi**'.

BASE CLASS DAN DERIVATED CLASS

- Pertama-tama dibuat ***kelas pertama*** yang akan digunakan sebagai ***base class*** dan ***kelas*** ini kita berikan nama ***BangunDatar***, dibuat ***member class*** yang berupa ***dua variables*** dan ***satu fungsi***.
- ***Variable – variable*** tersebut adalah ***lebar*** dan ***tinggi*** dengan ***type data*** adalah ***integer (int)*** dengan '***access specifier***' ***protected***.
- Sedangkan untuk fungsi nya adalah ***set_values()*** yang diberikan ***void*** dan '***access specifier***' nya ***public***.

BASE CLASS DAN DERIVATED CLASS

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

class BangunDatar {

protected :
    int lebar, tinggi;
}

Public :
    void set_values (int a, int b){
        lebar = a;
        tinggi = b;
    }
};
```

BASE CLASS DAN DERIVATED CLASS

Sampai di sini pasti ada beberapa pertanyaan, diantaranya adalah

- Kenapa memakai **protected** ?
- Kenapa **int** ?
- Kenapa **public** ?
- Kenapa **void** ?

BASE CLASS DAN DERIVATED CLASS

- **Protected** *access specifier* akan *mem-protect/ melindungi member/variable* dari *pengakses-san* di luar *kelas BangunDatar*.
- Bila memiliki *kelas* lain yang tidak ada hubungannya dengan *kelas BangunDatar* maka tidak akan bisa mengakses *variable* ini terlebih mengubah nilainya.
- Tetapi jika menjadikan *kelas* lain sebagai '**Derived Class**' / '**Kelas Turunan**', maka *Kelas turunan* tersebut akan memiliki hak akses ke *variable* dari *Base Class* ini.

BASE CLASS DAN DERIVATED CLASS

- Kenapa menggunakan **type data int** ? karena nilai diinginkan masuk ke dalam **variable lebar** dan **tinggi** adalah **angka (interger)**
- Kenapa **public** ? Tujuannya supaya **function /method** '**set_values()**' ini dapat di akses oleh **Kelas** apa saja
- Lalu kenapa **void** ? Karena **Function /Method** '**set_values()**' tidak **me-return** apa pun, **fungsi** ini hanya untuk **mengeset nilai** dari **lebar** dan **tinggi**, sehingga cukup memakai **void** saja

BASE CLASS DAN DERIVATED CLASS

- Selanjutnya dibuat *kelas baru* “**Kotak**” yang merupakan ‘**Derived Class**’ dari *kelas BangunDatar*.
- **Tanda colon (:) / titik dua** merupakan tanda *inheritance* dan *kelas* yang dibuat ini hanya memiliki **member** berupa *fungsi luas()* saja.
- Tetapi dari *fungsi luas* ini, kita bisa memanggil **variable lebar** dan **tinggi** miliknya *kelas BangunDatar* dan inilah gunanya *inheritance*
- **Kelas “Kotak”** ini nantinya akan memiliki semua **member** dari *kelas BangunDatar* termasuk *fungsi set_values* juga.

BASE CLASS DAN DERIVATED CLASS

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

```
class BangunDatar {
protected :
    int lebar, tinggi;
}
```

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

```
class Kotak : public BangunDatar{
public :
    int luas () {
        return (lebar * tinggi);
    }
};
```

BASE CLASS DAN DERIVATED CLASS

- Selanjutnya dibuat *kelas “Segitiga”* dengan menggunakan *rumus luas segitiga* yaitu *rumus luas segitiga* sama dengan *rumus luas kotak dibagi dua*.
- Karena *kelas segitiga* merupakan *kelas turunan* dari *kelas BangunDatar*, jadi tidak perlu membuat / mendefinisikan *variable lebar* dan *tinggi* yang baru, cukup menggunakan *variable* yang sudah ada pada *kelas BangunDatar*.

BASE CLASS DAN DERIVATED CLASS

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

class BangunDatar {
protected :
    int lebar, tinggi;
}

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

class Kotak : public BangunDatar{
public :
    int luas () {
        return (lebar * tinggi);
    }
};
```

```
class SegiTiga : public BangunDatar
public :
    int luas(){
        return (lebar * tinggi / 2);
    }
};
```

BASE CLASS DAN DERIVATED CLASS

- Selanjutnya kita dibuat sebuah **objek baru** (bukan variable atau function) yaitu '**kotakku**' dan '**segitigaku**' yang dibuat dari **kelas** nya masing masing.
- Dari **objek** tadi, dipanggil fungsi dari **base class BangunDatar** yaitu **fungsi set_values** dan semua **member** dari **kelas BangunDatar** (**variable lebar, tinggi, fungsi set_values**) akan diturunkan ke **derived class** nya
- Setelah di-**set** nilai untuk **panjang** dan **lebar** dari masing masing **objek**, maka kemudian dipanggil **fungsi luas** dari masing masing **objek** tersebut.

MAIN FUNCTION & OBJECT

```
int main () {  
    Kotak kotakku;  
    Segitiga segitigaku;  
  
    kotakku.set_values(3,4);  
    segitigaku.set_values(3,4);  
  
    cout << kotakku.luas() << end;  
    cout << segitiga.luas() << end;  
  
    Return 0;  
}
```

BASE CLASS, DERIVATED CLASS, & OBJECT

```
#include <iostream>
using namespace std;
// Membuat Kelas Bangun Datar
class BangunDatar{
protected:
    int lebar, tinggi;
public:
    void set_values(int a, int b){
        lebar = a;
        tinggi = b;
    }
};

// Membuat Kelas Turunan Bangun Datar yaitu Kotak
class Kotak: public BangunDatar {
public:
    int luas(){
        return (lebar * tinggi);
    }
};
```

BASE CLASS, DERIVATED CLASS, & OBJECT

```
// Membuat Kelas Turunan Bangun Datar yaitu Segitiga
class Segitiga: public BangunDatar {
    public:
        int luas(){
            return (lebar * tinggi / 2);
        }
};

int main(){
    Kotak kotakku;
    Segitiga segitigaku;

    kotakku.set_values(3,4);
    segitigaku.set_values(3,4);

    cout << "Luas Kotak   : " << kotakku.luas() << endl;
    cout << "Luas Segitiga   : " << segitigaku.luas() << endl;

    return 0;
}
```

POLIMORFISME

- ***Polimorfisme*** adalah suatu pemrograman yang memiliki banyak bentuk dalam arti gaya penulisan, bentuk penulisan atau ragam penulisan.
- ***Polimorfisme*** meliputi :
 - 1. Pointer to Base Class***
 - 2. Virtual Member***
 - 3. Abstract Class***

POINTER TO BASE CLASS

```
// Pointer to Base Class
#include <iostream>
using namespace std;

// Membuat Kelas CPolygon
class CPolygon {
    protected :
        int panjang, lebar;
    public :
        void set_values (int a, int b){
            panjang = a;
            lebar = b;
        }
};

// Membuat Kelas Turunan CRectangle dari Base Class CPolygon
class CRectangle : public CPolygon {
    public :
        int luas(){
            return (panjang * lebar);
        }
};
```

POINTER TO BASE CLASS

```
// Membuat Kelas Turunan CTriangle dari Base Class CPolygon
class CTriangle : public CPolygon {
    public :
    int luas(){
        return (panjang * lebar / 2);
    }
};

int main() {
    CRectangle Kotak;
    CTriangle SegiTiga;

    CPolygon *poly1 = &Kotak;
    CPolygon *poly2 = &SegiTiga;

    poly1-> set_values (5,8);
    poly2-> set_values (5,8);

    cout << "Luas Kotak    : " << Kotak.luas () << endl;
    cout << "Luas Segi Tiga  : " << SegiTiga.luas () << endl;

    return 0;
}
```

FUNGSI VIRTUAL

- ***Fungsi Virtual*** adalah ***fungsi*** yang mendukung adanya ***polymorphic function***, artinya ***fungsi*** tersebut dapat didefinisikan ulang pada ***kelas-kelas turunannya***.
- ***Fungsi virtual*** ini biasanya terdapat pada ***kelas-kelas dasar***, tetapi bisa juga dideklarasikan pada ***kelas-kelas turunan*** yang akan dijadikan sebagai ***kelas dasar*** bagi ***kelas-kelas lainnya***.

FUNGSI VIRTUAL

- Dalam C++, untuk mendefinisikan ***fungsi*** sebagai ***fungsi virtual*** adalah dengan menggunakan ***keyword virtual***, yaitu dengan menempatkan didepan pendeklarasian ***fungsi*** tersebut.
- Pendefinsian ***fungsi virtual*** yang terdapat pada ***kelas dasar*** biasanya tidak begitu berarti artinya ***kode-kode*** yang terdapat didalamnya masih bersifat umum.

FUNGSI VIRTUAL

```
// Pointer to Base Class
#include <iostream>
using namespace std;

// Membuat Kelas CPolygon
class CPolygon {
    protected :
    int panjang, lebar;
    public :
    void set_values (int a, int b){
        panjang = a;
        lebar = b;
    }
    // Membuat Virtual Member
    virtual int luas(){
        return 0;
    }
};

// Membuat Kelas Turunan CRectangle dari Base Class CPolygon
class CRectangle : public CPolygon {
    public :
    int luas(){
        return (panjang * lebar);
    }
};
```

FUNGSI VIRTUAL

```
// Membuat Kelas Turunan CTriangle dari Base Class CPolygon
class CTriangle : public CPolygon {
    public :
    int luas(){
        return (panjang * lebar / 2);
    }
};

int main() {
    CRectangle Kotak;
    CTriangle SegiTiga;

    CPolygon *poly1 = &Kotak;
    CPolygon *poly2 = &SegiTiga;

    poly1-> set_values (5,8);
    poly2-> set_values (5,8);

    cout << "Luas Kotak          : " << poly1->luas () << endl;
    cout << "Luas Segi Tiga       : " << poly2->luas () << endl;
    return 0;
}
```

ABSTRACT BASE CLASS

- **Kelas *abstrak*** adalah sebuah **kelas** yang mempunyai ***fungsi virtual murni*** didalamnya, baik satu ataupun lebih.
- Maka dari itu kita tidak diizinkan untuk melakukan instansiasi terhadap ***kelas abstrak***.

ABSTRACT BASE CLASS

```
// Abstract Base Class
#include <iostream>
using namespace std;

// Membuat Kelas CPolygon
class CPolygon {
protected :
    int panjang, lebar;
public :
    void set_values (int a, int b){
        panjang = a;
        lebar = b;
    }

    // Membuat Pure Virtual Member
    virtual int luas()= 0;};

// Membuat Kelas Turunan CRectangle dari Base Class CPolygon
class CRectangle : public CPolygon {
public :
    int luas(){
        return (panjang * lebar);
    }
};
```

ABSTRACT BASE CLASS

```
// Membuat Kelas Turunan CTriangle dari Base Class CPolygon
class CTriangle : public CPolygon {
    public :
    int luas(){
        return (panjang * lebar / 2);
    }
};

int main() {
    CRectangle Kotak;
    CTriangle SegiTiga;

    CPolygon *poly1 = &Kotak;
    CPolygon *poly2 = &SegiTiga;

    poly1-> set_values (5,8);
    poly2-> set_values (5,8);

    cout << "Luas Kotak          : " << poly1->luas () << endl;
    cout << "Luas Segi Tiga     : " << poly2->luas () << endl;
    return 0;
}
```