Chapter 7 Inheritance

Pewarisan (Inheritance)

Pemrograman Berorientasi Objek mempunyai fitur penting yang memudahkan pemrogram dalam membuat program yaitu pewarisan (*inheritance*). Aspek penting pewarisan dalam pemrograman berorientasi objek adalah pemakaian kode program yang sudah ada (*code reuse*), yang akan dibahas pada bab ini dan polimorfisme (*polymorhism*) yang akan dibahas pada bab berikutnya.

Kelas yang sudah ada dapat digunakan lagi untuk dikembangkan menjadi kelas yang baru, dalam hal ini kelas yang sudah ada dinamakan **kelas dasar** (*base class*) sedang kelas baru yang akan dibuat dinamakan **kelas turunan** (*derived class*). Dengan demikian, semua anggota **kelas dasar** yang tidak bersifat privat akan diwarisi oleh **kelas turunan**nya dan pemrogram tinggal menambahkan anggota-anggota baru untuk menambahkan fungsionalitas kelas tersebut.

Penulisan Penurunan

Untuk membuat kelas turunan dapat dilakukan dengan cara sama seperti mendeklarasikan kelas biasa dengan menambahkan titik dua (:) setelah nama kelas dan diikuti dengan jenis penurunan (public dsb.) dan nama kelas dasar yang akan diturunkan, bentuk umum adalah seperti berikut:

```
class <kelas_turunan> : <jenis_penurunan> <kelas_dasar>
```

Sebagai contoh misalnya akan dibuat kelas turunan **Silinder** dari kelas dasar **Lingkaran**, maka dapat dituliskan :

```
class Silinder : public Lingkaran
```

Mengenai jenis akses **public** ini akan dibahas nanti, sekarang kita akan memakai **public**. Kelas dasar yang akan diturunkan harus sudah dideklarasikan lebih dahulu, jika tidak maka kita akan menjumpai pesan kesalahan kompiler.

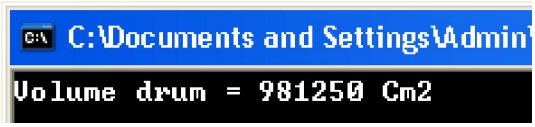
Labs.1 Contoh Pewarisan (*Inheritance*).

1. Buka Qt Creator dan buat project Qt Console Application baru dengan nama **Labs1**, kemudian tulis kode berikut.

```
#include <QtCore/QCoreApplication>
#include <iostream>
using namespace std;
class Lingkaran{
public:
    //Konstruktor
    Lingkaran(float radius = 0){
        Lingkaran::radius = radius;
    }
    //Destruktor
```

```
    ~Lingkaran(){}
protected :
    float radius;
public :
    float getLuas(){
        return 3.14 * radius * radius;
    }
};

class Silinder : public Lingkaran{
public:
    //Konstruktor
    Silinder(float radius, float tinggi){
        Silinder::radius = radius; //<-- warisan
        Silinder::tinggi = tinggi; //<-- anggota baru
    }
    //Destruktor
    ~Silinder(){}
private:
    float tinggi; //<-- anggota baru
public:
    float getVolume(){ //<-- anggota baru
        //getLuas() adalah warisan
        return getLuas() * tinggi;
    }
};

int main(int argc, char *argv[])
{
    QCoreApplication a(argc, argv);
    Silinder drum(50,125);
    cout << "Volume drum = " << drum.getVolume() << " Cm2" << endl;
    return a.exec();
}
```

2. Kemudian jalankan kode di atas dengan menekan tombol Ctrl+R, outputnya adalah sebagai berikut.



Analisa Program :

- Pada program diatas sudah dibuat kelas Lingkaran, dengan variabel anggota **radius** bertipe **float** dan fungsi anggota **getLuas()** yang mengemalikan nilai **float**.
- Kelas kedua adalah Silinder yang merupakan turunan dari kelas Laingkaran, oleh karena itu deklarasi dituliskan : `class Silinder : Lingkaran`, dengan demikian kelas **Silinder** akan mewarisi anggota kelas **Lingkaran** yang tidak **private**, yaitu : variabel anggota **radius** bertipe **float** dan fungsi anggota **getLuas()** yang mengemalikan nilai **float**.

- Tampak pada kelas **Silinder** ditambahkan variabel anggota **tinggi** bertipe **float** dan fungsi anggota **getVolume()** yang mengembalikan nilai **float**. Ini memberikan contoh penambahan fungsionalitas dari kelas yang sudah ada.
- Pada fungsi anggota **getVolume()** pada kelas Silinder, nilai kembalian dirumuskan `return getLuas() * tinggi`, ini menjelaskan bahwa fungsi `getLuas()` tersebut sekarang juga menjadi milik kelas **Silinder** (mendapat warisan).
- Pada hasil eksekusi, tampak bahwa fungsi **getVolume()** menghitung (3.14 * 50 * 50)* 125 dan menghasilkan nilai **981250**. Ini berarti perhitungan **getLuas()** memakai fungsi anggota milik kelas **Lingkaran** yang diwariskan kepada kelas **Silinder**.

Jenis Akses Penurunan Kelas

Deklarasi kelas **Silinder** di atas adalah : `class Silinder : public Lingkaran`, ini berarti semua anggota yang bersifat `public` dan `protected` dari kelas **Lingkaran** akan diwariskan kepada kelas `Silinder` dan pada kelas `Silinder` anggota-anggota warisan tersebut akan tetap mempunyai jenis akses seperti itu. Namun jika modifier akses `public` dihilangkan maka berarti pewarisan memakai jenis akses `private`, sebab secara default C++ memakai jenis akses `private` jika modifer akses tidak dituliskan. Jika ini terjadi, maka akan terjadi perubahan modifier akses terhadap anggota-anggota warisan tersebut di dalam kelas `Silinder`, yaitu semua anggota yang diwariskan (baik berjenis `public` maupun `protected`) akan berubah menjadi `private` di dalam kelas `Silinder`.

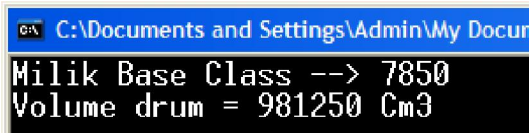
Labs.2 Jenis Akses Public Pada Penurunan

1. Buka project **Labs1** di atas, kemudian tambahkan (edit) kode berikut pada fungsi **main()** :

   `cout << "Milik Base Class --> " << drum.getLuas() << endl;`

```
int main(int argc, char *argv[])
{
    QCoreApplication a(argc, argv);
    Silinder drum(50,125);
    cout << "Milik Base Class --> " << drum.getLuas() << endl;
    cout << "Volume drum = " << drum.getVolume() << " Cm2" << endl;
    return a.exec();
}
```

2. Kemudian jalankan kode di atas dengan menekan tombol Ctrl+R, outputnya adalah sebagai berikut.



Analisa Program :

- Tampak pada program dapat mengakses metode warisan kelas Lingkaran dari dalam program utama (**main()**). Ini menunjukkan bahwa metode tersebut diwariskan ke kelas Silinder dan jenis aksesnya masih tetap sama yaitu public.

3. Kemudian hapuslah jenis akses penurunan `public` (atau gantilah dengan `private`) pada deklarasi kelas **Silinder** yang tadinya:

```
class Silinder : public Lingkaran
```

Sehingga menjadi:

```
class Silinder : Lingkaran
```

Kemudian jalankan kode diatas dengan menekan tombol Ctrl+R, maka tidak akan ada output karena terjadi kesalahan kompilasi sebagai berikut.

Analisa Program :

- Tampak pada program metode warisan kelas **Lingkaran** bernama **getLuas()** tidak dapat diakses dari dalam program utama (**main()**). Hal ini disebabkan karena metode tersebut ketika diwariskan ke kelas **Silinder** jenis aksesnya berubah menjadi **private**, yang berarti diwariskan akan tetapi hanya dapat diakses dari dalam kelas **Silinder**, akibatnya ketika akan diakses dari program utama (**main()**), terjadi kesalahan kompilasi seperti di atas.

Berikut ini adalah perubahan jenis akses anggota dari kelas dasar ke kelas turunan berdasarkan jenis akses penurunan:

Jenis akses penurunan	Jenis akses pada kelas dasar	Jenis akses baru pada kelas turunan

`private`	`private`
	`protected`
	`public`
		Tidak diwariskan
		`private`
		`private`

`protected`	`private`
	`protected`
	`public`
		Tidak diwariskan
		`protected` (tetap)
		`protected`

`public`	`private`
	`protected`
	`public`
		Tidak diwariskan
		`protected` (tetap)
		`public` (tetap)

TIPS

Pada umumnya jenis akses penurunan adalah **public***, oleh karena itu biasakan menuliskan jenis akses* **public** *ketika akan menurunakan suatu kelas.*

Warisan

Anggota-anggota kelas (*member variable* dan *member function*) mempunyai jenis akses **private**,

protected dan **public**, jika tidak dituliskan pada deklarasi anggota tersebut maka akan digunakan jenis akses **private**. Dalam hal pewarisan (*inheritance*) pada pemrograman berorientasi objek, seperti sudah dijelaskan di atas bahwa anggota yang diwariskan adalah anggota dengan jenis akses **public** atau **protected**. Jenis akses **public** pada suatu anggota artinya bahwa anggota tersebut dapat diakses dari manapun dan akan diwariskan jika kelas tersebut diturunkan, sedangkan jenis akses **protected** berarti anggota tersebut diwariskan kepada kelas turunannya dan hanya bisa diakses dari dalam kelas turunan tersebut. Berikut ini adalah tabel yang menjelaskan jenis akses dan aksesibilitas suatu anggota:

Aksesibilitas	public	protected	private
Dari dalam kelas itu sendiri	**Ya**	**Ya**	**Ya**
Dari kelas beda turunan	**Ya**	**Ya**	**Tidak**
Dari kelas beda tetapi bukan turunan	**Ya**	**Tidak**	**Tidak**

Perlu diketahui bahwa **konstruktor** dan **destruktor** tidak diwariskan. Hal ini bisa dimaklumi, sebab konstruktor bekerja spesifik untuk kelas tersebut. Pada C++ jelas bahwa nama **konstruktor** sama dengan nama kelasnya, karena nama kelas turunan tidak mungkin sama dengan nama kelas dasar, maka tidak mungkin **konstruktor** kelas dasar juga merupakan **konstruktor** kelas turunan.

Namun demikian, pada konteks pewarisan, perlu diketahui bahwa itu tidak berarti **konstruktor** kelas dasar dapat diabaikan, sebab bagaimanapun juga dalam pembentukan objek **konstruktor** suatu kelas pasti bekerja (oleh karena itu diberi nama “konstruktor” yang artinya pembentuk). Berikut ini hal-hal yang perlu diperhatikan pada pewarisan mengenai konstruktor:

Tiap Kelas Mempunyai Konstruktor

Tidak ada kelas yang tidak mempunyai konstruktor. Adalah benar bahwa secara eksplisit kita bisa menuliskan sebuah kelas tanpa mendeklarasikan konstruktor sama sekali, namun itu tidak berarti bahwa kelas tersebut tidak mempunyai kontruktor, sebab sebenarnya yang dieksekusi oleh komputer bukan kode program yang kita tulis tersebut, melainkan hasil kompilasi dari kode program tersebut. Pada waktu dikompilasi, kompiler akan menambahkan konstruktor tanpa parameter yang tidak melakukan apa-apa seperti berikut:

```
<nama_kelas>(){}
```

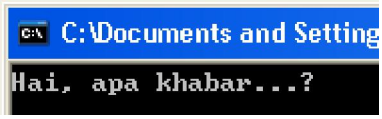
Labs.3 Konstruktor default

1. Buka Qt Creator, buat project Qt Console Application dengan nama **Labs3**. Kemudian tulis kode berikut.

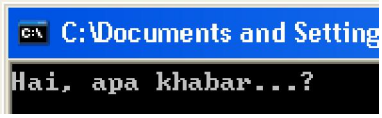
```
#include <QtCore/QCoreApplication>
#include <iostream>
using namespace std;
class Kelasku{
public:
    Kelasku(){} //<-- konstruktor ini boleh tidak ditulis

public:
```

```
    void hai(){
        cout << "Hai, apa khabar...?" << endl;
    }
};

int main(int argc, char *argv[])
{
    QCoreApplication a(argc, argv);
    Kelasku test;
    test.hai();
    return a.exec();
}
```

7. Tekan Ctrl+R untuk menjalankan kode di atas, outputnya adalah sebagai berikut.



8. Kemudian hapuslah konstruktor **`Kelasku(){}`**, kemudan tekan Ctrl+R untuk menjalankan kode di atas, outputnya adalah sebagai berikut.



Analisa Program:

- Pada contoh program ini tampak bahwa ada konstruktor maupun tidak ada konstruktor program di atas tetap bisa dijalankan dan tidak ada perbedaan sama sekali. Hal ini disebabkan oleh karena jika suatu kelas tidak mempunyai kontruktor, maka secara otomatis kompiler akan menambahkan konstruktor default (yaitu konstruktor tanpa parameter dan tanpa program apapun) pada hasil kompilasi, jadi pada contoh program di atas hasil kompilasi dengan atau tanpa konstruktor adalah tetap sama.

Konstruktor Kelas Turunan Pasti Memanggil Konstruktor Kelas Dasar

Seperti dijelaskan di atas bahwa tidak ada kelas yang tidak mempunyai konstruktor, demikian juga dengan kelas turunan. Pada waktu ada pembentukan suatu objek dari suatu kelas turunan, secara otomatis ada terlebih dahulu pembentukan objek kelas dasarnya karena harus ada anggota-anggota yang diwariskan, dengan demikian bisa dimengerti bahwa konstruktor kelas turunan pasti memanggil konstruktor kelas dasarnya.

Sama seperti pada penulisan kelas biasa, pada kelas turunan juga bisa tidak dituliskan konstruktor secara eksplisit dan pada kasus ini pada saat kompilasi kompiler akan menambahkan konstruktor kosong tanpa parameter. Namun bentuk konstruktor kosong pada kelas turunan berbeda dengan konstruktor kosong kelas biasa, karena pada konstruktor kosong kelas turunan akan memanggil konstruktor kosong kelas dasarnya.

Labs.4 Konstruktor default kelas turunan

1. Buka Qt Creator, buka project Qt Console Application dengan nama **Labs3** tadi. Kemudian ubah isi konstruktor kelas **Kelasku** dan tambahkan kelas **Turunan** berikut.

```
#include <QtCore/QCoreApplication>
#include <iostream>
using namespace std;
class Kelasku{
public:
    Kelasku(){
        cout << "Konstruktor Kelas Dasar dijalankan..." << endl;
    }
public:
    void hai(){
        cout << "Hai, apa khabar...?" << endl;
    }
};
class Turunan : public Kelasku{
};

int main(int argc, char *argv[])
{
    QCoreApplication a(argc, argv);
    Turunan test; //<-- membuat objek dari kelas Turunan saja
    return a.exec();
}
```

2. Tekan Ctrl+R untuk menjalankan kode di atas, outputnya adalah sebagai berikut.

Analisa Program:

- Pada contoh program ini tampak bahwa tidak ada objek yang dibuat dari kelas dasar, namun jika dilihat hasil eksekusinya, konstruktor yang dijalankan adalah konstruktor kelas dasar. Ini berarti bahwa ada pemanggilan kontruktor kelas dasar, yaitu pada waktu pembentukan objek kelas Turunan.
- Berdasarkan kenyataan bahwa tiap kelas pasti punya kontruktor, maka ini berarti kelas Turunan juga mempunyai kontruktor namun konstruktor tersebut di dalamnya ada pemanggilan kontruktor kelas dasarnya.
- Dari percobaan ini, bisa disimpulkan bahwa pada saat dikompilasi, karena kelas turunan secara eksplisit tidak dituliskan konstruktor, maka kompiler akan menambahakan konstukor kosong tanpa parameter yang memanggil konstruktor kelas dasarnya seperti berikut :

```
Turunan():Kelasku(){} //<-- Konstruktor default kelas Turunan
```

Karakteristik konstruktor kelas turunan ini penting untuk dipahami, karena kadang-kadang kita lupa

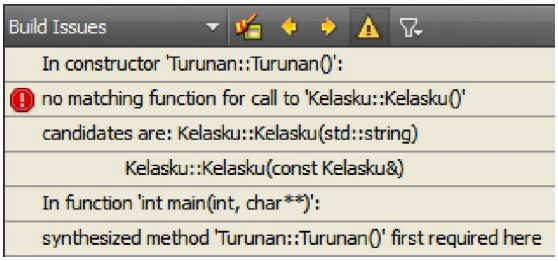
bahwa pada pembuatan kelas turunan pasti di dalamnya ada pemanggilan konstruktor kelas dasarnya.

Labs.5 Konstruktor default kelas turunan memanggil konstruktor kelas dasar

1. Buka Qt Creator, buka project Qt Console Application dengan nama **Labs3** tadi. Kemudian ubah isi konstruktor kelas **Kelasku** seperti berikut.

```
#include <QtCore/QCoreApplication>
#include <iostream>
using namespace std;
class Kelasku{
public:
    Kelasku(string kata){
        cout << "Konstruktor Kelas Dasar" << endl;
        cout << "Mengucapkan : " << kata << endl;
    }
public:
    void hai(){
        cout << "Hai, apa khabar...?" << endl;
    }
};

class Turunan : public Kelasku{
};

int main(int argc, char *argv[])
{
    QCoreApplication a(argc, argv);
    Turunan test; //<-- membuat objek dari kelas Turunan saja
    return a.exec();
}
```

2. Tekan Ctrl+R untuk menjalankan kode di atas, tidak akan ada output karena ada kesalahan dengan pesan kesalahan kompilasi sebagai berikut.



Analisa Program:

- Seperti pada percobaan **Lab.4**, kelas Turunan tidak mempunyai konstruktor secara eksplisit, sehingga dibuatkan konstruktor oleh kompiler berupa :

  ```
  Turunan():Kelasku(){} //<-- Konstruktor default kelas Turunan
  ```

  Namun masalahnya sekarang pada kelas dasar tidak mempunyai konstruktor tanpa parameter seperti itu, sehingga kompiler manampilkan pesan :

  ```
  no maching function call to ‘Kelasku::Kelasku()’
  ```

Maksud pesan ini adalah bahwa pada kelas Kelasku tidak terdapat konstruktor **Kelasku()** dengan tanpa parameter. Ini juga membuktikan bahwa konstruktor default mempunyai bentuk seperti yang sudah dijelaskan pada analisa percobaan **Lab.4**.

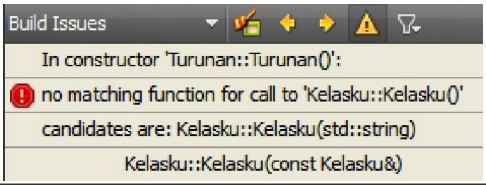
Dengan demikian jelas bahwa konstruktor kelas dasar pasti dipanggil oleh konstruktor default kelas turunan. Akan tetapi bagaimanakah jika pada kelas turunan mempunyai konstruktor sendiri? Bisakah kita membuat konstruktor sendiri pada kelas turunan tanpa memanggil konstruktor kelas dasar? Tentu saja secara eksplisit bisa kita menuliskan konstruktor pada kelas turunan tanpa memanggil konstruktor kelas dasar, namun tetap saja kompiler nantinya akan menambahkan pemanggilan konstruktor default kelas dasar (tanpa parameter) jika pada konstruktor kelas turunan tidak memanggil salah satu kontuktor kelas dasarnya. Untuk meneliti mengenai hal ini, tambahkan konstuktor yang ditulis secara eksplisit pada kelas Turunan seperti berikut.

Labs.6 Konstruktor kelas turunan harus memanggil salah satu konstruktor kelas dasar

1. Buka Qt Creator, buka project Qt Console Application dengan nama **Labs3** tadi. Kemudian tambahkan konstruktor pada kelas **Turunan** seperti berikut.

```
#include <QtCore/QCoreApplication>
#include <iostream>
using namespace std;
class Kelasku{
public:
    Kelasku(string kata){
        cout << "Konstruktor Kelas Dasar" << endl;
        cout << "Mengucapkan : " << kata << endl;
    }
public:
    void hai(){
        cout << "Hai, apa khabar...?" << endl;
    }
};

class Turunan : public Kelasku{
        Turunan(){} //<-- membuat konstruktor pada kelas turunan
};

int main(int argc, char *argv[])
{
    QCoreApplication a(argc, argv);
    Turunan test; //<-- membuat objek dari kelas Turunan saja
    return a.exec();
}
```

2. Tekan Ctrl+R untuk menjalankan kode di atas, tidak akan ada output karena ada kesalahan dengan pesan kesalahan kompilasi sebagai berikut.



Analisa Program:

- Pada program ini mencoba untuk menghindari pemanggilan kontruktor kelas dasar dengan cara membuat sendiri konstruktor pada kelas Turunan yang tidak memanggil konstuktor kelas dasar. Namun tetap saja kompiler memberikan pesan kesalahan yang sama, yaitu :

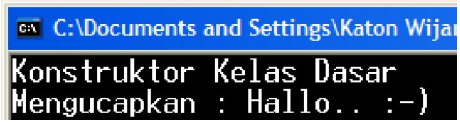
  no maching function call to 'Kelasku::Kelasku()'

  Padahal jelas pada konstruktor yang ditulis pada kelas Turunan sama sekali tidak pernah memanggil konstruktor tanpa kelas dasar parameter tersebut. Percobaan ini membuktikan bahwa bagaimanapun juga pada kelas turunan, konstruktor kelas dasar pasti dipanggil, dan jika secara eksplisit tidak dituliskan pemanggilan konstruktor kelas dasar, maka kompiler akan menambahkan pemanggilan konstruktor default (konstruktor tanpa parameter) pada kelas dasar.

3. Sekarang ubahlah konstruktor kelas Turunan, agar secara eksplisit memanggil konstuktor yang ada pada kelas dasar seperti berkut :

```
class Turunan : public Kelasku{
public:
    //memanggil konstruktor kelas dasar dng sebuah parameter
    Turunan():Kelasku("Hallo.. :-)"){}
};
```

Tekan Ctrl+R untuk menjalankan kode di atas, tidak akan ada output karena ada kesalahan dengan pesan kesalahan kompilasi sebagai berikut.

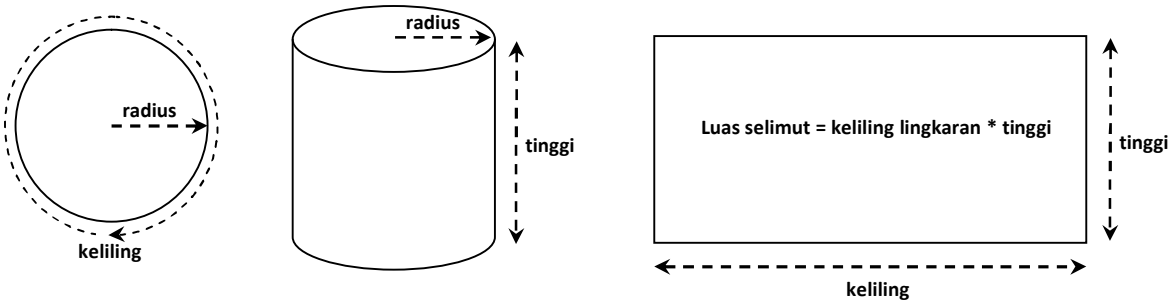


Analisa Program:

- Pada program ini sekarang kelas Turunan mempunyai konstruktor yang memanggil konstruktor kelas dasar. Karena pada kelas dasar hanya mempunyai sebuah konstruktor dengan satu parameter bertipe string yang tanpa nilai default, maka yang dipanggil adalah konstruktor dengan sebuah parameter bertipe **string**.
- Dengan demikian perlu selalu diingat, bahwa konstuktor kelas turunan harus memanggil salah satu konstruktor kelas dasar, jika tidak dilakukan maka kompiler akan menambahkan pemanggilan konstruktor default kelas dasar.

Mengganti Metode Kelas Dasar Pada Kelas Turunan (Overriding)

Ada kalanya kelas turunan mempunyai implementasi lain untuk nama metode yang sama dengan kelas dasarnya. Sebagai contoh misalnya, kelas Lingkaran mempunyai implementasi (rumus) menghitung luas pada metode **getLuas()** adalah : 3.14 * radius * radius, sedangkan kelas turunannya, mislanya Silinder mempunyai luas pada metode **getLuas()** yang terdiri dari dua luas tutup dan luas selimut dengan rumus 2 * (3.14 * radius * radius) + (2 * 3.14 * radius) * tinggiseperti gambar ilustrasi berikut:



Dengan demikian kelas turunan **Silinder** harus membuat implementasi yang berbeda untuk metode **getLuas()** supaya hasil dari metode tersebut sesuai dengan objek **Silinder**. Membuat metode yang sama dengan metode milik kelas dasarnya dinamakan ***Overriding*** (*override* artinya mengesampingkan, atau boleh juga dikatakan menimpa) dengan demikian metode yang dipanggil untuk objek dari kelas Silinder adalah metode yang baru yang ditulis pada kelas **Silinder**.

Suatu metode bisa dikatakan override dari metode kelas dasarnya jika memenuhi 2 syarat, yaitu **nama metode** dan ***signature*** dari metode tersebut sama. ***Signature*** artinya daftar parameter yang ada pada metode, yaitu banyaknya parameter maupun tipe data dari masing-masing parameter tersebut.

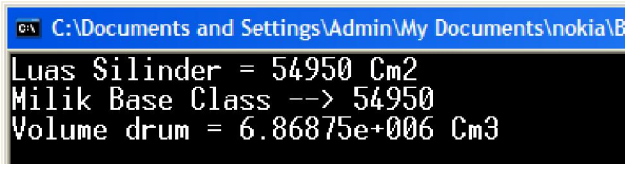
Labs.7 Melakukan Overriding

1. Buka project Qt Console Application projek Labs.1 yang tadi sudah dibuat, kemudian ubah kode menjadi seperti berikut

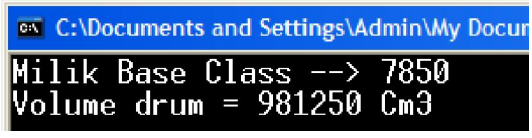
```
#include <QtCore/QCoreApplication>
#include <iostream>
using namespace std;
class Lingkaran{
public:
    //Konstruktor
    Lingkaran(float radius = 0){
        Lingkaran::radius = radius;
    }
    //Destruktor
    ~Lingkaran(){}
protected :
    float radius;
public :
    float getLuas(){
        return 3.14 * radius * radius;
    }
};

class Silinder : public Lingkaran{
public:
```

```
    //Konstruktor
    Silinder(float radius, float tinggi){
        Silinder::radius = radius; //<-- warisan
        Silinder::tinggi = tinggi; //<-- anggota baru
    }
    //Destruktor
    ~Silinder(){}
private:
    float tinggi; //<-- anggota baru

public:
    float getLuas(){ //<-- Overriding
        float luasTutup = 3.14 * this->radius * this->radius;
        float luasSelimut = 2 * 3.14 * this->radius * this->tinggi;
        float luasSilinder = 2 * luasTutup + luasSelimut;
        return luasSilinder;
    }
    float getVolume(){ //<-- anggota baru
        //getLuas() sebenarnya sudah ditimpa
        return getLuas() * tinggi;
    }
};

int main(int argc, char *argv[])
{
    QCoreApplication a(argc, argv);
    Silinder drum(50,125);
    cout << "Luas Silinder = " << drum.getLuas() << " Cm2" << endl;
    cout << "Milik Base Class --> " << drum.getLuas() << endl;
    cout << "Volume drum = " << drum.getVolume() << " Cm3" << endl;
    return a.exec();
}
```

2. Tekan Ctrl+R untuk menjalankan program diatas, outputnya adalah sebagai berikut.



Bandingkan dengan hasil keluaan pada program di **Lab.1**. seperti berikut :



Analisa:

- Pada program ini, pembuatan objek dilakukan dengan memberikan nilai radius=50 dan tinggi = 125, ini tampak pada program utama (main()) : `Silinder drum(50,125)`, sama dengan ketika membuat objek pada **Lab.1.** Namun tampak pada hasil eksekusi, Luas **Silinder**

adalah **54950 Cm2** karena menggunakan rumus baru, tetapi pada keluaran **Milik Base Class** nilainya juga sama yaitu **54950**, dan akibatnya volume drum yang seharusnya **981250 Cm3** manjadi **6.86875e+006 Cm3**. Ini semua terjadi karena pada kelas Silinder tidak lagi memakai rumus getLuas() milik kelas dasarnya. Pada satu sisi **getLuas()** pada kelas **Silinder** sudah benar, sesuai dengan yang diharapkan, namun ketika metode **getVolume()** menghitung volume memakai metode **getLuas()**, ternyata sudah berubah menjadi rumus luas Silinder yang akibatnya perhitungan volume menjadi sangat besar (salah).

- Walaupun pada kasus ini menimbulkan masalah, tetapi percobaan ini memperlihatkan adanya overriding terhadap metode milik **kelas dasar**.

Memanggil Metode Kelas Dasar

Pada kasus **Lab.7** di atas sebenarnya metode **getLuas()** masih dibutuhkan pada kelas **Silinder** untuk menghitung volume (**getVolume()**), bahkan sebenarnya untuk menghitung luas tutup, yang sebenarnya juga luas lingkaran, masih membutuhkan metode **getLuas()** milik kelas dasar **Lingkaran**. Namun karena adanya kebutuhan yang berbeda pada kelas **Silinder** untuk menghitung luasnya maka dilakukan *overriding* terhadap metode **getLuas()**. Untuk mengatasi hal ini diperlukan suatu cara untuk tetap dapat mengakses anggota milik kelas dasar, yaitu dengan cara menyebutkan nama kelas dasar kemudian diikuti dengan dua titik dua (::) dan anggota yang akan diakses.

```
<kelas_dasar>::<anggota>
```

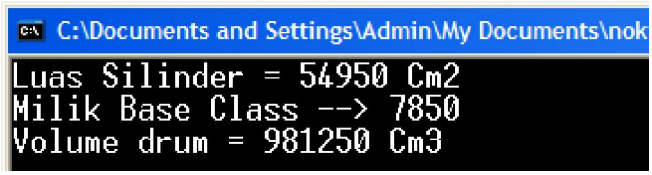
Labs.8 Mengakses metode kelas dasar

1. Buka project Qt Console Application projek Labs.1 yang baru saja dibuat, kemudian ubah kode pada metode : **getLuas()**, **getVolume()** dan program utama. Berikut ini adalah potongan program yang mengalami perubahan saja:

```
public:
    float getLuas(){ //<-- Overriding
        float luasSelimut = 2 * 3.14 * this->radius * this->tinggi;
        float luasSilinder = 2 * Lingkaran::getLuas() + luasSelimut;
        return luasSilinder;
    }
    float getVolume(){ //<-- anggota baru
        //getLuas() milik kelas dasar
        return Lingkaran::getLuas() * tinggi;
    }
};
int main(int argc, char *argv[])
{
    QCoreApplication a(argc, argv);
    Silinder drum(50,125);
    cout << "Luas Silinder = " << drum.getLuas() << " Cm2" << endl;
    cout << "Milik Base Class --> " << drum.Lingkaran::getLuas() << endl;
    cout << "Volume drum = " << drum.getVolume() << " Cm3" << endl;
    return a.exec();
}
```

Bagian yang diubah

2. Tekan Ctrl+R untuk menjalankan program diatas, outputnya adalah sebagai berikut.



Analisa:

- Sekarang keluaran "Luas Silinder" dengan "Milik Base Class" berbeda, yaitu **54950** sedangkan "Milik Base Class" yang menghitung luas lingkaran adalah **7850**. Walaupun keduanya sama-sama memanggil metode **getLuas()**, namun metode yang dipanggil pada "Milik Base Class" adalah metode milik kelas **Lingkaran**, yaitu dengan cara penulisan nama metode ditambahkan nama kelas dasar dan dua titik dua (::) didepannya seperti berikut:

  ```
  drum.Lingkaran::getLuas()
  ```

- Pada waktu mengitung **getLuas()** pada kelas **Silinder**, bisa memanfaatkan metode **getLuas()** milik kelas dasar dengan cara memanggil metode milik kelas dasar, sehingga dengan demikian metode **getLuas()** ini seperti tampak pada kode program di atas, bisa menjadi lebih ringkas.
- Demikian juga pada metode **getVolume()**, sekarang tidak ada kesalahan seperti tadi, karena rumus **getLuas()** yang digunakan sudah benar, yaitu metode **getLuas()** milik kelas dasar.
- Dari percobaan **Lab.7** ini tampak cara melakukan overriding dan cara memanggil anggota milik kelas dasar.

Penyembunyian Metode Kelas Dasar

Ketika terjadi overriding terhadap suatu metode kelas dasar, maka semua metode milik kelas dasar yang bernama sama, yaitu metode-metode yang dioverloading pada kelas dasar, akan disembunyikan (tidak diwariskan) kepada kelas turunan.

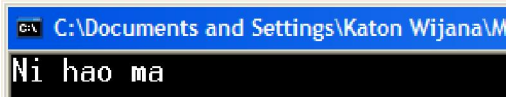
Labs.9 Penyembunyian metode kelas dasar

1. Jalankan Qt Console Application projek, buat projek bernama Labs.9 seperti berikut:

```
#include <QtCore/QCoreApplication>
#include <iostream>
using namespace std;
class Dasar{
public:
    void hallo() const{
        cout << "Hallo" << endl;
    }
    void hallo(string kata) const{
        cout << "Hallo " << kata << endl;
    }
    void hallo (string kata, string nama) const{
        cout << "Hallo " << kata << " nama saya " << nama << endl;
    }
};
```

```
class Turunan : public Dasar{
public:
    void hallo() const{ //<-- override terhadap salah satu metode
        cout << "Ni hao ma" << endl;
    }
};

int main(int argc, char *argv[])
{
    QCoreApplication a(argc, argv);
    Turunan agus;
    agus.hallo();
    //agus.hallo("apa kahabar");
    //agus.hallo("apa kahabar","Agus");
    return a.exec();
}
```

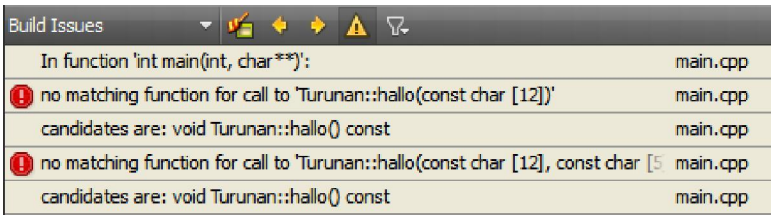
2. Tekan Ctrl+R untuk menjalankan program diatas, outputnya adalah sebagai berikut.



3. Hilangkan tanda comment pada pemanggilan metode hallo pada dua baris di progam utama menjadi seperti berikut:

```
int main(int argc, char *argv[])
{
    QCoreApplication a(argc, argv);
    Turunan agus;
    agus.hallo();
    agus.hallo("apa kahabar");
    agus.hallo("apa kahabar","Agus");
    return a.exec();
}
```

4. Tekan Ctrl+R untuk menjalankan program diatas, maka tidak akan ada output dan menucul pesan kesalahan saat kompilasi sepeti berikut.



Analisa:

- Tampak pada hasil kompilasi kedua tidak dijalankan, pesan kesalahan adalah tidak ditemukannya metode dengan sebuah parameter char[12] (string) dan metode dengan dua buah parameter kedua-duanya bertipe char[12] (string). Padahal jelas terlihat pada kelas **Dasar** terdapat kedua metode tersebut dan jenis aksesnya adalah `public`, bahkan cara penurunannya pada kelas **Dasar** juga menggunakan jenis akses `public`, seharusnya dengan

cara demikian metode-metode tersebut diwariskan kepada kelas **Turunan**. Percobaan ini menunjukkan bahwa jika terjadi overriding terhadap suatu metode kelas dasar, maka semua metode milik kelas dasar yang bernama sama, yaitu metode-metode yang dioverloading pada kelas dasar, akan disembunyikan (tidak diwariskan) kepada kelas turunan.

Metode Virtual

Metode Virtual adalah metode yang seharusnya dioverride oleh kelas turunannya, dengan tujuan jika ada variabel pointer bertipe kelas dasarnya yang berisi objek bertipe kelas turunan tersebut, maka metode yang menanggapi adalah metode milik kelas turunan hasil override tersebut. Ini sebenarnya bagian yang sangat penting yang diperlukan pada polimorfisme yang akan dibahas pada bab berikutnya.

Jika suatu metode pada kelas dasar tidak virtual, kemudian pada kelas turunan melakukan override terhadap metode tersebut, maka jika ada suatu variabel pointer bertipe kelas dasar yang berisi objek bertipe kelas turunan memanggil metode tersebut, yang menanggapi adalah metode milik kelas dasar. Percobaan berikut ini merupakan penyederhanaan dari kelas **Lingkaran** sebagai kelas dasardan kelas **Silinder** sebagai kelas turunan untuk memahami metode virtual.

Labs.10 Metode virtual dan non virtual

1. Jalankan Qt Console Application projek, buat projek bernama Labs.10 seperti berikut:

```
#include <QtCore/QCoreApplication>
#include <iostream>
using namespace std;
class Lingkaran{
public:
    virtual float getLuas(){
        cout << "Luas Lingkaran" << endl;
        return 0;
    }
};

class Silinder : public Lingkaran{
public:
    float getLuas(){
        cout << "Luas Silinder" << endl;
        return 0;
    }
};

int main(int argc, char *argv[])
{
    QCoreApplication a(argc, argv);
    Lingkaran *objek1;       //<-- variabel pointer bertipe kelas dasar
    objek1 = new Silinder(); //<-- objek bertipe kelas turunan
    objek1->getLuas();       //<-- memanggil metode yang dioverride
    return a.exec();
}
```

2. Tekan Ctrl+R untuk menjalankan program diatas, hasil keluaran adalah sepeti berikut.

3. Sekarang hapuslah kata kunci `virtual` metode **getLuas()** pada kelas **Lingkaran** seperti berikut.

```
class Lingkaran{
public:
    float getLuas(){
        cout << "Luas Lingkaran" << endl;
        return 0;
    }
};
```

4. Tekan Ctrl+R untuk menjalankan program diatas, hasil keluaran adalah sepeti berikut.

Analisa:

- Tampak pada program utama, variabel pointer bertipe kelas dasar (**Lingkaran**) digunakan untuk menunjuk objek bertipe kelas turunan (**Silinder**). Ketika dipanggil metode **getLuas()** yang ada di kelas dasar maupun turunan, maka metode mana yang menanggapi tergantung apakah metode tersebut bersifat **virtual** pada kelas dasar atau tidak.
- Pada percobaan pertama, metode **getLuas()** pada kelas Lingkaran dibuat **virtual**, ketika program utama dijalankan, tampak bahwa yang menanggapi adalah metode milik kelas turunan (metode yang ada pada objek), yaitu mencetak “Luas Silinder”.
- Pada percobaan kedua, tidak ada perubahan kode program sama sekali kecuali menghilangkan kata kunci **virtual** pada metode **getLuas()** pada kelas Lingkaran, hasilnya tampak bahwa yang menanggapi adalah metode milik kelas dasar (metode yang ada pada kelas Lingkaran), yaitu mencetak “Luas Lingkaran”.
- Percobaan pertama, yaitu membuat metode **virtual**, adalah yang diperlukan pada poses polimorfisme yang akan dibahas pada bab berikutnya.

Catatan

- Kata kunci **virtual** tidak membuat metode tersebut harus dioverride.
- Pada kelas turunan metode **virtual** yang dioveride otomatis **virtual** walaupun tidak dtiulis, namun sebaiknya untuk kemudahan perawatan sebaiknya kata kunci **virtual** ditulis.
- Variabel pointer bertipe kelas dasar jika digunakan untuk menunjuk objek bertipe kelas turunan, untuk mengakses anggota kelas turunannya, ia hanya bisa memanggil metode **virtual** yang dioverride oleh kelas turunannya. Dengan kata lain, variabel tersebut tidak bisa memanggil metode-metode kelas turunan yang bukan merupakan override dari metode **virtual** kelas dasar.

Pemotongan (*Slicing*)

Perlu diperhatikan bahwa kemampuan untuk memanggil metode kelas turunan dari variabel bertipe kelas dasar hanya berlaku untuk variabel pointer dan variabel referensi, sedangkan variabel nilai (*value variable*) tidak dapat mengakses metode virtual seperti itu.

Ketika kita membuat objek bertipe kelas turunan, sebenarnya sebelumnya sudah dibuat objek bertipe kelas dasarnya, seperti sudah dibahas secara tidak langsung pada bagian yang membicarakan konstruktor di atas. Jadi ada bagian yang merupakan anggota kelas dasar dan ada bagaian yang merupakan anggota kelas turunan. Sebagai contoh misalnya berikut ini ilustrasi mengenai objek bertipe **Silinder** yang merupakan turunan dari kelas **Lingkaran** yang dibahas pada Lab.10 di atas.

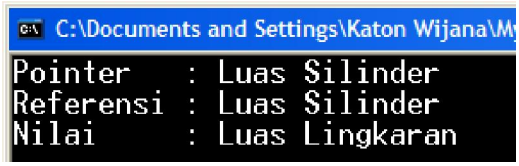
Pada konversi dari suatu variabel ke variabel lain, bisa terjadi **Pemotongan** (***Silicing***). Supaya lebih jelas lakukan percobaan berikut ini.

Labs.11 Metode virtual dan non virtual

1. Jalankan Qt Console Application projek, buka projek bernama **Labs.10** yang dibuat tadi, kemudian ubah kode progam pada bagian program utama seperti berikut:

```
int main(int argc, char *argv[])
{
    QCoreApplication a(argc, argv);
    Silinder* s = new Silinder(); //<-- objek bertipe Silinder
    Lingkaran* objek1 = s;  //<-- variabel pointer bertipe kelas dasar
    Lingkaran& objek2 = *s; //<-- variabel referensi bertipe kelas dasar
    Lingkaran objek3 = *s;  //<-- variabel nilai bertipe kelas dasar
    cout << "Pointer   : " ;
    objek1->getLuas(); //<-- memanggil dai variabel pointer
    cout << "Referensi : " ;
    objek2.getLuas(); //<-- memanggil dai variabel referensi
    cout << "Nilai     : " ;
    objek3.getLuas(); //<-- memanggil dai variabel nilai
    return a.exec();
}
```

2. Tekan Ctrl+R untuk menjalankan program diatas, hasil keluaran adalah sepeti berikut.



Analisa:

- Tampak pada hasil percobaan, pointer dan referensi memanggil metode virtual, sehingga

yang dieksekusi adalah metode **getLuas()** milik **Silinder**. Ini tampak pada 2 baris pertama hasil keluaran di atas. Sedangkan pada baris ke 3 metode yang dieksekusi adalah metode milik kelas dasar (Lingkaran) itu sendiri.

- Variabel **objek3** bertipe kelas dasar (Lingkaran), maka ketika menerima objek bertipe kelas turunan (Silinder) kompiler memotong (*slices down*) objek Silinder menjadi bentuk kelas dasar (Lingkaran) saja. Oleh karena itu ketika dipanggil metode **getLuas()** maka yang ada hanya anggota-anggota kelas dasar (Lingkaran). Potongan objek bertipe kelas turunan (Silinder) hilang, inilah efek dari variabel nilai yang diberi nilai objek kelas turunannya, efek ini disebut **Pemotongan** (***Slicing***) karena bagian kelas turunan (Silinder) dipotong keluar ketika dikonversikan menjadi kelas dasar (Lingkaran).

Memakai static_cast

Seperti tertulis pada catatan di atas, pointer bertipe kelas dasar yang menunjuk objek bertipe kelas turunan tidak bisa memanggil metode-metode kelas turunan yang bukan merupakan override dari metode **virtual** kelas dasar. Namun dengan mekanisme casting hal ini bisa dilakukan.

Casting adalah mekanisme yang digunakan untuk mengubah tipe variabel dari tipe data ke tipe data yang lain. **`static_cast`** adalah mekanisme yang dapat digunakan untuk mengkonversikan pointer ke tipe pointer lain yang mempunyai hubungan kekerabatan (*inheritance*) dan melakukan konversi secara ekplisit tipe data standar. **`static_cast`** pada saat kompilasi melakukan pemeriksaan untuk memastikan bahwa pointer yang akan di-"cast" mempunyai hubungan yang benar. Dengan **`static_cast`** sebuah pointer dapat di-"***up-casted***" menjadi tipe kelas dasar, atau bisa pula di-"***down-casted***" menjadi bertipe kelas turunan. Jika suatu variabel bertipe kelas dasar diberi nilai objek bertipe kelas turunan sebagai berikut:

```
Kelas_dasar* variabel1 = new Kelas_turunan();
```

Maka tipe data variabel tersebut bisa di-"***down-casted***" menjadi tipe data kelas turunan dengan cara:

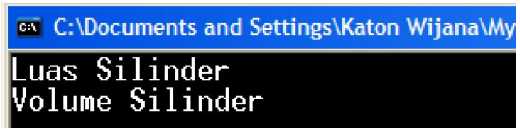
```
Kelas_turunan* variabel2 = static_cast<Kelas_turunan*>(variabel1);
```

Labs.11 Memakai static_cast

1. Jalankan Qt Console Application projek, buka projek bernama Labs.10 di atas, ubah pada bagian program utama (**main()**) menjadi seperti berikut:

```
int main(int argc, char *argv[])
{
    QCoreApplication a(argc, argv);
    Lingkaran* objek1;       //<-- variabel pointer bertipe kelas dasar
    objek1 = new Silinder(); //<-- objek bertipe kelas turunan
    objek1->getLuas();       //<-- memanggil metode yang dioverride
    static_cast<Silinder*>(objek1)->getVolume(); //<-- casting mjd Silinder
    return a.exec();
}
```

2. Tekan Ctrl+R untuk menjalankan program diatas, hasil keluaran adalah sepeti berikut.



Analisa:

- Tampak pada program utama, variabel pointer bertipe kelas dasar (**Lingkaran**), yaitu **objek1**, di-"***down-casted***" menjadi tipe data kelas turunan (Silinder). Ketika dipanggil metode **getVolume()** yang sebenanya tidak ada di kelas dasar tetapi hanya ada di kelas turunan, tampak bahwa metode **getVolume()** milik kelas Silinder menanggapinya.
- Percobaan ini menunjukkan bahwa tipe data pointer dapat dikonversikan menjadi tipe data pointer lain.

Catatan

- Casting terhadap pointer hanya bisa dilakukan terhadap tipe-tipe yang mempunyai hubungan kekerabatan (*inheritance*).
- Dengan **casting** kita bisa memaksa variabel pointer bertipe kelas dasar yang digunakan untuk menunjuk objek bertipe kelas turunan untuk memanggil metode yang bukan merupakan override dari metode virtual kelas dasar, namun cara ini tidak cukup aman jika objek tersebut bukan dari kelas turunan.