Bahasa C++: Inheritance

IF2210 – Semester II 2022/2023

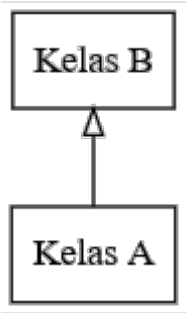
Sumber: Diktat Bahasa C++ oleh Hans Dulimarta

Pewarisan & penurunan kelas

- Konsep-konsep yang berkaitan erat OOP: objek, kelas, pewarisan (*inheritance*), *polymorphism*, dan *dynamic binding*.
- Pewarisan merupakan ciri unik dari OOP.
- Pewarisan: pendefinisian dan pengimplementasian sebuah kelas berdasarkan kelas-kelas yang sudah ada (*reuse*).

Pewarisan & penurunan kelas

- Kelas A mewarisi kelas B:
  - A = kelas turunan (*derived class/subclass*), dan
  - B = kelas dasar (*base class/superclass*)



  - Seluruh atribut & method B diwariskan ke A, kecuali ctor, dtor, cctor, dan operator=. A memiliki ctor, cctor, dtor, dan operator= sendiri.
  - Kelas A akan memiliki dua bagian:
    1. Bagian yang diturunkan dari B, dan
    2. Bagian yang didefinisikan sendiri (spesifik terhadap A)
  - Fungsi di dalam kelas turunan dapat mengakses semua atribut & method di dalam bagian non-private.

Penurunan kelas dalam C++

```
class kelas-turunan: mode-pewarisan kelas-dasar
  // ...
;
```

- *Mode-pewarisan*: mempengaruhi tingkat pengaksesan setiap anggota (method & atribut) kelas dasar jika diakses melalui fungsi/method **di luar** kelas dasar maupun di luar kelas turunan.
- Contoh:

```
class Minibus: public Kendaraan {
  // ...
};
```

Penurunan kelas dalam C++

- Perubahan tingkat pengaksesan akibat pewarisan:

Tingkat akses di *base class*	Mode pewarisan		
	`private`	`protected`	`public`
`private`	`private`	`private`	`private`
`protected`	`private`	`protected`	`protected`
`public`	`private`	`protected`	`public`

- private: = “sangat tertutup” (hanya method kelas tersebut yang dapat mengakses,
- public: = “sangat terbuka” (fungsi/method manapun, di dalam atau di luar kelas dapat mengakses anggota dalam bagian ini),
- protected: “setengah terbuka”/“setengah tertutup” (hanya kelas turunan yang dapat mengakses anggota pada bagian ini).

Contoh pewarisan

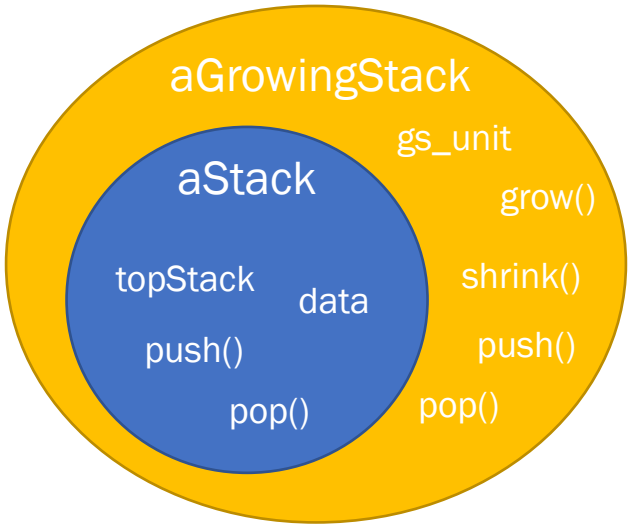
- Growing Stack: Stack yang kapasitasnya dapat bertambah/berkurang secara otomatis
  - `push()`: jika Stack penuh perbesar kapasitas
  - `pop()`: jika kapasitas tak terpakai cukup besar perkecil kapasitas
- Kelas `GStack` dapat diwariskan dari kelas `Stack` dengan cara:
  1. Mengubah perilaku `pop()` dan `push()` yang ada pada kelas `Stack`.
  2. Menambahkan atribut yang digunakan untuk menyimpan faktor penambahan/penciutan kapasitas stack

```
// File GStack.h - Deklarasi kelas GStack

#ifndef GSTACK_H
#define GSTACK_H

#include "Stack.h"

class GStack: public Stack {
  public:
    // ctor, cctor, dtor, oper= (tidak dituliskan)
    // redefinition of push & pop
    void push (int);
    void pop (int&);
  private:
    int gs_unit;
    // method untuk mengubah kapasitas
    void grow();
    void shrink();
};

#endif GSTACK_H
```

```
// File GStack.cc
// Definisi method-method kelas GStack

#include <stdio.h>
#include "GStack.h"

void GStack::push (int x) {
  if (isFull())
    grow();
  Stack::push(x);
}

void GStack::pop (int& x) {
  Stack::pop(x);
  if (size - topStack > gs_unit)
    shrink();
}
```

ctor, dtor, cctor, dan `operator=` kelas dasar

- Komponen yang berasal dari kelas dasar dapat dianggap sebagai “sub-objek” dari kelas turunan
- Pada penciptaan objek kelas turunan, konstruktor kelas dasar akan diaktifkan **sebelum** konstruktor kelas turunan.
- Pada pemusnahan objek kelas turunan, destruktor kelas dasar dipanggil **setelah** destruktor kelas turunan.



Penanganan *copy constructor*

- Ada tiga kasus:
  1. Kelas turunan tidak memiliki cctor, kelas dasar memiliki
  2. Kelas turunan memiliki cctor, kelas dasar tidak memiliki
  3. Baik kelas turunan maupun kelas dasar memiliki cctor
- Kasus (1): cctor kelas dasar akan dipanggil, inisialisasi kelas turunan dilakukan secara *bitwise copy*
- Pada kasus (2) dan (3), cctor dari kelas dasar **tidak dipanggil**, inisialisasi kelas dasar menjadi tanggung jawab kelas turunan.

Penanganan *copy constructor*

- Penginisialisasian kelas dasar oleh kelas turunan melalui ctor atau cctor dilakukan melalui *constructor initialization list*

```
// File GStack.cc
// Definisi method-method kelas GStack

#include <stdio.h>
#include "GStack.h"

GStack::GStack(const GStack& s): Stack(s) {
  gs_unit = s.gs_unit;
}
```

Operator assignment

Assignment ditangani seperti inisialisasi.

- Jika kelas turunan **tidak** mendefinisikannya, operator= dari kelas dasar akan dipanggil (jika ada).
- Jika kelas turunan mendefinisikan operator= maka operasi *assignment* dari kelas dasar menjadi tanggung jawab kelas turunan.

```
// File GStack.cc
#include <stdio.h>
#include "GStack.h"

GStack& GStack::operator=(const GStack& s) {
  Stack::operator=(s); // oper= dari Stack
  gs_unit = s.gs_unit;
  return *this;
}
```

Polymorphism

- Objek-objek dari kelas turunan memiliki sifat sebagai kelas tersebut dan sekaligus kelas dasarnya.
  - *polymorphism* (*poly* = banyak, *morph* = bentuk).
- *reference* (“ref”) dan *pointer* (“ptr”) dapat bersifat polimorfik.
- Dalam C++ ref/ptr dapat digunakan untuk *dynamic binding.* ref/ptr memiliki tipe statik dan tipe dinamik.
  - Tipe statik = tipe objek pada saat deklarasikan,
  - Tipe dinamik = tipe objek pada saat eksekusi, dapat berubah bergantung pada objek yang diacu.

Dynamic binding

- Tipe dinamik digunakan pada saat eksekusi untuk memanggil method yang berasal dari kelas berbeda-beda melalui sebuah ref/ptr dari **kelas dasar.**
- Method yang akan dipanggil secara dinamik, harus dideklarasikan sebagai *virtual* (dilakukan di **kelas dasar**).
- Dalam contoh `Gstack` method `push()` dan `pop()` akan dipanggil secara dinamik
  - Kelas `Stack` harus mendeklarasikan sebagai method virtual

```
// File: Stack.h
class Stack {
  public:
    // ctor, cctor, dtor, & oper=

  public:
    // services
    virtual void push(int); // <=== penambahan "virtual"
    virtual void pop(int&); // <=== penambahan "virtual"
};
```

```
#include <GStack.h>

// tipe memiliki tipe dinamik
void funcVal(Stack s) { s.push (10); }

// memiliki tipe dinamik
void funcPtr(Stack *t) { t->push (10); }

// memiliki tipe dinamik
void funcRef(Stack& u) { u.push (10); }

main() {
  GStack gs;
  funcVal(gs); // Stack::push() dipanggil
  funcPtr(&gs); // GStack::push() dipanggil
  funcRef(gs); // GStack::push() dipanggil
}
```

Virtual destructor

```
Stack* sp[MAX_ELEM];
// ... kode-kode lain
for (i=0; i<MAX_ELEM; i++)
  delete sp[i]; // tipe elemen: Stack/ GStack !
```

- Destruktor mana yang akan dipanggil `delete sp[i];`?
- Diperlukan *virtual destructor*

```
class Stack {
  public:
    // ctor, dtor, cctor
    //...
    virtual ~Stack();
    //
};
```

Virtual destructor

- Di dalam kelas turunan (`GStack`), destruktor **tidak perlu** dideklarasikan virtual karena otomatis mengikuti jenis dtor kelas dasar.

```cpp
class Base {
  public:
    virtual ~Base() { /* releases Base's resources */ }
};

class Derived: public Base {
  ~Derived() { /* releases Derived's resources */ }
};

int main() {
  Base* b = new Derived;
  delete b; // Makes a virtual function call to Base::~Base()
            // since it is virtual, it calls Derived::~Derived()
            // which can release resources of the derived class,
            // and then calls Base::~Base() following the usual
            // order of destruction
}
```

Abstract base class (ABC)

- *Abstract Base Class* = kelas dasar untuk objek abstrak.
  - Contoh: `DataStore`, `Vehicle`, `Shape`, dst.
- Bagaimana implementasi `Shape::Draw()`?
  - Tidak dapat diimplementasikan di dalam kelas `Shape`, namun harus dideklarasikan.
  - Dideklarasikan sebagai method *pure virtual*.
- ABC = kelas yang memiliki method *pure virtual*

Abstract base class

```
class DataStore {
  public:
    // ...
    virtual void Clear() = 0; // pure virtual
    // ...
};
```

- Tidak ada objek yang dapat dibuat dari kelas dasar abstrak
- Di dalam kelas non-abstrak (yang mewarisi kelas dasar abstrak) method *pure virtual* harus diimplementasikan.

```
// File: Stack.h
// Deskripsi: deklarasi kelas Stack yang diturunkan dari
//            DataStore

class Stack: public DataStore {
  public:
    // ...
    void Clear ();
};

// File: Tree.h
// Deskripsi: deklarasi kelas Tree yang diturunkan dari DataStore
class Tree: public DataStore {
  public:
    // ...
    void Clear ();
};
```