MINGGU 3 & 4

Praktikum Pemrograman Berbasis Objek
oleh Asisten IF2210 2019/2020

OUTLINE

1. Review Praktikum Minggu 1 & 2
2. Inheritance
3. Polymorphism
4. Virtual function
5. Abstract Base Class

REVIEW PRAKTIKUM MINGGU I & 2

- Soal 1
  - Kesalahan umum:
    - Menulis `delete this` di destruktor
    - Menulis boii...iing sebanyak radius, seharusnya huruf i yang sebanyak radius
- Soal 2
  - Panggil constructor dengan `new` untuk dapat mengontrol urutan destructor dipanggil
- Soal lainnya rata-rata ada kesalahan salah ketik atau salah baca soal (misal: setHeight pada Bottle)

INHERITANCE

- Inheritance merupakan komponen penting dalam pemrograman berorientasi objek.
- Inheritance memungkinkan kita membuat sebuah *class* dari suatu *class* lain

CONTOH INHERITANCE : VEHICLE

```
class Vehicle {
protected:
    float fuel;
public:
    Vehicle();
    void addFuel(float fuel);
    void drive();
};
```

```
Vehicle::Vehicle() {
    this->fuel = 0;
}

void Vehicle::addFuel(float fuel) {
    this->fuel += fuel;
}

void Vehicle::drive() {
    cout << "Driving ..." << endl;
}
```

CONTOH INHERITANCE : MOTORCYCLE

```
class MotorCycle : public Vehicle {
public:
    MotorCycle();
    void drive();
};
```

MotorCycle adalah turunan dari Vehicle

```
MotorCycle::MotorCycle() : Vehicle() {
    //
}

void MotorCycle::drive() {
    cout << "Use helmet" << endl;
    Vehicle::drive();
}
```

Memanggil method milik parent

CONTOH INHERITANCE : CAR

```
class Car : public Vehicle {
public:
    Car();
    void openDoor();
    void drive();
};
```

Dapat menambahkan method lain

```
Car::Car() : Vehicle() {
    //
}

void Car::openDoor() {
    cout << "Door opened" << endl;
}

void Car::drive() {
    cout << "Use seatbelt" << endl;
    Vehicle::drive();
}
```

CONTOH INHERITANCE: APLIKASI

```
int main() {
    Car c;
    c.addFuel(20);
    c.openDoor();
    c.drive();

    MotorCycle m;
    m.addFuel(10);
    m.drive();
}
```

POLYMORPHISM

- *Polymorphism* = memiliki banyak bentuk
- Sebuah objek yang memiliki *parent class* (melalui *inheritance*) dapat "bertingkah" seperti *parent class* tersebut

POLYMORPHISM

```
void naiki(Vehicle* v) {
    v->addFuel(10);
    v->drive();
}

int main() {
    Car c;
    naiki(&c);
}
```

Output:

???

POLYMORPHISM

```
void naiki(Vehicle* v) {
    v->addFuel(10);
    v->drive();
}

int main() {
    Car c;
    naiki(&c);
}
```

Output:

Driving ...

VIRTUAL FUNCTION

Pada contoh sebelumnya, Vehicle tidak menjalankan definisi pada kelas anak. Untuk itu, kita perlu menandai method kelas Vehicle sebagai virtual.

Dengan demikian, definisi method akan dicari ke kelas anak

VIRTUAL FUNCTION

```
class Vehicle {
protected:
    float fuel;
public:
    Vehicle();
    void addFuel(float fuel);
    virtual void drive();
};
```

Output:

Use seatbelt
Driving ...

VIRTUAL FUNCTION

```
void naiki(Vehicle* v) {
    v->addFuel(10);
    v->drive();
}

int main() {
    Car c;
    naiki(&c);
}
```

Catatan: jika parameter naiki() bukan pointer, maka hanya akan muncul “Driving …”

Jika parameter naiki() adalah reference, maka tetap muncul “Use seatbelt”

VIRTUAL DESTRUCTOR

- Bagaimana jika destructor? Apakah perlu diberi virtual?
  - Jika tidak diberi virtual, maka bisa jadi destructor yang dipanggil hanya destructor kelas parent.
  - Dengan demikian, beberapa hal yang perlu di-*free* oleh kelas anak, tidak dilaksanakan.
  - Secara praktik, menjadikan destructor virtual pada kelas parent lebih baik.

PURE VIRTUAL FUNCTION

- Pada kelas parent, terkadang ada method yang tidak dapat diimplementasikan, misalnya `Vehicle` memiliki `getNumOfWheels`. Tergantung kelas anaknya, nilai kembaliannya bisa berbeda-beda
- Karena itu, method ini disebut *pure virtual*, yakni tidak memiliki implementasi oleh kelas *parent*.

CONTOH PURE VIRTUAL

```
class Vehicle {
protected:
    float fuel;
public:
    Vehicle();
    void addFuel(float fuel);
    virtual void drive();
    virtual int getNumOfWheels() = 0;
};
```

```
// Car.cpp
int Car::getNumOfWheels() {
    return 4;
}

// MotorCycle.cpp
int MotorCycle::getNumOfWheels() {
    return 2;
}
```

ABSTRACT BASE CLASS

- Kelas yang memiliki setidaknya satu pure virtual function disebut sebagai abstract base class, kelas yang tidak dapat diinstansiasikan
- Di bahasa lain, ini juga sering disebut sebagai interface