MINGGU 7 & 8

Praktikum Pemrograman Berbasis Objek
oleh Asisten IF2210 2019/2020

OUTLINE

1. Constructor, member, dan method
2. Main program
3. Method overloading
4. Inheritance
5. Abstract Class
6. Interface

KELAS DI JAVA

	C++	JAVA
1.	**cctor**	Tak ada terjemahan
2.	**ctor initialization list**	Tak ada
3.	Operator overloading	Tak ada, fungsi biasa
4.	Parameter passing: by value, by ref	Hanya by value
5.	Templates	Kelas generik
6.	class	public class, private class, protected, package private
7.	Inheritance: public, protected, private	Inheritance: hanya ada public
8.	Multi-inheritance	"tidak boleh" + interface
9.	ABC	Interface
10.	Abstract class	Abstract class
11.	Pemanggilan ctor base class eksplisit tidak bisa	Ctor superclass bisa diinvokasi: super() – hanya baris pertama ctor

Sumber: slide kuliah

KELAS DI JAVA

C++	JAVA
1. friend	1. Tidak ada friend
2. main program tidak mungkin dalam kelas	2. main adalah *method* sebuah kelas
3. Scope: kelas + file	3. *Scope*: kelas (*catatan)
4. namespace	4. package
5. bool	5. Boolean
6. string	6. String
7. Try-catch: tidak ada *keyword* "finally"	7. Try-catch: Ada *keyword* "finally"
8. Throw objek apapun, walau ada kelas "exception"	8. Throw: harus objek turunan throwable

Sumber: slide kuliah

CONTOH CLASS: STACK

```h
// stack.h
#ifndef STACK_H
#define STACK_H

class Stack {
private:
    int size;
    int capacity;
    int* data;
public:
    Stack();
    Stack(int cap);
    Stack(const Stack& s);
    ~Stack();

    void push(int x);
    int pop();
};

#endif
```

```java
// Stack.java
// nama file harus sama dengan nama class

class Stack {
  private int size;
  private int capacity;
  private int[] data;

  public Stack() { /* TODO */ }
  public Stack(int cap) { /* TODO */ }

  public void push(int x) { /* TODO */ }
  public int pop() { /* TODO */ }
}
```

Di java, deklarasi dan definisi kelas tidak dipisah

CONTOH CLASS: STACK

```cpp
// stack.cpp
#include "stack.h"

Stack::Stack() {
    this->capacity = 10;
    this->size = 0;
    this->data = new int[this->capacity];
}

Stack::Stack(int cap) {
    this->capacity = cap;
    this->size = 0;
    this->data = new int[this->capacity];
}
```

```java
// Stack.java
class Stack {
 ...
 public Stack() {
   this.capacity = 10;
   this.size = 0;
   this.data = new int[this.capacity];
 }

 public Stack(int cap) {
   this.capacity = cap;
   this.size = 0;
   this.data = new int[this.capacity];
 }
 ...
}
```

CONTOH CLASS: STACK

```cpp
// stack.cpp
#include "stack.h"

void Stack::push(int x) {
    if (this->size < this->capacity) {
        this->data[this->size] = x;
        this->size++;
    }
}

int Stack::pop() {
    int top = 0;
    if (this->size > 0) {
        this->size--;
        top = this->data[this->size];
    }
    return top;
}
```

```java
// Stack.java
class Stack {
 ...
 public void push(int x) {
   if (this.size < this.capacity) {
     this.data[this.size] = x;
     this.size++;
   }
 }

 public int pop() {
   int top = 0;
   if (this.size > 0) {
     this.size--;
     top = this.data[this.size];
   }
   return top;
 }
 ...
}
```

MAIN PROGRAM

- Program utama merupakan kelas yang memiliki sebuah method
  `public static void main(String[] args)`
- Lakukan kompilasi dengan command
  `javac Stack.java StackMain.java`
- Berbeda dengan C++ yang menghasilkan executable files, java menghasilkan file class yang perlu dieksekusi dalam JVM dengan command
  `java StackMain`

CONTOH CLASS: STACK (MAIN)

```cpp
#include <iostream>
#include "stack.h"
using namespace std;

int main() {
    Stack s1;
    s1.push(2);
    s1.push(3);
    cout << s1.pop() << endl;

    Stack s2(15);
    s2.push(5);
    cout << s2.pop() << endl;

    Stack s3(s1);
    s3.push(1);
    cout << s3.pop() << endl;
    cout << s3.pop() << endl;
}
```

```java
class StackMain {
 public static void main(String[] args) {
   Stack s1 = new Stack();
   s1.push(2);
   s1.push(3);
   System.out.println(s1.pop());

   Stack s2 = new Stack(15);
   s2.push(5);
   System.out.println(s2.pop());

   // tidak ada copy constructor

 }
}
```

OVERLOADING DI JAVA

- Di java, tidak ada operator overloading
- Namun java menyediakan method overloading

INHERITANCE

- Inheritance Java mirip dengan C++, tapi tidak ada multiple inheritance. Selain itu, inheritance di java hanya bersifat public.
- Cara menuliskan bahwa sebuah kelas A inherit dari kelas B:
  `class A extends B`

CONTOH KELAS REKENING

```
class Rekening {
 private String nama;
 private double saldo;
 private double sukuBunga;

 public Rekening(String nama, double saldo, double sukuBunga) {
   this.nama = nama;
   this.saldo = saldo;
   if (this.saldo < 0) {
     this.saldo = 0;
   }
   this.sukuBunga = sukuBunga;
 }

 public double hitungBiaya() {
   return Math.min(0.1 * this.saldo, 10.0);
 }

...
```

```
 public String getNama() {
   return this.nama;
 }

 public double getSaldo() {
   return this.saldo;
 }

 public double getSukuBunga() {
   return this.sukuBunga;
 }
}
```

CONTOH KELAS REKENING

```
class Rekening {
 ...

 public void setor(double uang) { /* abstract? */ }
 public void tarik(double uang) { /* abstract? */ }

 /* update saldo tiap awal bulan*/
 public void update() { /* abstract? */ }

 ...
}
```

Tergantung jenis rekening (tabungan, giro, atau deposito), mekanisme setor, tarik, dan update bisa berbeda-beda

KELAS ABSTRAK

- Seperti C++, Java mampu mendefinisikan kelas abstrak.
- Artinya, kelas tersebut tidak dapat diinstansiasikan.
- Untuk disebut abstrak, sebuah kelas memiliki minimal satu method abstrak (di C++ disebut pure virtual)

CONTOH KELAS ABSTRAK REKENING

```
abstract class Rekening {
 ...

 abstract public void setor(double uang);
 abstract public void tarik(double uang);
 abstract public void update();

 ...
}
```

CONTOH KELAS TURUNAN REKENING: REKENING TABUNGAN

```
class RekeningTabungan extends Rekening {
 public RekeningTabungan(String nama, double saldo) {
   super(nama, saldo, 0.05);
 }

 public void setor(double uang) {
   this.saldo += uang;
 }

 public void tarik(double uang) {
   if (this.saldo >= uang) {
     this.saldo -= uang;
   }
 }

 public void update() {
   this.saldo += (this.getSukuBunga() * this.saldo - this.hitungBiaya());
 }
}
```

Di kelas anak, sudah tidak ada keyword abstract, yang artinya kelas dapat diinstansiasikan.

Namun, saldo merupakan private member dari kelas Rekening, sehingga nilainya tidak dapat diakses

KOREKSI KELAS ABSTRAK REKENING

```
abstract class Rekening {
 protected String nama;
 protected double saldo;
 protected double sukuBunga;

 ...
```

Karena itu, kita ubah akses ke nama, saldo, dan sukuBunga menjadi protected.

Selain itu, opsi lainnya adalah membuat getter dan setter yang public. Opsi ini lebih baik, namun untuk kemudahan penjelasan, kita gunakan opsi pertama.

ref: https://stackoverflow.com/questions/1568091/why-use-getters-and-setters-accessors#1568230

INTERFACE

- Di Java, kita dapat membuat interface: seperti kelas C++ yang:
  - hanya memiliki method
  - semua methodnya pure virtual
- Interface != header
- Berbeda dengan inheritance, sebuah kelas bisa implement banyak interface sekaligus
- Cara mendefinisikan interface:
  `interface IA { ... }`
- Cara menggunakan interface:
  `class A implements IA { ... }`

INTERFACE

- Interface menambahkan abstraksi ke kelas yang kita buat.
- Sebagai contoh, Anda bisa membuat interface List yang memiliki method getElement, setElement, add, dll.
- Pengguna List tidak peduli dengan implementasinya: apakah menggunakan array, list rekursif, double pointer, atau lainnya.
- Bahkan Anda juga bisa membuat implementasi List Anda sendiri, namun masih mampu polymorph dengan List milik orang lain.

INTERFACE

- Sebuah interface dapat mengextend interface lain.
- Misalnya, interface IRekening memiliki sifat bisa ditarik dan disetor, sehingga kita buat interface Tarik-able dan Setor-able

CONTOH PENGGUNAAN INTERFACE

```
interface IRekening extends Tarikable, Setorable {
 public void update();

 public String getNama();
 public double getSaldo();
 public double getSukuBunga();

 public double hitungBiaya();
}
```

```
interface Setorable {
 public void setor(double uang);
}
```

```
interface Tarikable {
 public void tarik(double uang);
}
```

INTERFACE

```
interface LivingThing {
  public void eat();
  public void breathe();
  public void haveChild();
  public void grow();
  public void move();
}
```

```
class AnaerobicBacteria implements LivingThing {
  public void eat();
  public void breathe(); // Gimana cara implement ini???
  public void haveChild();
  public void grow();
  public void move();
}
```

INTERFACE SEGREGATION PRINCIPLE

```
interface Breathing {
  public void breathe();
}
```

```
interface Moveable {
  public void move();
}
```

```
interface Reproduce {
  public void haveChild();
}
```

```
class AnaerobicBacteria implements Moveable, Reproduce {
  public void haveChild();
  public void move();
}
```

By splitting the interface, kita memberikan kesempatan programmer-programmer selanjutnya untuk menggunakan interface kita untuk hal lain.