



Tutorial 4

Praktikum Pemrograman Berbasis Objek
Asisten IF2210 2023/2024

Gimana Praktikum 3 nya?

Asisten minta maaf ya untuk beberapa kesalahan di soal Praktikum 3



Kesalahan Umum Praktikum 3

- Banyak yang mengecek indeks untuk soal nomor 2. Sebetulnya tinggal try catch aja, biar C++ yang ngurus.

Constructor, Member, dan Method

C++ vs Java

C++ vs Java

C++

1. **ctor**
2. **ctor initialization list**
3. **Operator overloading**
4. Parameter passing:
by value, by ref
5. Templates
6. class
7. Inheritance: public, protected,
private
8. Multi-inheritance
9. **ABC**
10. **Abstract class**
11. Pemanggilan ctor base class
eksplisit tidak bisa

JAVA

1. Tak ada terjemahan
2. Tak ada
3. **Tak ada, fungsi biasa**
4. Hanya by value
5. Kelas generik
6. public class, private class,
protected, package private
7. Inheritance:
hanya ada public
8. “tidak boleh” + interface
9. **Interface**
10. **Abstract class**
11. Ctor superclass bisa diinvokasi:
super() – hanya baris pertama
ctor

C++ vs Java

C++

1. friend
2. main program tidak mungkin dalam kelas
3. Scope: kelas + **file**
4. **namespace**
5. bool
6. string
7. Try-catch: tidak ada *keyword* “finally”
8. Throw objek apapun, walau ada kelas “exception”

JAVA

1. Tidak ada friend
2. main adalah *method* sebuah kelas
3. *Scope*: kelas (*catatan)
4. package
5. Boolean
6. String
7. Try-catch: Ada *keyword* “finally”
8. Throw: harus objek turunan throwable

Contoh: Stack - Header

```
// 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();
};
```

```
// 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 */ }
}
```

Contoh: Stack - Constructor

```
// 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];
}
...
```

```
// 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: Stack - Cctor & Destructor

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

Stack::Stack(const Stack &s) {
    this->capacity = s.capacity;
    this->size = s.size;
    this->data = new int[s.capacity];
    for (int i = 0; i < this->capacity;
i++) {
        this->data[i] = s.data[i];
    }
}

Stack::~Stack() { delete[] this->data;
}
...
```

```
// Stack.java
class Stack {
    ...
    // ini sebenarnya merupakan sebuah
    constructor
    public Stack(Stack s) {
        this.capacity = s.capacity;
        this.size = s.size;
        this.data = new int[s.capacity];
        for (int i = 0; i < this.capacity;
i++) {
            this.data[i] = s.data[i];
        }
    }

    // tidak ada destructor
}
```

Contoh: Stack - Implementasi Fungsi

```
// 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;
}
```

```
// 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

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: Stack - Main Program

```
// stack.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;
}
```

```
// StackMain.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());
    }
}
```

Overloading

Overloading

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

Contoh: Stack - Method Overloading

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

    public void push(int x, int y) {
        if (this.size < this.capacity) {
            this.data[this.size] = x;
            this.size++;
        }
        if (this.size < this.capacity) {
            this.data[this.size] = y;
            this.size++;
        }
    }
    ...
}
```

```
// StackMain.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, 6);
        System.out.println(s2.pop());
        System.out.println(s2.pop());
    }
}
```


Inheritance

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
```

Abstract Class

Abstract Class

- 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: Rekening

```
// Rekening.java
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;
}

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

Contoh: Rekening - Abstract Class

```
// Rekening.java
abstract class Rekening {
    protected String nama;
    protected double saldo;
    protected 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;
}

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

/* update saldo tiap awal bulan*/
abstract void update();
}
```

Contoh: RekeningTabungan

```
// RekeningTabungan.java
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());
    }
}
```

```
// RekeningMain.java
class RekeningMain {
    public static void main(String[] args) {
        RekeningTabungan rt = new
RekeningTabungan("Gopay", 1000000);
        rt.setor(1000000);
        System.out.println(rt.getSaldo());
    }
}
```

Interface

Interface

- Di Java, kita dapat membuat interface: seperti kelas C++ yang:
 - hanya memiliki method
 - **semua** method-nya pure virtual
- Interface != header
- Berbeda dengan inheritance, sebuah kelas bisa mengimplemen 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.

Contoh: IRekening

```
// IRekening.java
interface IRekening {
    public void setor(double uang);
    public void tarik(double uang);
    public void update();

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

```
// Rekening.java
abstract class Rekening implements IRekening {
    ...
}
```

Interface

- Sebuah interface dapat meng-extend interface lain.
- Misalnya, interface IRekening memiliki sifat bisa ditarik dan disetor, sehingga kita buat interface Tarikable dan Setorable

Contoh: IRekening

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

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

    public double hitungBiaya();
}
```

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

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

Interface Segregation Principle

- I pada SOLID
- Suatu kode tidak boleh “dipaksa” untuk bergantung pada metode yang tidak digunakannya.



Contoh: Interface Segregation (Bad)

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

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

Contoh: Interface Segregation (Good)

```
interface Breathing {  
    public void breathe();  
}  
  
interface Moveable {  
    public void move();  
}  
  
interface Reproduce {  
    public void haveChild();  
}  
  
interface Growable {  
    public void grow();  
}  
  
interface Eatable {  
    public void eat();  
}
```

```
class AnaerobicBacteria implements Moveable,  
Reproduce, Growable, Eatable {  
    public void haveChild();  
    public void move();  
    public void grow();  
    public void eat();  
}  
  
class AerobicBacteria implements Moveable,  
Reproduce, Growable, Eatable, Breathing {  
    public void haveChild();  
    public void move();  
    public void grow();  
    public void eat();  
    public void breathe();  
}
```


Dokumentasi Java

- <https://docs.oracle.com/javase/8/docs/api/>



Sekian.

Ditunggu praktikum dan tutorial selanjutnya.