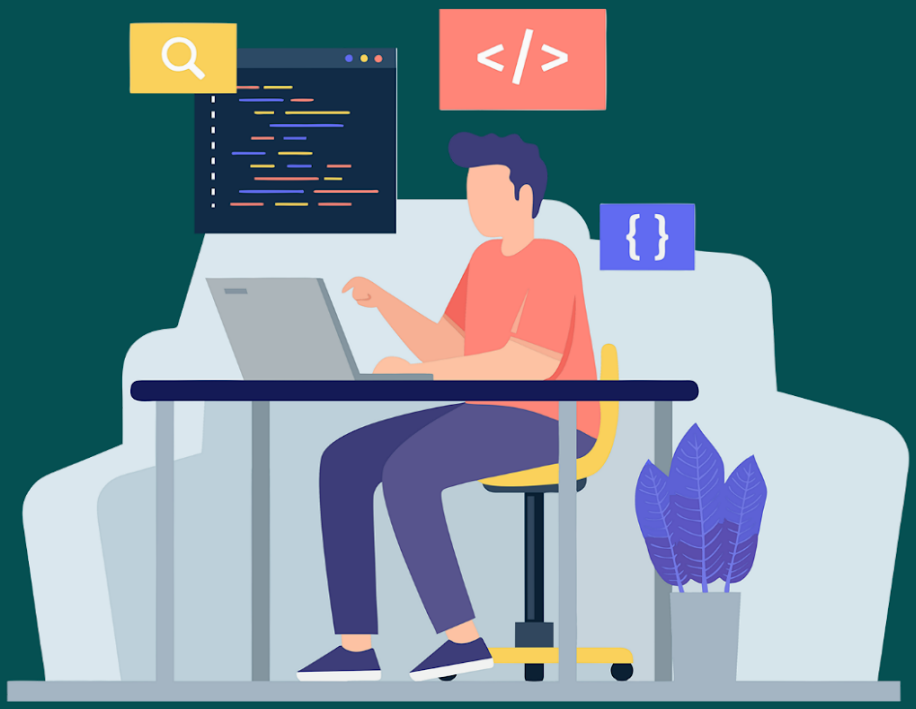
Modul Praktikum
Desain dan Pemrograman Berorientasi Objek

PENDAHULUAN KELAS DAN ENKAPSULASI



TIM ASISTEN PEMROGRAMAN
ANGKATAN 12

Departemen Pendidikan Ilmu Komputer
Fakultas Pendidikan Matematika dan Ilmu Pengetahuan Alam
Universitas Pendidikan Indonesia
2022

Pemrograman Berorientasi Objek

Pemrograman Berorientasi Objek (*Object-Oriented Programming / OOP*) merupakan suatu paradigma pemrograman yang berbasis konsep objek, dimana satu objek berisikan berbagai macam data atau variabel —dikenal sebagai ***attribute***— serta prosedur dan fungsi —dikenal sebagai ***method***. Kumpulan atribut dan metode tersebut dibungkus ke dalam suatu kelas atau objek, dan satu objek bisa berkomunikasi dengan objek lainnya.

Istilah	Keterangan
Class	Realisasi atau "cetak biru" dari suatu objek yang akan dibuat, terdiri atas **properti** dan **metode**.
Inner Class	Kelas yang didefinisikan di dalam suatu kelas. Biasanya, kelas ini digunakan secara khusus untuk satu kelas saja, sehingga tidak dibuat di kelas yang berbeda.
Property / Attribute	Nilai yang dimiliki oleh suatu kelas. Contoh: Class "Manusia" mempunyai properti **nama** dan **jenis kelamin**.
Method / Behavior / Function	Aksi atau tindakan yang bisa dilakukan oleh suatu kelas. Contoh: Class "Manusia" bisa melakukan **makan** dan **tidur**.
Encapsulation	Proses "membungkus" atribut dan metode dari suatu kelas agar tidak bisa diakses sembarangan oleh kelas lain.
Instance / Instantiation	Pembuatan suatu objek (contoh/sample) dalam program menggunakan kelas yang tersedia.
Constructor	Metode spesifik yang pasti ada (default) dalam sebuah kelas dan otomatis terpanggil ketika dilakukan instansiasi pada kelas. Digunakan untuk memberikan inisiasi/nilai awal pada properti kelas untuk suatu objek dan meminta alokasi memori untuk objek. • Pada **C++** dan **Java**, konstruktor didefinisikan sebagai metode yang diberi nama persis sesuai nama kelasnya.

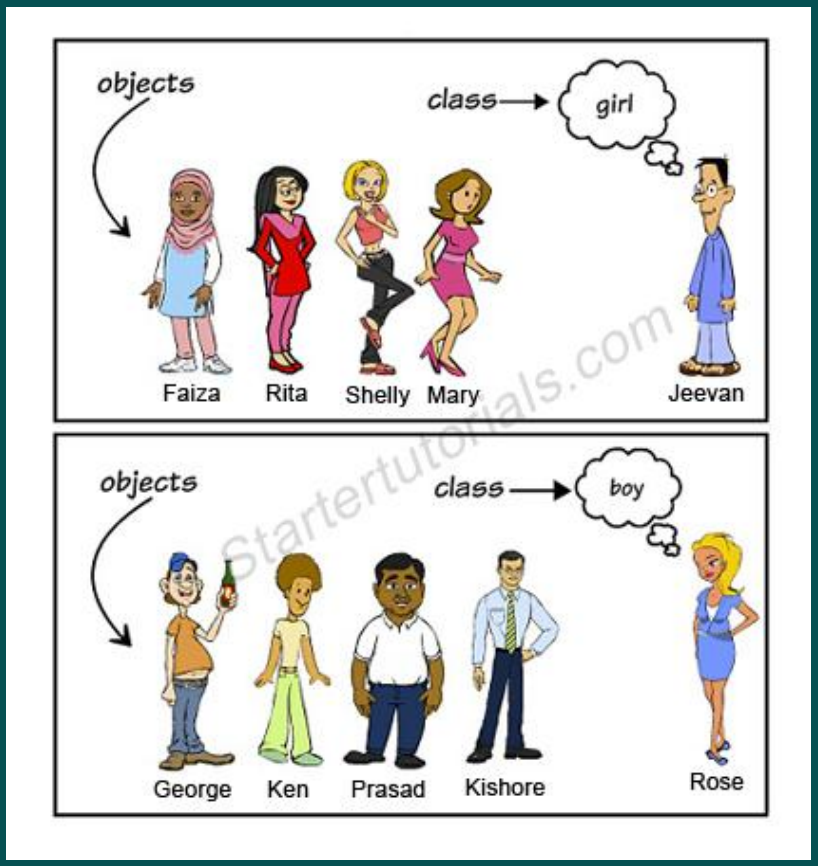
	• Pada **PHP**, konstruktor merupakan metode yang diberi nama "__construct". • Pada **Python**, konstruktor merupakan metode yang diberi nama "__init__".
Destructor	Metode spesifik yang dilakukan ketika suatu objek akan dibuang atau dihapus dari program. • Pada **C++**, destruktor didefinisikan sebagai metode yang diberi nama persis sesuai nama kelasnya, dengan tambahan simbol "~" di awal. • Pada **PHP**, destruktor didefinisikan sebagai metode yang diberi nama "__destruct". • Pada **Java** dan **Python**, destruktor tidak perlu didefinisikan secara eksplisit karena keduanya sudah mempunyai fitur untuk membersihkan sampah memori secara otomatis (*garbage collection*). Meski begitu, destruktor masih bisa ditambahkan secara manual melalui metode "finalize" untuk Java dan "__del__" untuk Python.

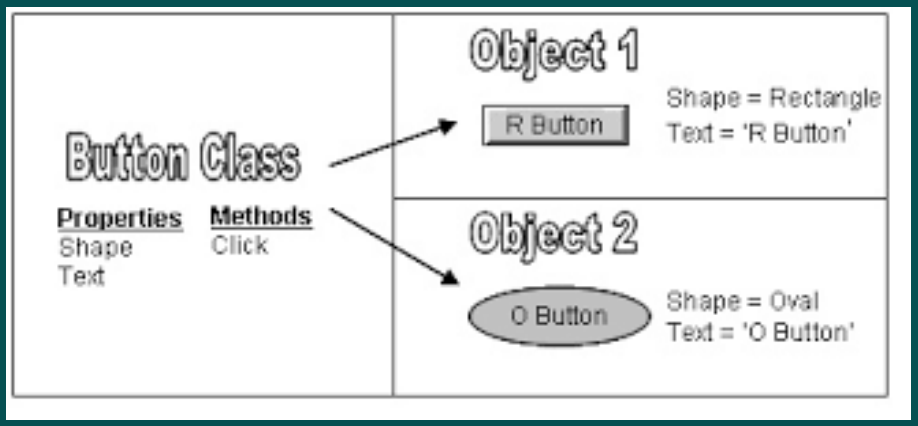
Class, Object & Instance

Class

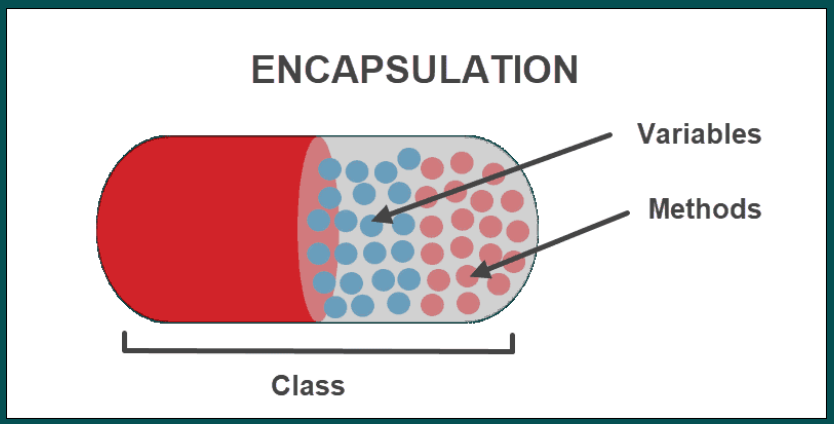
Object

Instance adalah object yang dibuat dari class.





Encapsulation



Example 2.
class : Brain
property : private mind
method : setMind( ), getMind( )
If our mind is public, people can read our thoughts directly.
But if it's private, people can know our thoughts only if they ask.

Access Modifier

Access modifier di dalam Object Oriented Programming (OOP) akan menentukan apakah kelas lain dapat menggunakan field atau meng-invoke methods dari suatu kelas. Ada beberapa macam access modifier yang dapat digunakan yaitu private, default, protected, dan public.

- Private : akses hanya di dalam class
- Default : bisa diakses selama masih dalam satu package
- Protected : bisa diakses selama masih dalam satu package ATAU bisa diakses dari luar package asalkan diakses oleh class turunan
- Public : bisa diakses dari manapun bahkan package yang berbeda

Modifier	Class	Package	Subclass	World
Private	Yes	No	No	No
Default	Yes	Yes	No	No
Protected	Yes	Yes	Yes	No
Public	Yes	Yes	Yes	Yes

Non Access Modifier

- Static : Deklarasi variabel atau metode yang berdiri sendiri, tanpa perlu instance dari suatu kelas.
- Final : Variabel hanya bisa diinisiasi sekali
- Abstract : Kelas yang mendefinisikan method-nya secara umum.
- Synchronized : Menangani kasus *multiple thread*.

Implementasi Kelas dan Enkapsulasi: C++

Berikut adalah contoh implementasi kelas dan enkapsulasi pada bahasa pemrograman C++. Kelas ini akan membuat suatu objek “Human” yang mempunyai atribut nama dan jenis kelamin, serta metode makan dan tidur.

Human.cpp

```cpp
// Import library.
#include <iostream>
#include <string>

// Using standard namespace.
using namespace std;

// Class declaration. For C++, the first letter doesn't have to be
// capitalized, but it's better so we can distinguish it with the other types.
class Human
{
    // Private attributes.
    private:
        string name;
        char gender;

    // Public methods.
    public:

        /* Constructors. */

        // Constructor. Take note that it doesn't have any return type.
        Human()
        {
            // Set name and gender to default value.
            this->name = "";
            this->gender = '-';
        }

        // Another constructor with parameter. This one will be called if the
        // object is instanced with the given parameter.
        Human(string name, char gender)
        {
            // "This" object's attributes will be set with the given parameter attributes.
            this->name = name;
            this->gender = gender;
        }
```

```cpp

        /* Getter and Setter. */

        // Get name.
        string get_name()
        {
            return this->name;
        }

        // Set name.
        void set_name(string name)
        {
            this->name = name;
        }

        // Get gender.
        char get_gender()
        {
            return this->gender;
        }

        // Set gender.
        void set_gender(char gender)
        {
            this->gender = gender;
        }

```

```
        /* Public methods, or in this case if you prefer, "behaviors". */

        // Human eats.
        void eat()
        {
            cout << this->name << " is eating!" << '\n';
        }

        // Human sleeps.
        void sleep()
        {
            cout << this->name << " is sleeping!" << '\n';
        }

        /* Destructors. */

        // Default destructor. Leave it blank for now.
        ~Human()
        {

        }
};

```

Main.cpp

```
// Import library and file.
#include <bits/stdc++.h>
#include "Human.cpp"

// Using standard namespace.
using namespace std;

int main()
{
    /* Default code. */

    // [Method #1]: Object instantiation using default constructor.
    Human rain;

    // The object's attributes, however, should be set manually.
    rain.set_name("Rain");
    rain.set_gender('L');

    // [Method #2]: Object instantiation using constructor with parameter.
    Human techi("Techi", 'P');

    /* Experimental code. */

    // Let's try again, this time with input and list.
    int i, n = 0;
    string name;
    char gender;

    // You have taken Data Structures, haven't you? Let's try this convenient,
    // premade Linked List library!
    list<Human> llist;

```

```
    cin >> n;
    for(i = 0; i < n; i++)
    {
        // Temporary object.
        Human temp;

        // Attribute input.
        cin >> name >> gender;

        // Assign input to the temporary object.
        temp.set_name(name);
        temp.set_gender(gender);

        // Insert temporary object into the list. See? Even the linked list is
        // an object itself.
        llist.push_back(temp);
    }

    /* Output. */

    // These code will work because we "communicate" with the private attributes
    // through its public methods.
    cout << '\n' << "Automatic output : " << '\n';
    cout << "The first human's name is " << rain.get_name() << '\n';
    cout << "The first human's gender is " << rain.get_gender() << "\n\n";
    cout << "The second human's name is " << techi.get_name() << '\n';
    cout << "The second human's gender is " << techi.get_gender() << "\n\n";

    // But these code won't because we access their private attributes
    // without any permissions.
    // * Remove the comment to test it.
    // cout << "The first human's gender is " << rain.gender << '\n';
    // cout << "The second's name is " << techi.name << '\n';

    // Meanwhile, this is an example to print all elements of a list. Explore
    // and discover more code by yourself!
    cout << "Iteration output : " << '\n';
    i = 0;
    for(list<Human>::iterator it = llist.begin(); it != llist.end(); it++)
    {
        cout << (i + 1) << ". " << it->get_name() << " | " << it->get_gender() << '\n';
        i++;
    }

    return 0;
}
```

Implementasi Kelas dan Enkapsulasi: Java

Berikut adalah contoh implementasi kelas dan enkapsulasi pada bahasa pemrograman Java. Perhatikan setiap detailnya, karena ada beberapa aturan yang harus diikuti pada bahasa ini.

Human.java

```java
// Class declaration. For Java, the first letter MUST BE capitalized.
public class Human
{
    // Private attributes.
    private String name;
    private char gender;

    /* Constructors. */

    // Constructor. Take note that it doesn't have any return type.
    public Human()
    {
        this.name = "";
        this.gender = '-';
    }

    // Another constructor with parameter.
    public Human(String name, char gender)
    {
        this.name = name;
        this.gender = gender;
    }

    /* Getter and Setter. */

    // Get name.
    public String getName()
    {
        return this.name;
    }

    // Set name.
    public void setName(String name)
    {
        this.name = name;
    }

    // Get gender.
    public char getGender()
    {
        return this.gender;
    }

    // Set gender.
    public void setGender(char gender)
    {
        this.gender = gender;
    }

    /* Public methods, or in this case if you prefer, "behaviors". */

    // Human eats.
    public void eat()
    {
        System.out.println(this.name + " is eating!");
    }

    // Human sleeps.
    public void sleep()
    {
        System.out.println(this.name + " is sleeping!");
    }

    // Nahh, Java doesn't need a destructor.
}
```

Main.java

```java
// Import library.
import java.util.Scanner;
import java.util.ArrayList;

public class Main
{
    Run | Debug
    public static void main(String[] args)
    {
        // [Method #1]: Object instantiation using default constructor and getter-setter.
        Human rain = new Human();
        rain.setName(name: "Rain");
        rain.setGender(gender: 'L');

        // [Method #2]: Object instantiation using constructor with parameter.
        Human techi = new Human(name: "Techi", gender: 'P');

        // Let's try another method as usual.
        int i, n = 0;
        String name;
        char gender;

        // Another useful data structure you can use.
        ArrayList<Human> list = new ArrayList<>();

        // Even the input have to be "communicated" through an object.
        // * Ignore the warning, the right solution is more confusing.
        Scanner sc = new Scanner(System.in);    Resource leak: 'sc' is never closed

        // This is called Exception Handling. In a nutshell, it's something like
        // "if there is an error inside the 'try' code, break and move to the
        // 'catch' code". You will learn it later in the class.
        try
        {
            n = sc.nextInt();
        }
        catch(Exception e)
        {
            System.out.println(x: "The input is not an integer!");
        }

        // Iteration, et cetera et cetera.
        for(i = 0; i < n; i++)
        {
            // Why don't we use the same 'try-catch' code here...?
            name = sc.next();
            gender = sc.next().charAt(index: 0);

            Human temp = new Human();
            temp.setName(name); temp.setGender(gender);
            list.add(temp);
        }

        /* Output. */

        System.out.println(x: "\n Automatic output :");
        System.out.println("The first human's name is " + rain.getName());
        System.out.println(rain.getName() + "'s gender is " + rain.getGender());
        System.out.println("The second human's name is " + techi.getName());
        System.out.println(techi.getName() + "'s gender is " + techi.getGender());
        System.out.println();

```

```
        // How to iterate through ArrayList. Seems easy, right?
        // Same as a regular array, hence it's called 'ArrayList'.
        System.out.println(x: "Iteration output : ");
        for(i = 0; i < list.size(); i++)
        {
            System.out.println(Integer.toString(i + 1) + ". "
                + list.get(i).getName() + " | " + list.get(i).getGender());
        }

        // Close scanner. What the heck is this? well, search it on Internet.
        sc.close();
    }
}

```

Implementasi Kelas dan Enkapsulasi: Python

Berikut adalah contoh implementasi kelas dan enkapsulasi pada bahasa pemrograman Python. Jumlah baris kode lebih pendek, tetapi perlu berhati-hati karena *debugging* dan *backtracking* pada bahasa ini cukup membingungkan.

Human.py

```python
# Class declaration.
class Human:

    # Private attributes. No data type, no instantiation, just plain
    # declaration. Use double underscores (__) for private attribute, single
    # (_) for protected, and none for public.
    __name = ""
    __gender = ''

    # Python uses single constructor only. Put a default value so it will use
    # that in case you don't set the arguments.
    def __init__(self, name = "", gender = ''):
        self.__name = name
        self.__gender = gender

    # Getter and Setter. #

    # Get name.
    def get_name(self):
        return self.__name

    # Set name.
    def set_name(self, name):
        self.__name = name

    # Get gender.
    def get_gender(self):
        return self.__gender

    # Set gender.
    def set_gender(self, gender):
        self.__gender = gender

```

```
    # Public methods, or in this case if you prefer, "behaviors". #

    # Human eats.
    def eat(self):
        print(self.__name, "is eating!")

    # Human sleeps.
    def sleep(self):
        print(self.__name, "is sleeping!")


```

main.py

```
# Import class file.
from Human import Human

# Instantiation without any arguments, it will use the default value.
rain = Human()
rain.set_name("Rain")
rain.set_gender('L')

# Instantiation with arguments, it will override the default value.
techi = Human("Techi", 'P')

# Another effort, this time using array.
humans = []

# Iteration, but this one is really simple.
n = int(input())
for i in range (n):
    name = str(input())
    gender = input()[0]

    humans.append(Human(name, gender))

# Output. #

print()
print("Automatic output : ")
print("The first human's name is", rain.get_name())
print("Now,", end = ' ')
rain.eat()
print("The second human's name is", techi.get_name())
print("Now,", end = ' ')
techi.sleep()

```

```
# Use foreach-equivalent method to traverse into an array.
i = 0
print("Iteration output : ")
for human in humans:
    print(str(i + 1) + ".", human.get_name(), "|", human.get_gender())
    i += 1

```

Implementasi Kelas dan Enkapsulasi: PHP

Buat file di dalam folder htdocs XAMPP.

Human.php > ...

```php
<?php

class Human {
    private $name = '';
    private $gender = '';

    public function __construct($name = '', $gender = '')
    {
        $this->name = $name;
        $this->gender = $gender;
    }

    public function setName($name) {
        $this->name = $name;
    }

    public function getName() {
        return $this->name;
    }

    public function setGender($gender) {
        $this->gender = $gender;
    }

    public function getGender() {
        return $this->gender;
    }

    public function eat() {
        return "$this->name is eating!";
    }

    public function sleep() {
        return "$this->name is sleeping zzz";
    }
}

?>
```

main.php > ...

```php
<?php
require ('Human.php');

$human1 = new Human('Sekar', 'Female');
echo $human1->getName();
echo $human1->getGender();
echo '<hr>';

$human1->setName('Sekar MK');
echo $human1->getName();

$human2 = new Human('Satria', 'Male');
echo '<hr>';

echo 'In the night ' . $human1->eat() . '<br>';
echo 'In the afternoon ' . $human2->sleep() . '<br>';

?>
```

Latihan

Buatlah program berbasis OOP menggunakan bahasa pemrograman C++, Java, Python, dan PHP yang menampilkan informasi daftar mahasiswa (sekumpulan objek mahasiswa) dan memiliki fitur menambah, mengubah, dan menghapus data. Setiap mahasiswa memiliki data nama, NIM, program studi, fakultas, dan foto profil (khusus bahasa PHP).

Note.

- Boleh menambahkan properti/atribut baru
- Tampilkan data selengkap-lengkapnya dalam bentuk list/tabel
- Program dikumpulkan pada repository GitHub yang dibuat public dengan nama "LATIHAN1DPBO2023"
  - Hanya program pada branch Main yang akan dinilai dan diperiksa
  - Jika waktu pengumpulan sudah habis dan ingin mengupdate kode program, update pada branch lain karena mengupdate branch Main setelah waktu pengumpulan terlewat maka program tidak akan dinilai
- Struktur folder

  - CPP
    - program
    - screenshot

  + Java
  + Python
  + PHP

  README.md
- File README berisi desain program, penjelasan alur, dan dokumentasi saat program dijalankan (screenshot/screen record, pilih salah satu bahasa sebagai contoh)
- Submit link repository pada form berikut: https://forms.gle/rvb1hKxbQVuYNbhKA

Penutup

Terima kasih atas kerja sama seluruh pihak yang membantu dalam penyusunan modul ini, semoga apa yang telah didapatkan bisa bermanfaat di masa yang akan datang.

Daftar Pustaka

Sukamto, Rosa A. (2018). *Pemrograman Berorientasi Objek*. Bandung: Modula.

Asisten Pemrograman 11. (2022). *Modul Desain dan Pemrograman Berorientasi Objek*.