

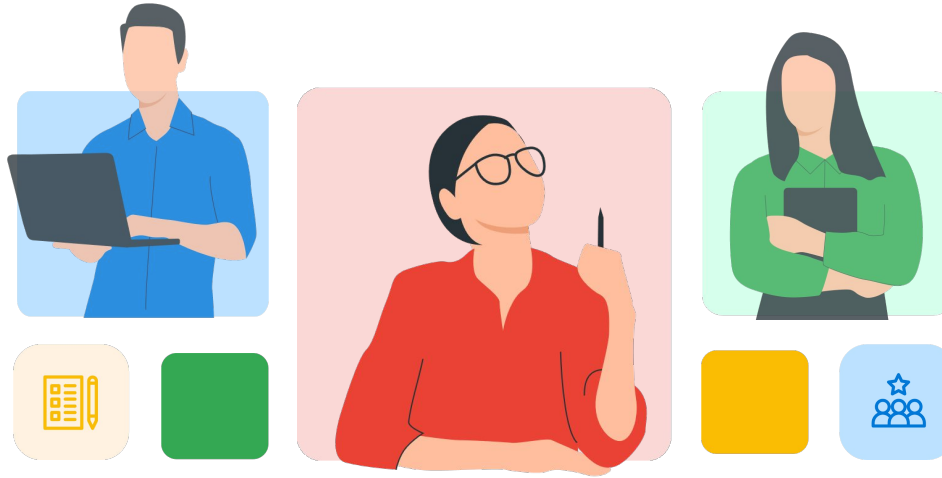


79 Bootcamp Digitalization






OOP






1. Pengenalan OOP




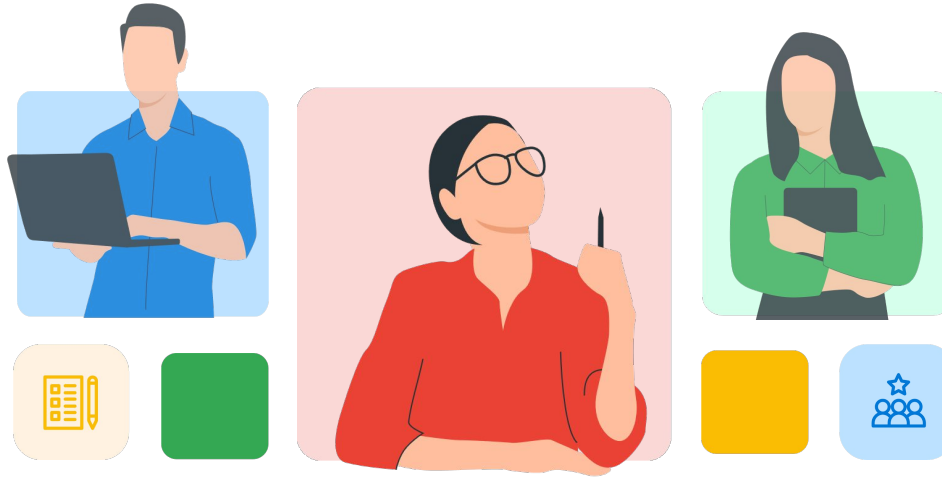
Mengapa Object Oriented Programming? OOP (Object Oriented Programming) masih merupakan salah satu paradigma atau teknik pemrograman yang populer dalam pengembangan aplikasi.

Dengan paradigma OOP kita dapat dengan mudah memvisualisasikan kode karena OOP sendiri mirip dengan skenario kehidupan nyata.



Dalam penerapan OOP kita menggabungkan kumpulan-kumpulan fungsi atau atribut yang memiliki kesamaan dalam sebuah unit yang kita sebut sebagai objek.





2. Class & Object

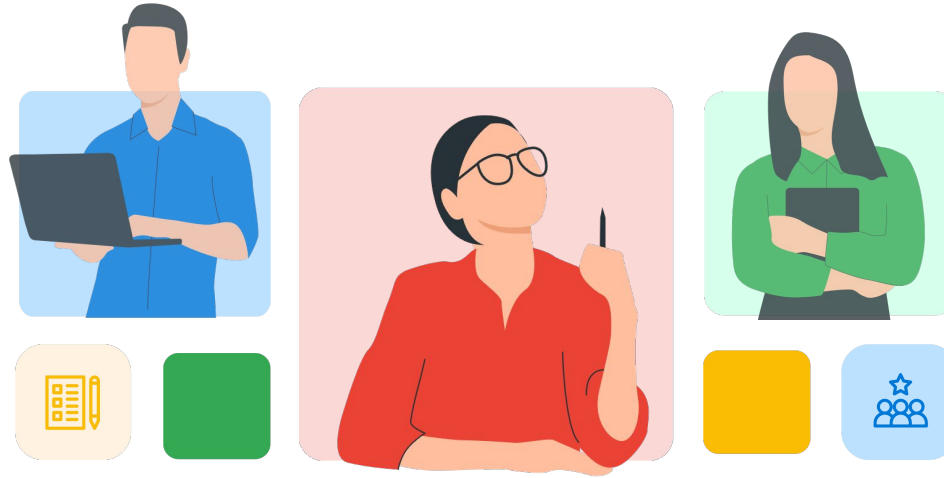


Class & Object

Class merupakan sebuah blueprint yang dapat dikembangkan untuk membuat sebuah objek. Blueprint ini merupakan sebuah template yang di dalamnya menjelaskan seperti apa perilaku dari objek itu (berupa properti ataupun function).

Person	
Properties/Attribute	Function
Name	greeting()
Address	
Age	



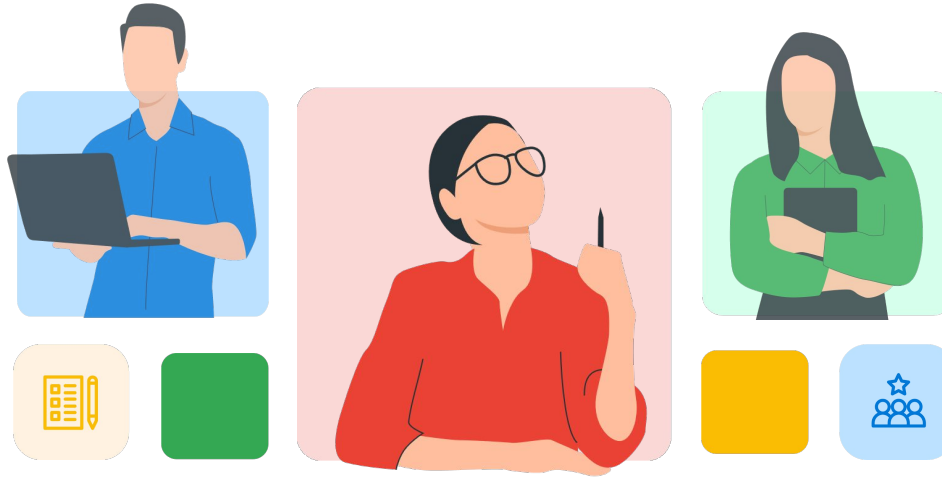


3. Pilar Penting OOP

Pilar Penting OOP



- Inheritance
- Encapsulation
- Abstraction
- Polymorphism



4. Inheritance



Inheritance

Kita mulai dengan membahas Inheritance atau yang akan kita sering dengar dengan istilah pewarisan. Dalam object oriented programming, inheritance merupakan salah satu konsep penting. Kenapa? Karena kita bisa meminimalisir penulisan berulang pada fungsi, properti, dan variable.

Kok bisa? Inheritance memungkinkan kita untuk mendefinisikan sebuah class (induk) ke class baru (anak) dan memberi kita kesempatan untuk menggunakan member dari class yang diwariskan tersebut. Inheritance dapat didefinisikan juga sebagai proses di mana suatu objek memperoleh sifat dan perilaku dari objek lainnya.

Ketika ingin membuat class dengan fungsi yang sudah tersedia pada class lain, kita tidak perlu lagi menulis ulang kode tersebut di dalam class yang kita buat. Cukup dengan mewarisi class tersebut maka kita bisa langsung mengaksesnya.

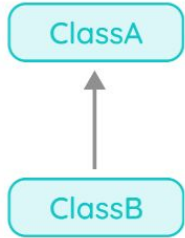
Ilustrasi

Teacher	
Properties/Attribute	Function
Name	greeting()
Address	teaching()
Age	
Subject	
Salary	
Programmer	
Properties/Attribute	Function
Name	greeting()
Address	programming()
Age	
ProgrammingLanguage	
Salary	
Driver	
Properties/Attribute	Function
Name	greeting()
Address	driving()
Age	
License Type	
Salary	

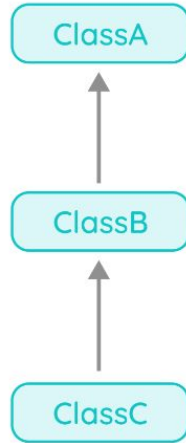


Jenis-jenis inheritance

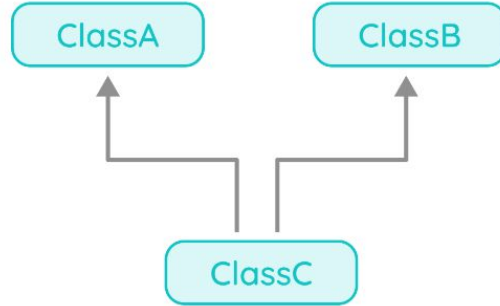
Single Inheritance



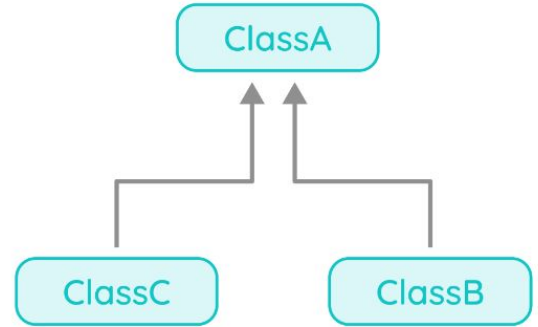
Multilevel Inheritance

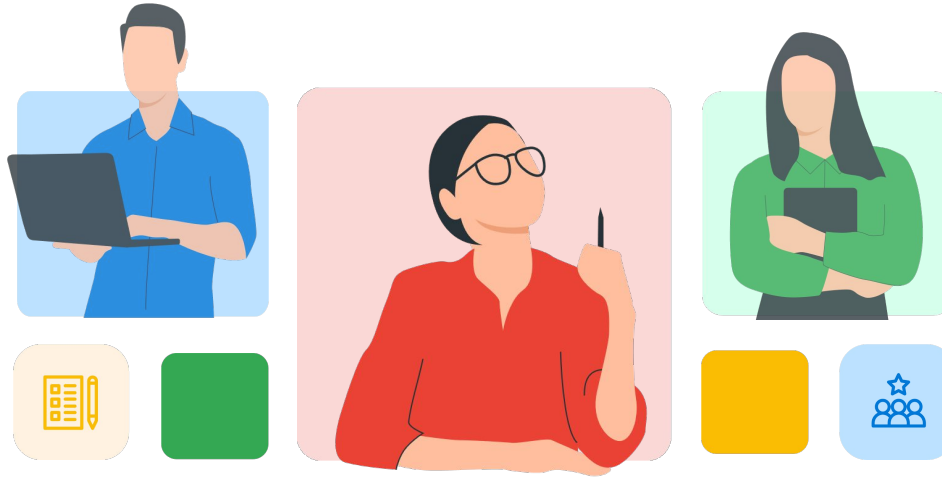


Multiple Inheritance



Hierarchical Inheritance





5. Encapsulation



Encapsulation



Encapsulation merupakan proses di mana sebuah penanganan data ditempatkan di dalam wadah tunggal yang disebut sebagai class. Saat menggunakan encapsulation, data dapat diisolasi dan tidak dapat diakses langsung dari luar. Dengan begini, kita cukup menggunakan data tersebut tanpa harus tahu bagaimana proses yang terjadi sampai data tersebut bisa digunakan.

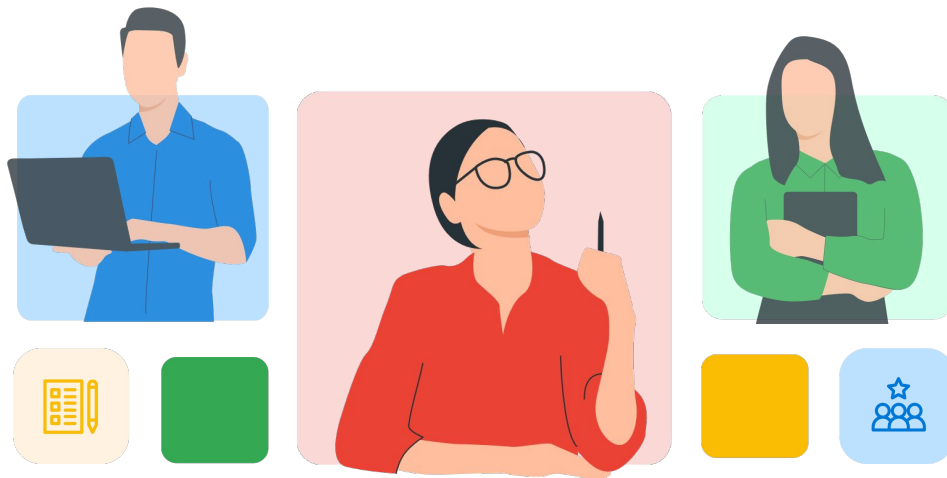
Ilustrasi Encapsulation

Dari contoh Potongan kode berikut, apa alasan kita untuk menggunakan encapsulation? Karena kita akan lebih leluasa dalam melakukan perubahan nilai tanpa harus mengakses propertinya secara langsung, cukup gunakan fungsi setter yang tersedia.

Selain itu, kode yang berada di dalam aplikasi diringkas berdasarkan bagiannya sehingga dapat dengan mudah diubah tanpa mempengaruhi bagian lainnya. Kode yang sudah dienkapsulasi tersebut dapat digunakan kembali di seluruh bagian aplikasi.

Yang perlu diketahui, encapsulation **bukan menyembunyikan** sebuah **data**. Tetapi, encapsulation yang **menyebabkan data** tersebut **tersembunyi**.

```
public class HistoryData {  
    private List data = new ArrayList();  
  
    public void setHistory(List historyData) {  
        Response response = /* Berisikan banyak Kodingan Logic */  
        data.clear();  
        data.addAll(response.data);  
    }  
  
    public List getHistory() {  
        return data;  
    }  
}
```



6. Abstraction



Abstraction



Abstraksi adalah salah satu konsep utama dalam pemrograman berorientasi objek (OOP). Dalam Java, abstraksi memungkinkan pengembang untuk membuat kelas abstrak yang tidak dapat diinstansiasi, tetapi dapat digunakan sebagai kerangka dasar untuk kelas turunan konkret.

Abstraksi adalah proses mengidentifikasi dan menentukan aspek penting dari suatu objek dan mengabaikan detail yang tidak relevan. Dalam pemrograman, abstraksi memungkinkan kita untuk membuat kelas abstrak yang berfungsi sebagai cetak biru atau kerangka dasar bagi kelas-kelas turunannya. Kelas abstrak tidak dapat diinstansiasi, tetapi dapat berisi metode abstrak (tidak memiliki implementasi) dan metode konkret (memiliki implementasi).

Abstract Class

Class abstrak adalah class yang tidak dapat diinstansiasi secara langsung, tetapi hanya dapat diwarisi oleh subclass. Class abstrak digunakan untuk mendefinisikan metode-metode abstrak (tidak memiliki implementasi) yang harus diimplementasikan oleh subclassnya.

Untuk membuat class abstrak, kita dapat menggunakan kata kunci "abstract" sebelum deklarasi class.

Subclass yang mewarisi class abstrak harus mengimplementasikan semua metode abstrak yang didefinisikan oleh superclassnya.

```
abstract class Bentuk {  
    // Metode abstrak  
    public abstract void hitungLuas();  
}
```

```
class PersegiPanjang extends Bentuk {  
    public void hitungLuas() {  
        // Implementasi metode hitungLuas() untuk PersegiPanjang  
    }  
}
```

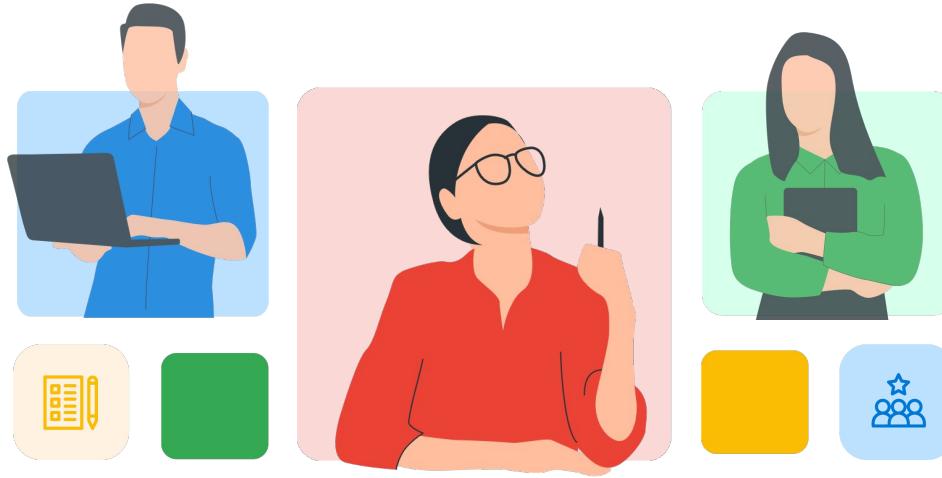
Interface

Interface adalah kumpulan metode-metode abstrak yang dapat diimplementasikan oleh class-class lain.

Interface menyediakan kontrak atau spesifikasi tentang perilaku yang harus diikuti oleh class-class yang mengimplementasikannya.

```
interface DoorInterface {  
    void openDoor();  
    void closeDorr();  
}
```

```
public class Mobil implements DoorInterface {  
  
    public void openDoor() {  
        // Implementasi metode openDoor() untuk Mobil  
    }  
    public void closeDoor() {  
        // Implementasi metode closeDoor() untuk Mobil  
    }  
}
```



7. Polymorphism





Polymorphism



Polimorfisme adalah kemampuan sebuah objek untuk mengambil banyak bentuk. Dalam konteks Java, polimorfisme memungkinkan objek untuk merespons metode yang sama dengan cara yang berbeda.

Polimorfisme memungkinkan penggunaan metode yang sama untuk berbagai jenis objek yang berbeda.

Polymorphism merupakan kemampuan objek, variabel, atau fungsi yang dapat memiliki berbagai bentuk. Secara umum polymorphism dalam OOP terjadi ketika suatu SuperClass direferensikan ke dalam SubClass. Alhasil kita dapat mengembangkan sebuah program secara umum, bukan spesifik



Ilustrasi Polymorphism

```
abstract class Person {  
    void activity();  
}  
  
class Father extends Person {  
    void activity() {  
        System.out.println("Mendidik Anak");  
    }  
}  
  
class Suami extends Person {  
    void activity() {  
        System.out.println("Mencari Nafkah");  
    }  
}  
  
class Employee extends Person {  
    void activity() {  
        System.out.println("Bekerja Menyelesaikan Tugas.");  
    }  
}
```

```
class Main {  
    public static void main(String[] args) {  
        Person person = new Father();  
        person.activity();           // "Mendidik Anak"  
  
        Person person2 = new Suami();  
        person2.activity();          // "Mencari Nafkah"  
  
        Person person3 = new Employee();  
        person3.activity();          // "Bekerja Menyelesaikan Tugas."  
    }  
}
```



**Tujuh
Sembilan**
Always Improving You

Sekian, Terima Kasih