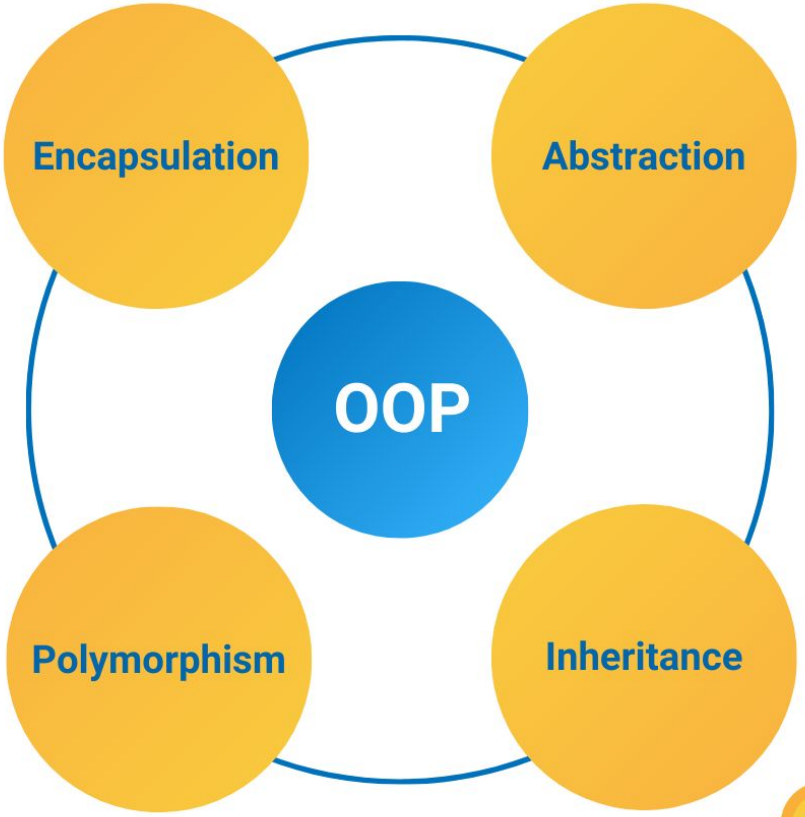
Object Oriented Programming Characteristic

Ratna Mufidah, S.Kom., M.Kom.



https://codingnomads.co/blog/what-is-object-oriented-programming-oop-concepts-in-java

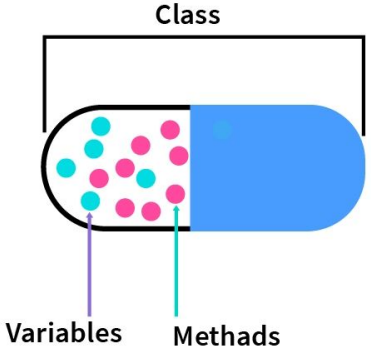
Encapsulation

A process of binding data (properties, variables) and operations or functions (methods) that works on data together in a single construct.

CLASS

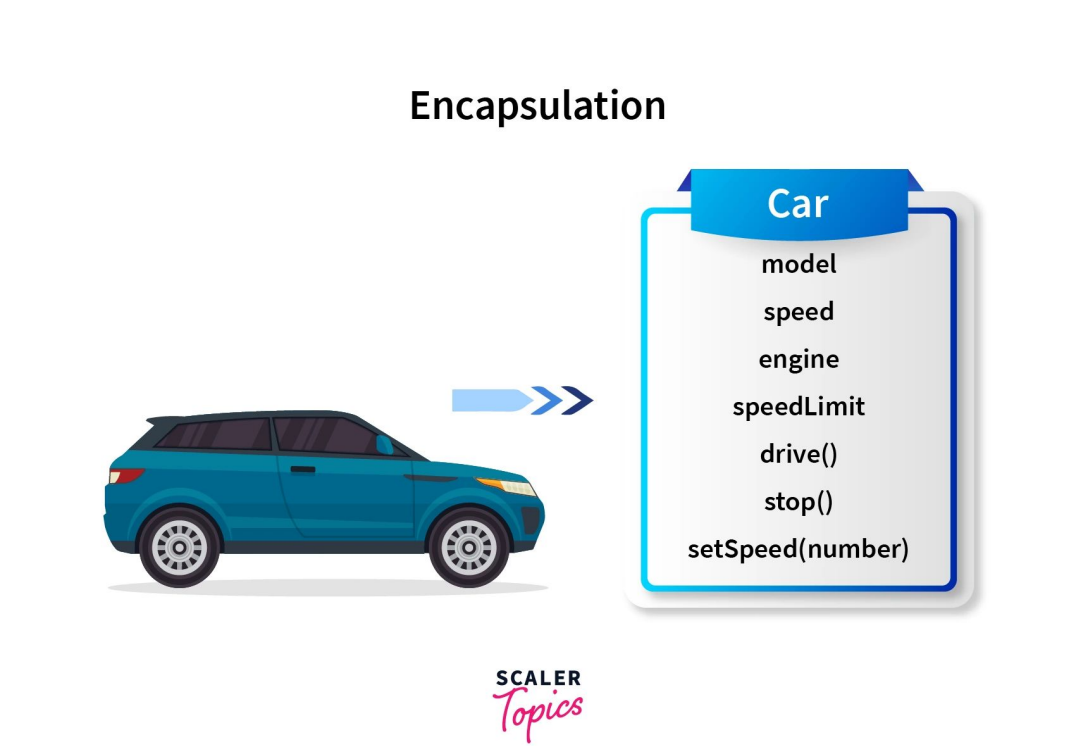
Encapsulation

```
Classs
{
    data members
        +
  methods (behavior)
}
```



SCALER Topics

Encapsulation



Encapsulation

- The mechanism **hides a process and data** in the system to **avoid interference** and **simplify** the use of the process itself.
- **Class access levels** (public, protected, and private) are **implementations** of the **encapsulation** concept.
- Data encapsulation can be done by:
  1. declaring the instance variable as private;
  2. declaring a public method for accessing these variables.

Encapsulation

- Encapsulation guarantees that no programmers other than those with access to the class can interfere with it. This will help the application prevent bugs or mistakes, such as incorrect object usage.

Encapsulation

class acces level

- **Public**
  The class, interface, method, constructors, attributes, or inner classes can be accessed or implemented by other classes anywhere.
- **Protected**
  The method, constructor, attribute, or inner class can be accessed and implemented by other existing classes in one package or its derivatives.

Encapsulation
class acces level

- **Default**
  The class, interface, method, constructor, attribute, or inner class can be accessed or implemented by other classes that are in one package (the derived classes and implementations must be in one package).

Encapsulation
class acces level

- **Private**
  The methods, constructors, attributes, and inner classes cannot be accessed at all by other classes and cannot be derived. An attribute is declared private under the following conditions:
  1. If other classes do not require these attributes,
  2. protects an attribute from the possibility of its value being changed by another method from another class.

Encapsulation

```java
package latihan;

public class Person {
	// private field
	private int age;

	// getter method
	public int getAge() {
	  return age;
	}

	// setter method
	public void setAge(int age) {
	  this.age = age;
	}

}
```

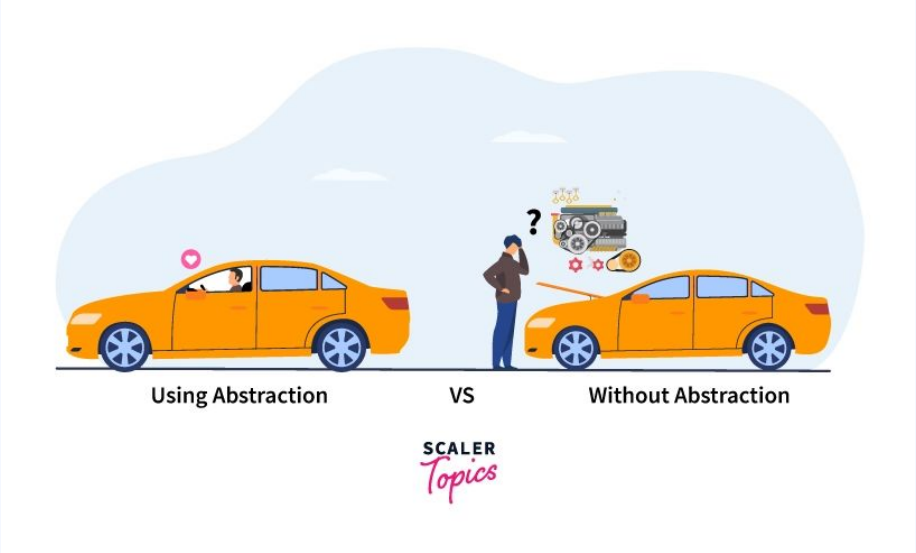
Person.java

```java
package latihan;

public class PersonMain {

	public static void main(String[] args) {
		// TODO Auto-generated method stub
		// create an object of Person
		Person p1 = new Person();

		// change age using setter
		p1.setAge(24);

		// access age using getter
		System.out.println("My age is " + p1.getAge());

	}

}
```

PersonMain.java

Abstraction

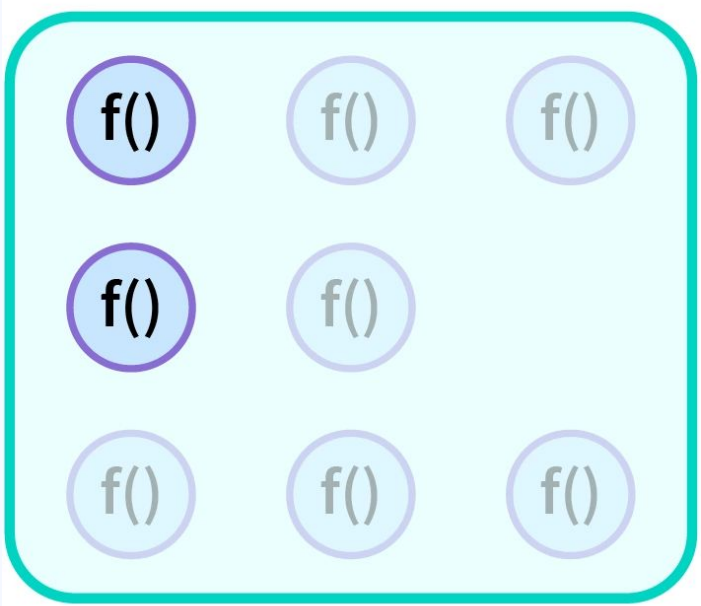
The act of hiding unnecessary details, and only showing essential information to the user. For example, a driver sees a car as a car, rather than as its individual components.



Abstraction

Abstraction is **a way to hide complexities** and give a simple user interface to the user.

In abstraction, **we just know what things do rather than how they do**. All the complexities are hidden and only a simple user interface is provided to work.



SCALER Topics

Abstraction

The abstract keyword is a non-access modifier, used for classes and methods:

1. **Abstract class**: is a restricted class that cannot be used to create objects (to access it, it must be inherited from another class).

2. **Abstract method**: can only be used in an abstract class, and it does not have a body. The body is provided by the subclass (inherited from).

Abstraction

Vehicle.java

```java
package latihan;

abstract class Vehicle {
    // Abstract method - the method below does not have a "method body"
    public abstract int numberOfWheels();

    // Below is a standard method with a body
    public void start() {
        System.out.println("Starting vehicle.");
    }
}
```

Abstract classes are **restricted classes** that cannot be used to create objects. The instance variables of an abstract class can be accessed when it is inherited from another class.

Encapsulation vs Abstraction

Encapsulation	Abstraction
Focus on grouping the variables/attributes and methods of the object together inside a single unit.	Focus on hiding the complex methods and only showing the essential things to the user.
It makes code more modular and easy to understand	It makes applications easy to use by hiding underlying complex working.
Provide security to the variables/attributes and methods by deciding who can access them.	Provide security to the application by hiding the working part from the user.
Hding the information.	Hiding the implementation.

Inheritance

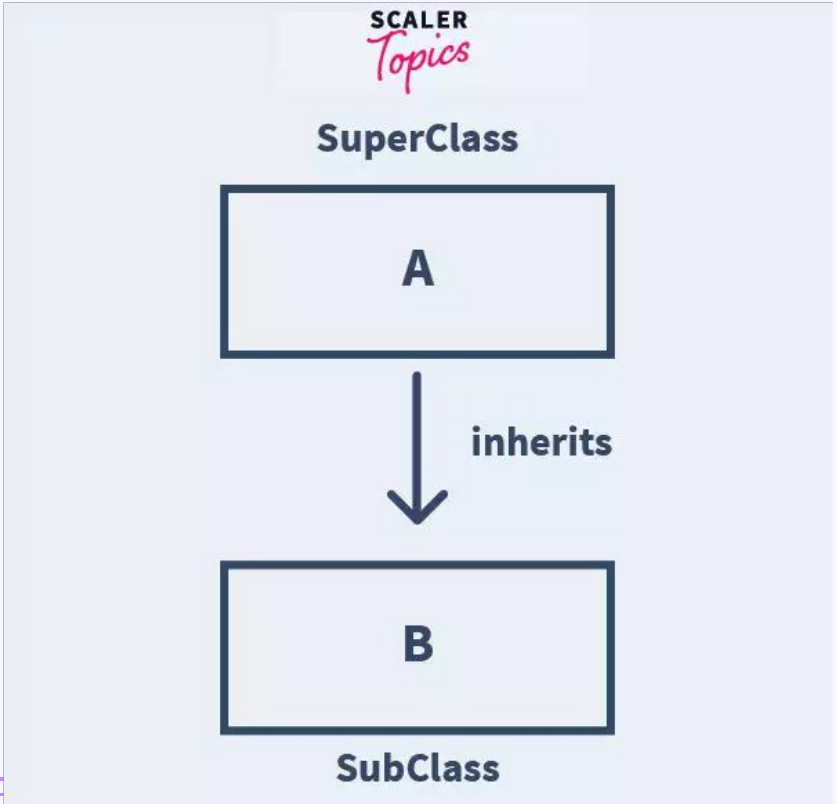
- Inheritance is one of the key features of to create a new class from an existing class.

- Two categories of “inheritance concept”:
  subclass (child) - the class that inherits from another class
  superclass (parent) - the class being inherited from

- Keyword: **extends** (The subclass uses the keyword to inherit the superclass)

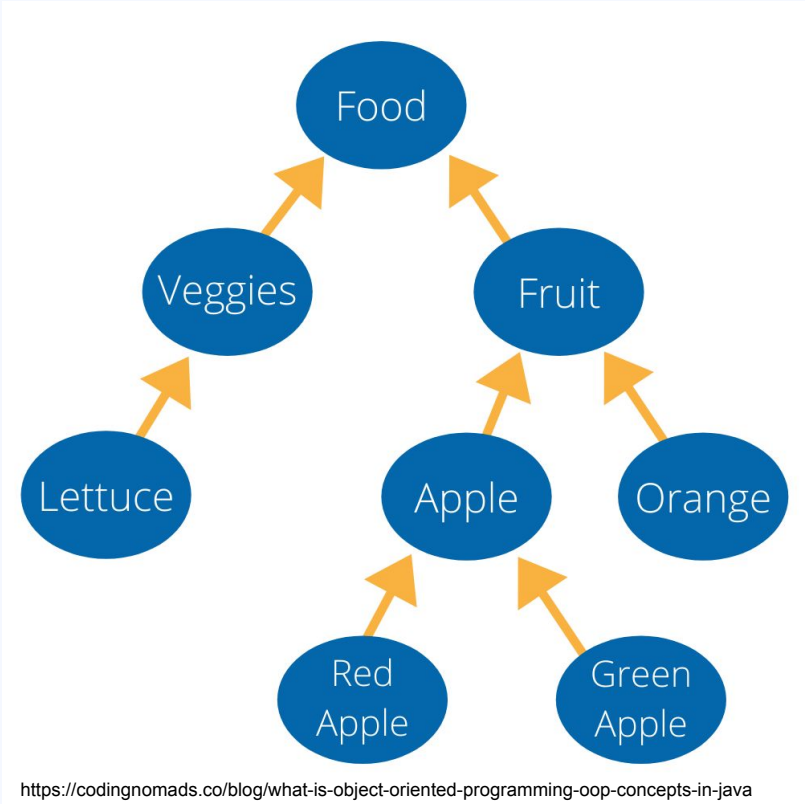
```
class derived-class extends base-class
{
  //methods and fields
}
```

Inheritance



The **subclass will get all the attributes and methods** of the superclass when it inherits the superclass.

Inheritance



https://codingnomads.co/blog/what-is-object-oriented-programming-oop-concepts-in-java

Inheritance

Animal.java

```java
package latihan;

class Animal {
    // attributes and method of the parent class
      String name;
      public void eat() {
        System.out.println("Yummy..yummy");
      }
}
```

Panda.java

```java
package latihan;

class Panda extends Animal {
    // new method in subclass
      public void display() {
        System.out.println("My name is " + name);
      }
}
```

Inheritance

AnimalMain.java

```java
package latihan;

public class AnimalMain {

    public static void main(String[] args) {
        // TODO Auto-generated method stub
        // create an object of the subclass
        Panda panda1 = new Panda();

        // access field of superclass
        panda1.name = "Yin yang";
        panda1.display();

        // call method of superclass
        // using object of subclass
        panda1.eat();

    }
}
```

Inheritance

Vehicle.java

```java
package latihan;

abstract class Vehicle {
    // Abstract method - the method below does not have a “method body”
    public abstract int numberOfWheels();

    // Below is a standard method with a body
    public void start() {
        System.out.println("Starting vehicle.");
    }
}
```

Car.java

```java
package latihan;

public class Car extends Vehicle {
    public int numberOfWheels() {
        // The method body of numberOfWheels() is provided here
        System.out.println("Car has 4 wheels");
        return 4;
          }
}
```

Inheritance

CarMain.java

```
1  package latihan;
2
3  public class CarMain {
4
5      public static void main(String[] args) {
6          // TODO Auto-generated method stub
7          Car myCar = new Car(); // Create a Car object
8
9          int wheels = myCar.numberOfWheels();
10         myCar.start();
11     }
12 }
```

Polymorphism

- Polymorphism means "**many forms**", and it occurs when we have many classes that are related to each other by inheritance.

- Polymorphism uses those methods to perform different tasks. This allows us to perform a single action in different ways.

Polymorphism

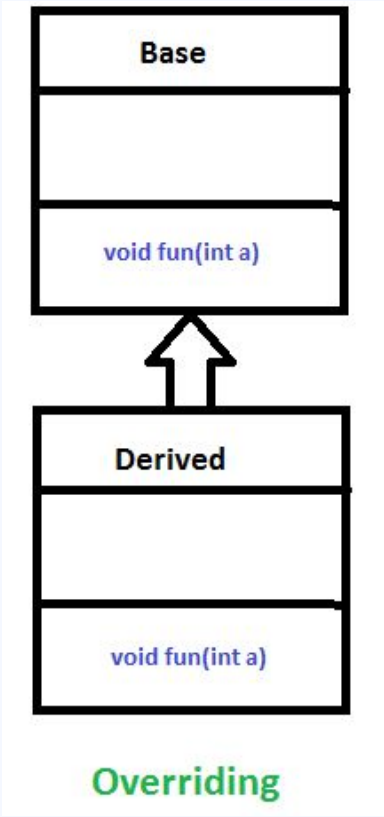
- **For example**, think of a superclass called **`Animal`** that has a method called **`animalSound()`**. Subclasses of Animals could be Cats, Dogs, Birds - And they also have their own implementation of an animal sound (the the cat meows, etc.)
- We can achieve polymorphism in Java using the following ways:
  - **Method Overriding**
  - **Method Overloading**

Polymorphism

Method Overriding

- To override a method in a parent class, the child class must declare a method with the same return type, name, and parameters. The method signatures must match exactly.

- @Override is an optional Java annotation that communicates that this method overrides a method in a parent class.



Polymorphism

Language.java

```java
package latihan;

class Language {
    public void displayInfo() {
        System.out.println("Common English Language");
    }
}
```

Java.java

```java
package latihan;

class Java extends Language {
    @Override
    public void displayInfo() {
        System.out.println("Java Programming Language");
    }
}
```

Polymorphism

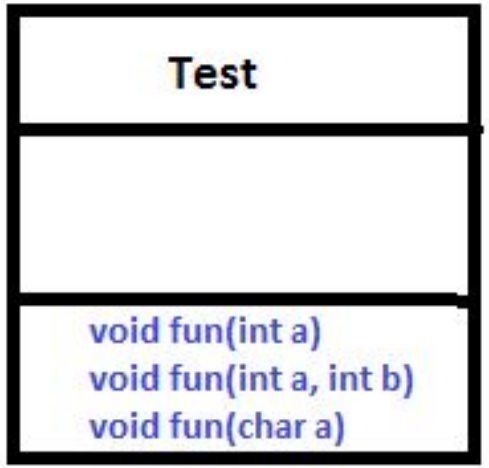
LanguageMain.java

```java
package latihan;

public class LanguageMain {

    public static void main(String[] args) {
        // TODO Auto-generated method stub

        // create an object of Java class
        Java j1 = new Java();
        j1.displayInfo();

        // create an object of Language class
        Language l1 = new Language();
        l1.displayInfo();


    }

}
```

Polymorphism

Method Overloading

- When there are multiple functions with the same name but different parameters then these functions are said to be overloaded. Functions can be overloaded by changes in the number of arguments or/and a change in the type of arguments.



Polymorphism

Pattern.java

```java
package latihan;

class Pattern {// method without parameter
	public void display() {
		for (int i = 0; i < 10; i++) {
			System.out.print("*");
		}
	}

	// method with single parameter
	public void display(char symbol) {
		for (int i = 0; i < 10; i++) {
			System.out.print(symbol);
		}
	}
}
```

Polymorphism

PatternMain.java

```
package latihan;

public class PatternMain {

    public static void main(String[] args) {
        // TODO Auto-generated method stub
        Pattern d1 = new Pattern();

        // call method without any argument
        d1.display();
        System.out.println("\n");

        // call method with a single argument
        d1.display('#');
    }
}
```

Thanks!

Ada pertanyaan?


slidesgo