OOP rewind
& SOLID Principles

PP_Paul

Let’s Rewind Concepts on OOP Concepts

What is a Class?

A class is a blueprint (It is user defined data types it could be anything) or prototype from which objects are created. This section defines a class that models the state and behavior of a real-world object. It intentionally focuses on the basics, showing how even simple classes can cleanly model state and behavior.

E.g.

```
class Demo {
    public static void main (String args[]) {
        System.out.println("Welcome to Java");
    }
}
```

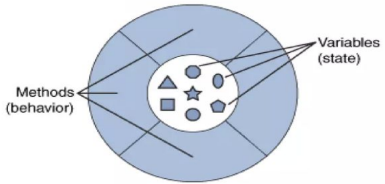
What is an Object?

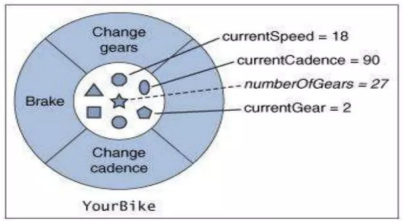
class hocche akta blueprint r object hocche oi blueprint hote akta physical instance-main difference of object & class

An object is a software bundle of related state and behavior. Software objects are often used to model the real-world objects that you find in everyday life (Object is real world Entity to represent a physical instance of a Class). A software object maintains its state in variables and implements its behavior with methods.

class create korle memory lage na,,object e lage,,tai bola hocche physical instance of a class

E.g.



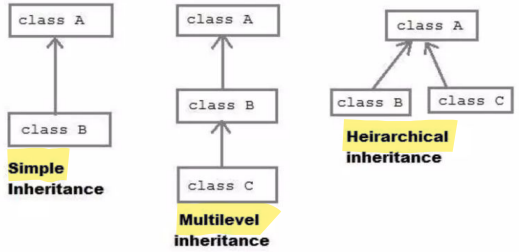


Instance of a Class

What is Inheritance?

Inheritance in object-oriented programming (OOP) is a mechanism that allows a class (referred to as the child or subclass) to inherit properties and behaviors (methods) from another class (referred to as the parent or superclass). This means that the child class can reuse the code and functionalities defined in the parent class, promoting code reuse and reducing redundancy.

Inheritance provides a powerful and natural mechanism for organizing and structuring your software. Now we will explain how classes inherit state and behavior from their super classes, and explains how to derive one class from another using the simple syntax provided by the Java programming language.



What is Inheritance?

simple akta class ke rakta class e inherit korbo extend er maddhome
jekhane akta child class, aktai parent class theke inherit hobe

E.g. Single Inheritance

```
class A
{
//statements;
}
class B extends A
{
    public static void main (String ar[])
    {
        System.out.println ("Welcome to Java Programming");
    }
}
```

What is Inheritance?

child er child kora, eikhane c er kace b er property ase, a ero ase

E.g. : Multilevel Inheritance

```
class A
{
    //statements;
}
class B extends A
{
   //statements;
}
class C extends B
{
  //statements;
  public static void main(String ar[])
  {
        //statements
  }
}
```

tree er moto akta relation chinta kori, akta base class ase, tar multiple children ase

E.g. Hierarchal Inheritance

```
class A
{
//statements;
}
class B extends A
{
//statements;
}
class C extends A
{
    public static void main(String ar[])
    {
        //statements;
    }
}
```

What is Abstraction?

Abstraction is the process of abstraction in Java is used to hide certain details and only show the essential features of the object. In other words, it deals with the outside view of an object (interface).

Abstract class cannot be instantiated; the class does not have much use unless it is subclass. This is typically how abstract classes come about during the design phase. A parent class contains the common functionality of a collection of child classes, but the parent class itself is too abstract to be used on its own.

What is Abstraction?

E.g.

```
abstract class A
{
   public abstract void sum(int x, int y);
}
class B extends A
{
    public void sum(int x,int y)
    {
        System.out.println(x+y);
    }
    public static void main(String ar[])
    {
        B obj=new B();
        obj.sum(2,5);
    }
```

vitorer function ta caller er janar dorkar nei-eitai abstraction

What is an Interface?

abstract class e ami chaile bivinno property or kono generic implementation korte parbo bt interface e ami only abstract method guloi rakhte parbo jeita thakbe signature wala-abstract & interface er main difference

jei class ta interface ke implement korbe, take obossoi sobgula method implement kora lagbe unless se nije abstract class hoi..(abstract class hole interface theke method implement na korleu cholbe)

An interface is a collection of abstract methods (it means all methods are only declared in an Interface). A class implements an interface, thereby inheriting the abstract methods of the interface. And that class implements interface then you need to defined all abstract function which is present in an Interface.

interface & class er main difference :-

An interface is not a class. Writing an interface is similar to writing a class, but they are two different concepts. A class describes the attributes and behaviors of an object. An interface contains behaviors that a class implements.

What is an Interface?

```
E.g.
interface A
{
    public void sumData(int x, int y);
}
class Demo implements A
{
      public void sumData (int x, int y)
      {
          System.out.println ("Total is "+(x+y));
      }
      public static void main (String ar[])
      {
          Demo d=new Demo ();
          d.sumData (10, 20);
      }
}
```

implement kora lagbe as normal class

What is an Encapsulation?

attribute, behavior er assess control korte pari-tai eke data hiding ow bola hoi

Encapsulation is one of the four fundamental OOP concepts. The other three are inheritance, polymorphism, and abstraction.

Encapsulation is the technique of making the fields in a class private and providing access to the fields via public methods. If a field is declared private, it cannot be accessed by anyone outside the class, thereby hiding the fields within the class. For this reason, encapsulation is also referred to as data hiding.

Benefits of Encapsulation:

Data Protection: By hiding data, you prevent accidental or malicious modification from outside the class. This ensures data integrity and consistency.
Controlled Access: You can define how data is accessed and modified through public methods (getters and setters). This allows you to implement logic for validation or specific operations before modifying the data.
Flexibility: You can change the internal implementation details of a class (how the data is stored or manipulated) without affecting external code that interacts with the public methods. This promotes maintainability and loose coupling between classes.

What is an Encapsulation?

main e age ke amra directly access korte parbo na

getter setter dutoi public kora bole agula dia age ke access kora jabe

```
class Person {

  // private field
  private int age;

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

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

```
class Main {
  public static void main(String[] args) {

    // 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());
  }
}
```

What is Polymorphism? (bohurupita)

Method overloading and method overriding uses concept of Polymorphism in Java where method name remains same in two classes but actual method called by JVM depends upon object at run time and done by dynamic binding in Java. Java supports both overloading and overriding of methods. In case of overloading method signature changes while in case of overriding method signature remains same and binding and invocation of method is decided on runtime based on actual object.

overloading er example e, jokhon akta class er instancetiate creat kore akta parameter pass kori, compiler runtime e bujhe nibe je take frst method ta execute korte hobe, r jodi duita parameter dei, tahole 2nd method ta execute korbe-ata jvm, runtime e, dynamic binding dia decide kore nei..

What is Polymorphism? [Method Overloading]

In Method overloading we have two or more functions with the same name but different arguments. Arguments must be changed on the bases of Number, orders and Data types.

E.g.

```
class A
{
    public void f1(int x)
    {
        System.out.println(x*x);
    }
    public void f1(int x,int y)
    {
        System.out.println(x*y);
    }
    public static void main(String ar[])
    {
        A a=new A();
        a.f1(5);
        a.f1(2,3);
    }
}
```

What is Polymorphism? [Method Overriding]

We have two classes and both classes have a function with the same name and same Parameters inheritance is necessary.

Eg. implementation ta alada

```
class B
{
    public void f1(int x,int y)
    {
        System.out.println(x+y);
    }
}
```

```
class A extends B
{
    public void f1(int x,int y)
    {
        System.out.println(x*y);
    }

public static void main(String ar[])
    {
        A a=new A();
        a.f1(5,5);
        B b=new B();
        b.f1(2,3);
    }
}
```

jodi ei portion na thakto r amra direct a.f1 kortam tahole error hoto na..cz by default parent er implement child e thakto cz public kora ase(inheritance er maddhome) akhn amra chacchi parent e jevabe ase,,oivabe na kore onno vabe korbo..

DP use- file opener: pdf-image-doc open kora lagbe..3 ta, 3 typer object chinta korle akhn similarly to r 3ta open korte parbo na..akta parent class ase dhori jar moddhe file location dile se file ta open kore..ete doc er jonno akdhoroner implementation thakbe, pdf er jonno.....imager er jonno.....ei jinisgular jonno method overridding concept ta use kori amra..


I WATCHED THIS
OOP Principles
BE REWIND
AND I DIDN'T
CRINGE AT ALL
imgflip.com

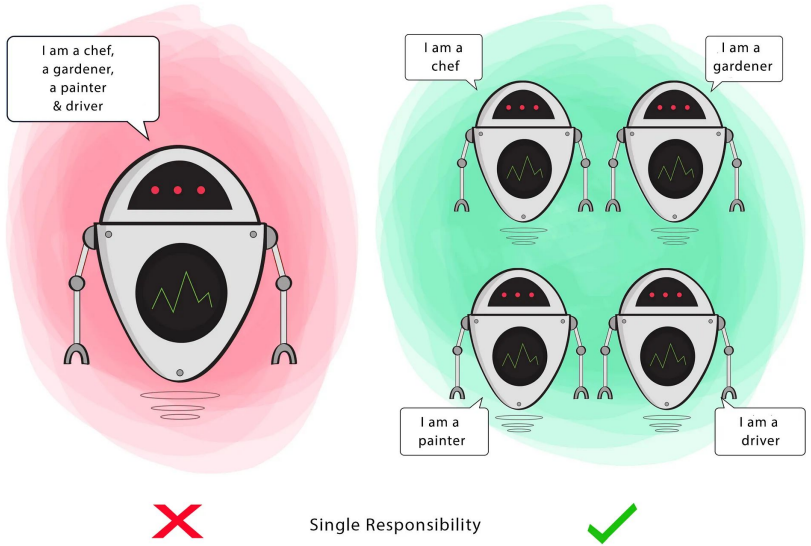
SOLID Principles

- S - Single-responsibility principle
  - One class should have one and only one responsibility.
- O - Open-closed principle
  - Objects or entities should be open for extension, but closed for modification.
- L - Liskov substitution principle
  - Derived types must be completely substitutable for their base types.
- I - Interface segregation principle
  - Clients should not be forced to depend on methods that they do not use.
- D - Dependency Inversion Principle
  - Entities must depend on abstractions not on concretions.

Single-responsibility principle

If a Class has many responsibilities, it increases the possibility of bugs because making changes to one of its responsibilities, could affect the other ones without you knowing.

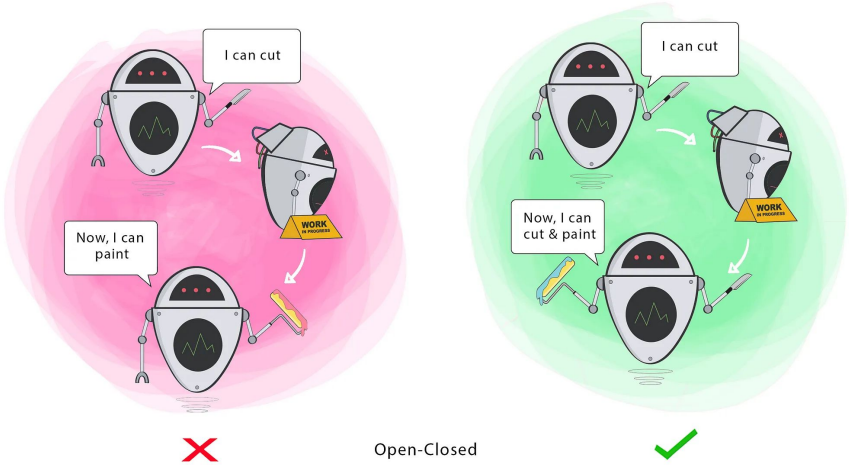
Goal:
This principle aims to separate behaviours so that if bugs arise as a result of your change, it won't affect other unrelated behaviours.



Open-closed principle

Changing the current behaviour of a Class will affect all the systems using that Class. If you want the Class to perform more functions, the ideal approach is to add to the functions that already exist NOT change them.

Goal:
This principle aims to extend a Class's behaviour without changing the existing behaviour of that Class. This is to avoid causing bugs wherever the Class is being used.



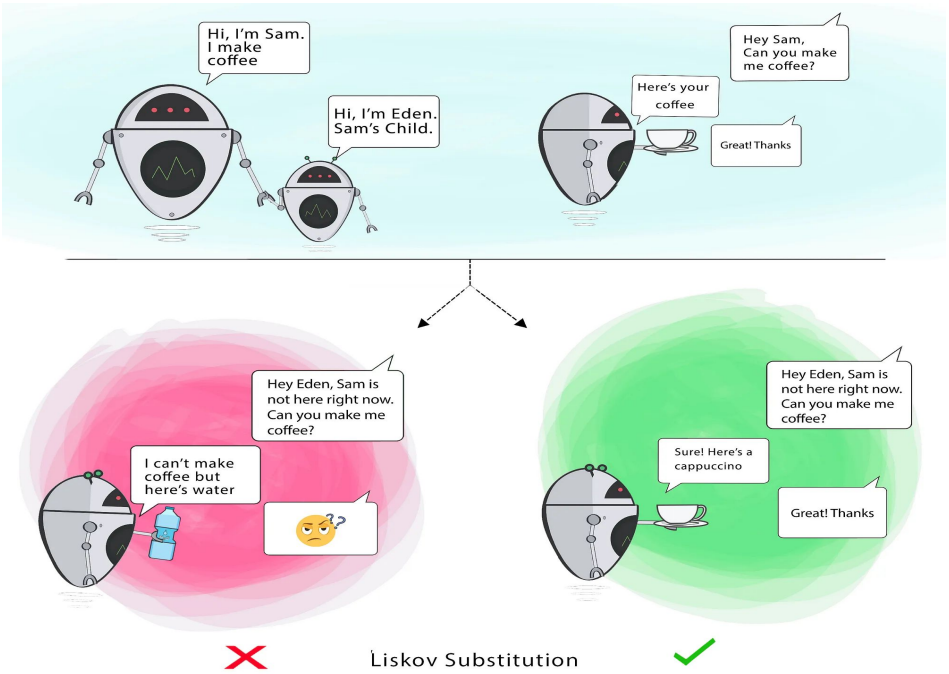
Liskov substitution principle

When a child Class cannot perform the same actions as its parent Class, this can cause bugs.

If you have a Class and create another Class from it, it becomes a parent and the new Class becomes a child. The child Class should be able to do everything the parent Class can do. This process is called Inheritance.

The child Class should be able to process the same requests and deliver the same result as the parent Class or it could deliver a result that is of the same type.

Goal:
This principle aims to enforce consistency so that the parent Class or its child Class can be used in the same way without any errors.

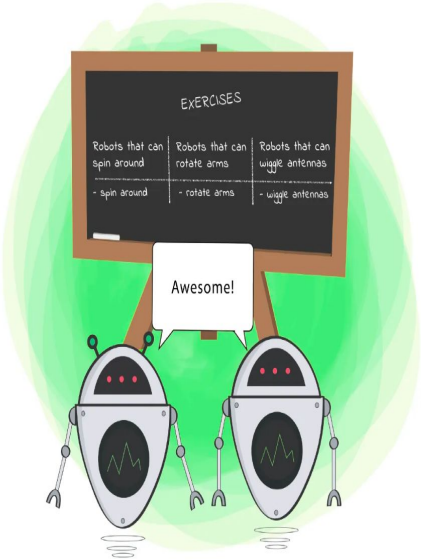


Interface segregation principle

When a Class is required to perform actions that are not useful, it is wasteful and may produce unexpected bugs if the Class does not have the ability to perform those actions.

A Class should perform only actions that are needed to fulfil its role. Any other action should be removed completely or moved somewhere else if it might be used by another Class in the future.

Goal:
This principle aims at splitting a set of actions into smaller sets so that a Class executes ONLY the set of actions it requires.



✗ Interface Segregation ✓

Dependency Inversion Principle

- High-level modules should not depend on low-level modules. Both should depend on the abstraction.

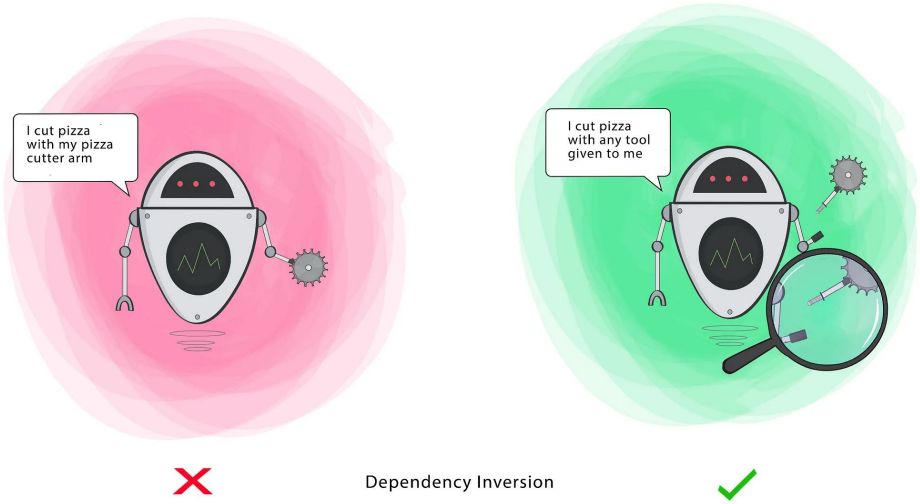
- Abstractions should not depend on details. Details should depend on abstractions.

Terms:
High-level Module(or Class): Class that executes an action with a tool.

Low-level Module (or Class): The tool that is needed to execute the action

Abstraction: Represents an interface that connects the two Classes.

Details: How the tool works



Dependency Inversion Principle

This principle says a Class should not be fused with the tool it uses to execute an action. Rather, it should be fused to the interface that will allow the tool to connect to the Class.

It also says that both the Class and the interface should not know how the tool works. However, the tool needs to meet the specification of the interface.

Goal:
This principle aims at reducing the dependency of a high-level Class on the low-level Class by introducing an interface.