# Object – Oriented Programming & Concepts

### What is OOPS

Object Oriented Programming or OOP is the technique to create programs based on **the real world**. It is a type of programming in which programmers define not only the data type of a data structure, but also the types of operations (functions) that can be applied to the data structure. In this way, the data structure becomes an object that includes both data and functions. In addition, programmers can create relationships between one object and another.

**Simula** is considered as the first object-oriented programming language.

The programming paradigm where everything is represented as an object is known as truly object-oriented programming language. **Smalltalk** is considered as the first truly object-oriented programming language.

Java is based an OOP. It **is not a pure Object Oriented Programming Language**.  
  
Many languages are Object Oriented. There are seven qualities to be satisfied for a programming language to be pure Object Oriented. They are:

1. Encapsulation/Data Hiding
2. Inheritance
3. Polymorphism
4. Abstraction
5. **All predefined types are objects**
6. All operations are performed by sending messages to objects
7. All user defined types are objects.

Java is not because it supports Primitive datatype such as int, byte, long... etc, to be used, which are not objects.  
Contrast with a pure OOP language like **Smalltalk**, where there are no primitive types, and boolean, int and methods are all objects.

**Why Java is popular than Smalltalk?**

* Smalltalk is too slow for big project
* Database connection was difficult
* Java is platform independent

### Why we need OOP?

*Designed / Implement your source code in terms of real world (i.e. Objects) then it is easy to understand, maintain, independent of developers. To remove complexity of codebase.*

Let’s see an example of Employee, their Address, role & Department

Address\_Data{

Flat, street, locality,city , state, country

}

Role\_Data {

roleId, roleName

}

Employee\_Data{

empId, name, roleId, mrgId, deptNo, address\_data

}

Department\_Data{

deptNo, deptName

}

Role\_Functions {

addRole(Role\_Data)

getRole(roleId)

listRole()

updateRole(Role\_Data)

deleteRole(roleId)

}

Employee\_Functions {

addEmplyoee(Employee\_Data)

getEmployee(empId)

listEmployee()

updateEmployee(Employee\_Data)

deleteEmployee(empId)

}

Department\_Functions {

addDepartment(Department\_Data)

getDepartment(deptNo)

listDepartment()

updateDepartment(Department\_Data)

deleteDepartment(deptNo)

}

In procedural approach, all functions should be inside a Main Fucntion where it is difficult to maintain code for a big system in future whereas in OOP since every object (i.e. Employee OR Role OR Department) are separate entity so put a particular object’s state & their behaviour together which will provide code is maintaintable, flexible, reusable, etc.

Therefore, principal advantages of object-oriented programming techniques over procedural programming techniques is that **they enable programmers to create modules that do not need to be changed when a new type of object is added**. A programmer can simply create a new object that inherits many of its features from existing objects. This makes object-oriented programs easier to modify.

### When do we use OOP?

Use Procedural approach where we don’t have detailed idea about all components of a system like in device driver progamming, etc. In this scenario always prefer to do Top-Down approach.

**When there are many simple independent tasks / modules to perform & severals programmers to involve completing the system by doing simultanuosly then always preferred to use OOP which flows Bottom-up approach.**

# Features / Principle of OOPs

### Object

Any entity that has state and behavior is known as an object. For example: chair, pen, table, keyboard, bike etc. It can be physical and logical.

### Class

**Collection of objects** is called class. It is a logical entity.

A class is a blueprint or prototype from which objects are created. This section defines a class that models the state and behavior of a real-world object.

### Abstraction

In one line definition is– *Only exposed what’s necessary*.

**Hiding internal details and showing functionality** is known as abstraction.

Abstraction is the concept of exposing only the required essential characteristics and behavior with respect to a context.

Abstraction simply means generalizing something to hide the complex logic that goes underneath.

For example:

1. **Real world Example of Abstraction:** -

Suppose you have an object Mobile Phone.

Suppose you have 3 mobile phones as following:-

Nokia 1400 (Features:- Calling, SMS)

Nokia 2700 (Features:- Calling, SMS, FM Radio, MP3, Camera)

Black Berry (Features:-Calling, SMS, FM Radio, MP3, Camera, Video Recording, Reading E-mails)

Abstract information (Necessary and Common Information) for the object "Mobile Phone" is make a call to any number and can send SMS."

so that, for mobile phone object you will have abstract class like following:-

    abstract class MobilePhone

    {

        public void Calling();

        public void SendSMS();

    }

    public class Nokia1400 extends MobilePhone

    {

    }

    public class Nokia2700 extends MobilePhone

    {

        public void FMRadio();

        public void MP3();

        public void Camera();

    }

    public class BlackBerry extends MobilePhone

    {

        public void FMRadio();

        public void MP3();

        public void Camera();

        public void Recording();

        public void ReadAndSendEmails();

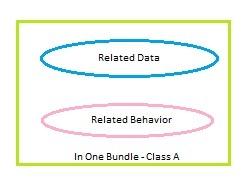
    }

Abstraction means putting all the variables and methods in a class which are necessary.

1. People own savings accounts, checking accounts, credit accounts, investment accounts, but not generic bank accounts. In this case, a bank account can be an abstract class and all the other specialized bank accounts inherit from bank account.
2. Phone call, we don't know the internal processing.
3. In general computer software, when we talk about abstraction the software language itself is an example for the concept of abstraction. When we write a statement as,  
   *a = b + c;*we are adding two values stored in two locations and then storing the result in a new location. We just describe it in an easily human understandable form. What happens beneath? There are registers, instruction sets, program counters, storage units, etc involved. There is PUSH, POP happening. High level language we use abstracts those complex details.
4. ATM is a good example of abstraction, we doesn't know what are all the internal processes that are going when we access it. Instead we know only the inquiries, deposit, withdrawl etc

In java, we use **abstract class and interface** to achieve abstraction.

### Encapsulation



In one line definition is– *Hide complexity*

Encapsulation in Java you see all member variables are made private so they are well encapsulated you can only change or access this variable directly inside this class. If you want to allow outside world to access these variables is better creating a getter and setter that allows you to do any kind of validation, security check before return loan so it gives you complete control of whatever you want to do and single channel of access for client who is controlled and managed.

Class Loan {

private String loan;

public String getLoan() {

**….**

return loan;

}

}

**Question**: Why it is recommended to use getter & settor (i.e. **Java beans or POJO or VO or DTO**)?

**Advantage of Encapsulation in Java and OOPS**

Here are few advantages of using **Encapsulation**while writing code in Java or any Object oriented programming language:

1. Encapsulated Code is more flexible and easy to change with new requirements.

2. Encapsulation in Java makes unit testing easy.

3. Encapsulation in Java allows you to control who can access what.

4. **Encapsulation also helps to write immutable class in Java which are a good choice in multi-threading**

**environment**.

5. Encapsulation reduce coupling of modules and increase cohesion inside a module because all piece of one thing

are encapsulated in one place.

6. Encapsulation allows you to change one part of code without affecting other part of code.

**Important point’s aboue encapsulation in Java**-

1. "Whatever changes encapsulate it" is a famous design principle.

2. Encapsulation helps in loose coupling and high cohesion of code.

3. Encapsulation in Java is achieved using access modifier private, protected and public.

4. **Factory pattern, Singleton pattern in Java makes good use of Encapsulation**

### Abstaction Vs Encapsulation

Both are complementary to each other because abstraction states shows only thing which are required to user & encapsulation states that hide the compelxities. Abstraction is more towards design part where as encapsulation is more towards low level implementation.

**Encapsulation** is **wrapping and controlling access of logically related data in a class**. One of the aspects of this is by using suitable **access modifiers**. A**bstraction** on the other hand is **generalizing the concept so that the underlying complex logic is hidden from the user**. In java this is achieved by using **interfaces** and **abstract classes**.

Abstraction is implemented in Java using interface and abstract class while Encapsulation is implemented using private, package-private and protected access modifier.

|  |  |
| --- | --- |
|  | |
| **Abstraction** | **Encapsulation** |
| 1. Abstraction solves the problem in the design level. | 1. Encapsulation solves the problem in the implementation level. |
| 2. Abstraction is used for hiding the unwanted data and giving relevant data. | 2. Encapsulation means hiding the code and data into a single unit to protect the data from outside world. |
| 3. Abstraction lets you focus on what the object does instead of how it does it | 3. Encapsulation means hiding the internal details or mechanics of how an object does something. |
| 4. Abstraction- Outer layout, used in terms of design.  For Example:-   Outer Look of a Mobile Phone, like it has a display screen and keypad buttons to dial a number. | 4. Encapsulation- Inner layout, used in terms of implementation.  For Example:- Inner Implementation detail of a Mobile Phone, how keypad button and Display Screen are connect with each other using circuits. |

**For example –**

1. complex logic is in the circuit board which is encapsulated in a touchpad and a nice interface (buttons) is provided to abstract it out.
2. GSM.java => sendMessage(MessageObject)

Abstraction is GSM->MessageDeliveryStatus sendMessage(MessageObject)

Encapsulation is -> + sendMessage(MessageObject)

* isDesviceActive()
* isReceipientPresent()
* isMessageTextValidated()
* isMessageDelivered()
* etc.

**Assignment3 –** Modify com.src.java.oops.\* using Interface

### Inheritance

Java Inheritance defines an is-a relationship between a superclass and its subclasses. This means that an object of a subclass can be used wherever an object of the superclass can be used. Class **Inheritance in java** mechanism is used to build new classes from existing classes. The inheritance relationship is transitive: if class x extends class y, then a class z, which extends class x, will also inherit from class y.

***Inheritance allows one class to reuse the functionality provided by its superclasses.***

class clsName2 extends clsName1

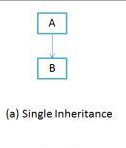
{

// class body

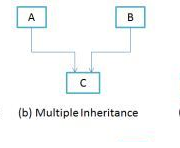
}

**Types of Inheritence**

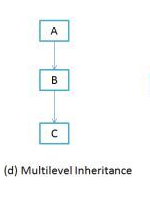
1. Single Inheritance



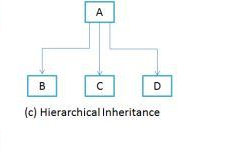
1. Multiple Inheritance



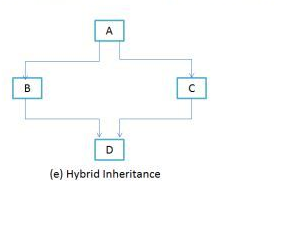
1. Mutilevel Inheritance



1. Hierarchical Inheritance

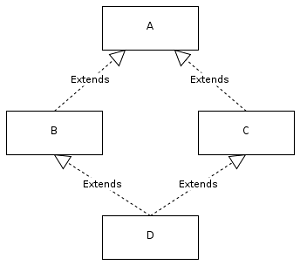


1. Hybrid Inheritance



**Why java does not support multiple inheritance?**

In order to enforce simplicity should be the main reason for omitting multiple inheritance. For instance, we can consider **diamond problem of multiple inheritance**.



We have two classes B and C inheriting from A. Assume that B and C are overriding an inherited method and they provide their own implementation. Now D inherits from both B and C doing multiple inheritances. D should inherit that overridden method, which overridden method will be used? Will it be from B or C? Here we have an ambiguity.

**Example**- com.src.java.oops.\*

### Polymorphism

Polymorphism literally means taking more than one form. Polymorphism is a characteristic of being able to assign a different behavior or value in a subclass, to something that was declared in a parent class.

Polymorphism means— we can dynamically change the behaviour of a method.

One name many form means method have same name with different behaviour

**Real World Example of Polymorphism:**

Example-1

A Teacher behaves to student.

A Teacher behaves to his/her seniors.

Here teacher is an object but attitude is different in different situation.

Example-2

Person behaves SON in house at the same time that person behaves EMPLOYEE in office.

Example-3

Your mobile phone, one name but many forms

* As phone
* As camera
* As mp3 player
* As radio

### Types of Polymorphism in JAVA

#### Static or Compile time Polymorphism or Method Overloading

Method overloading in Java occurs when two or more methods shares same name and fulfill at least one of the following condition.

**1)** Have different number of arguments.  
**2)** Have same number of arguments but their types are different.  
**3)** Have both different numbers of arguments with a difference in their types.

**Example – Core Java: com.src.java.polymorphism.overloading – Calculator.java**

Note: Return type not part of method overloading.

#### Dynamic or Runtime Polymorphism or Method Overriding

Method Overriding is possible in the case of inheritance where child class inherits the property of parent class, i.e; if parent class has some method then child class can override the method but keep in mind that signature of the method must be same as on parent class. At runtime JVM will decide which is to call depending upon its reference.  
In object oriented terms, overriding means to override the functionality of any existing method.

**Rules-**

* The argument list must exactly same that of the overridden method. If they don't match, you can end up with an overloaded method.
* The return type must be the same as, or a subtype of, the return type declared in the original overridden method in the superclass (also called covariant return type).
* You cannot override a method marked final.
* You cannot override a method marked static.
* The overriding method must not throw checked exceptions that are new or broader than those declared by the overridden method. For example, a method that declares a FileNotFoundException cannot be overridden by a method that declares a SQLException, Exception, or any other non-runtime exception unless it's a subclass of FileNotFoundException.

**Example – Core Java: com.src.java.polymorphism. runtime –**

**Parent.java**

**Child.java**

**TestRuntimePolymorphism.java**

**Assignment4 –** List down Difference between Overloading & Overriding

**Assignment5 –** Modify com.src.java.oops.TestMobilePhones.java using dynamic polymorphism

# Constructor in Java

**Constructor** is a **special type of method** that is used to initialize the object. Constructor is **invoked at the time of object creation**. It constructs the values i.e. provides data for the object that is why it is known as constructor.

**Constructor can’t be static**.

There are two types of constructors:

* default constructor (no-arg constructor)
* parameterized constructor
* Constructor Chaining (base class constructor called first)  
  Use super to call parent class constructor / methods  
  Use this to call invoking object constructor / methods

**Example – Core Java: com.src.java.constructor –**

* **Employee.java**
* **TestEmployeeClient.java**