# Java Day 1

OOP Concepts

## 1. What is OOP

OOP is a design philosophy. It stands for Object Oriented Programming.

**O**bject-**O**riented **P**rogramming (OOP) uses a different set of programming languages than old procedural programming languages (C, Pascal, etc.).

Everything in OOP is grouped as self sustainable "objects".

Hence, you gain re-usability by means of four main object-oriented programming concepts.

## 2. What is an object?

An object can be considered a "*thing*" that can perform a set of **related** activities.

The set of activities that the object performs defines the object's behavior.

For example, the hand can grip something or a *Student* (*object*) can give the name or address.

In pure *OOP* terms an object is an instance of a class.

## 3. What is a class?

A *class* is simply a representation of a type of *object*. It is the blueprint/ plan/ template that describe the details of an *object*. A class is the blueprint from which the individual objects are created. *Class* is composed of three things: a name, attributes, and operations.

```
public class Student
{
}
```

According to the sample given below we can say that the Student object, named objectStudent, has been created out of the Student class.

```
Student objectStudent = new Student();
```

## 4. How to design and identify a class?

This is an art; each designer uses different techniques to identify classes. However according to Object Oriented Design Principles, there are five principles that you must follow when design a class,

- SRP - The Single Responsibility Principle -
- A class should have one, and only one, reason to change.
- OCP - The Open Closed Principle -
- You should be able to extend a classes behavior, without modifying it.
- LSP - The Liskov Substitution Principle-
- Derived classes must be substitutable for their base classes.
- DIP - The Dependency Inversion Principle-
- Depend on abstractions, not on concretions.
- ISP - The Interface Segregation Principle-
- Make fine grained interfaces that are client specific.

## 5. What is Encapsulation or Information Hiding?

The encapsulation is the inclusion within a program object of all the resources need for the object to function - basically, the methods and the data.

In *OOP* the encapsulation is mainly achieved by creating classes, the classes expose public methods and properties.

The class is kind of a container or capsule or a cell, which encapsulate the set of methods, attribute and properties to provide its indented functionalities to other classes.

In that sense, encapsulation also allows a class to change its internal implementation without hurting the overall functioning of the system. That idea of encapsulation is to hide how a class does it but to allow requesting what to do.

## 6. What is Association?

Association is a relationship between two classes. It allows one object instance to cause another to perform an action on its behalf.

Association is the more general term that define the relationship between two classes, where as the aggregation and composition are relatively special.

```
public class StudentRegistrar
{
    public StudentRegistrar ();
    {
        new RecordManager().Initialize();
    }
}
```

In this case we can say that there is an association between StudentRegistrar and RecordManager or there is a directional association from StudentRegistrar to RecordManager or StudentRegistrar use a (*__Use__*) RecordManager. Since a direction is explicitly specified, in this case the controller class is the StudentRegistrar.

## 7. What is the difference between Association, Aggregation and Composition?

Association is a (***a***) relationship between two classes, where one class use another. But aggregation describes a special type of an association. Aggregation is the (***the***) relationship between two classes. When object of one class has an (***has***) object of another, if second is a part of first (containment relationship) then we called that there is an aggregation between two classes. Unlike association, aggregation always insists a direction.

## 8. What is abstraction and generalization?

Abstraction is an emphasis on the idea, qualities and properties rather than the particulars (a suppression of detail).
While abstraction reduces complexity by hiding irrelevant detail, generalization reduces complexity by replacing multiple entities which perform similar functions with a single construct. Generalization is the broadening of application to encompass a larger domain of objects of the same or different type. Programming languages provide generalization through variables, parameterization, generics and *polymorphism*. It places the emphasis on the similarities between objects.

## 9. What is an abstract class?

Abstract classes, which declared with the abstract keyword, cannot be instantiated. It can only be used as a super-class for other classes that extend the abstract class. Abstract class is the concept and implementation gets completed when it is being realized by a subclass.

In addition to this a class can inherit only from one abstract class (but a class may implement many interfaces) and must override all its abstract methods/ properties and may override virtual methods/ properties.

Abstract classes are ideal when implementing frameworks.

## 10. What is an interface?

Interface separates the implementation and defines the structure, and this concept is very useful in cases where you need the implementation to be interchangeable.

Interface can be used to define a generic template and then one or more abstract classes to define partial implementations of the interface.

Interface definition begins with the keyword interface. An interface like that of an abstract class cannot be instantiated.

Programming to an interface is considered good design.

## 11. What is the difference between an interface and abstract class?

An abstract class can have instance methods that implement a default behavior.

An Interface can only declare constants and instance methods, but cannot implement default behavior and all methods are implicitly abstract.

An interface has all public members and no implementation.

An abstract class is a class which may have the usual flavors of class members (private, protected, etc.), but has some abstract methods.

## 12. What is Inheritance?

The ability of a new class to be created, from an existing class by extending it, is called *inheritance*.

Java does not provide multiple inheritance.

## 13. What is Polymorphism?

Polymorphisms is a generic term that means 'many shapes'.

More precisely *Polymorphism* means the ability to request that the same operations be performed by a wide range of different types of things.

In *OOP*, Polymorphism is achieved by using many different techniques named method overloading, operator overloading, and method overriding,

## 14. What is method overloading?

Method overloading is the ability to define several methods all with the same name.

```
public class MyLogger
{
    public void LogError(Exception e)
    {
        // Implementation goes here
    }

    public bool LogError(Exception e, string message)
    {
        // Implementation goes here
    }
}
```

## 15. What is operator overloading?

The operator overloading (less commonly known as ad-hoc *polymorphisms*) is a specific case of *polymorphisms*in which some or all of operators like +, - or == are treated as polymorphic functions and as such have different behaviors depending on the types of its arguments.

```
public class Complex
{
    private int real;
    public int Real
    { get { return real; } }

    private int imaginary;
    public int Imaginary
    { get { return imaginary; } }

    public Complex(int real, int imaginary)
    {
        this.real = real;
        this.imaginary = imaginary;
    }

    public static Complex operator +(Complex c1, Complex c2)
    {
        return new Complex(c1.Real + c2.Real, c1.Imaginary + c2.Imaginary);
    }
}
```

## 16. What is method overriding?

Method overriding is a language feature that allows a subclass to override a specific implementation of a method that is already provided by one of its super-classes.

A subclass can give its own definition of methods but need to have the same signature as the method in its super-class.

This means that when overriding a method the subclass's method has to have the same name and parameter list as the super-class' overridden method.