Object-Oriented Programming (OOPS-2)

What you will learn in this lecture?

- Components of OOPs.
- Access modifiers with inheritance and protected modifiers.
- All about exception handling.

Encapsulation

Encapsulation is defined as the wrapping up of data under a single unit. It is the mechanism that binds together code and the data it manipulates. Another way to think about encapsulation is, it is a protective shield that prevents the data from being accessed by the code outside this shield.

- Technically in encapsulation, the variables or data of a class is hidden from any other class and can be accessed only through any member function of its own class in which they are declared.
- As in encapsulation, the data in a class is hidden from other classes using the data hiding concept which is achieved by making the members or methods of class as private and the class is exposed to the end user or the world without providing any details behind implementation using the abstraction concept, so it is also known as combination of data-hiding and abstraction..

- Encapsulation can be achieved by: Declaring all the variables in the class as private and writing public methods in the class to set and get the values of variables.

Inheritance

- Inheritance is a powerful feature in Object-Oriented Programming.
- Inheritance can be defined as the process where one class acquires the properties (methods and fields) of another. With the use of inheritance, the information is made manageable in a hierarchical order.
- The class which inherits the properties of the other is known as **subclass** *(derived class or child class)* and the class whose properties are inherited is known as **superclass** *(base class, parent class)*.

Super Keyword:

The super keyword in Java is a reference variable which is used to refer to an immediate parent class object.

Whenever you create an instance of a subclass, an instance of the parent class is created implicitly which is referred to by a super reference variable.

Let us take a real-life example to understand inheritance. Let's assume that **Human** is a class that has properties such as **height**, **weight**, **age**, etc and functionalities (or methods) such as **eating()**, **sleeping()**, **dreaming()**, **working()**, etc.

Now we want to create **Male** and **Female** classes. Both males and females are humans and they share some common properties (like **height**, **weight**, **age**, etc) and behaviors (or functionalities like **eating()**, **sleeping()**, etc), so they can inherit these properties and functionalities from the **Human** class. Both males and females also have some characteristics specific to them (like men have short hair and females have long hair). Such properties can be added to the **Male** and **Female** classes separately.

This approach makes us write less code as both the classes inherited several properties and functions from the superclass, thus we didn't have to re-write them. Also, this makes it easier to read the code.

Java Inheritance Syntax

```
class SuperClass{
    // Body of parent class
}

class SubClass extends SuperClass{
    // Body of derived class
}
```

To inherit properties of the parent class, **extends** keyword is used followed by the name of the parent class.

Example of Inheritance in Java

To demonstrate the use of inheritance, let us take an example.

A polygon is a closed figure with 3 or more sides. Say, we have a class called Polygon defined as follows.

```
class Polygon{
    int n;
    int[] sides;

```