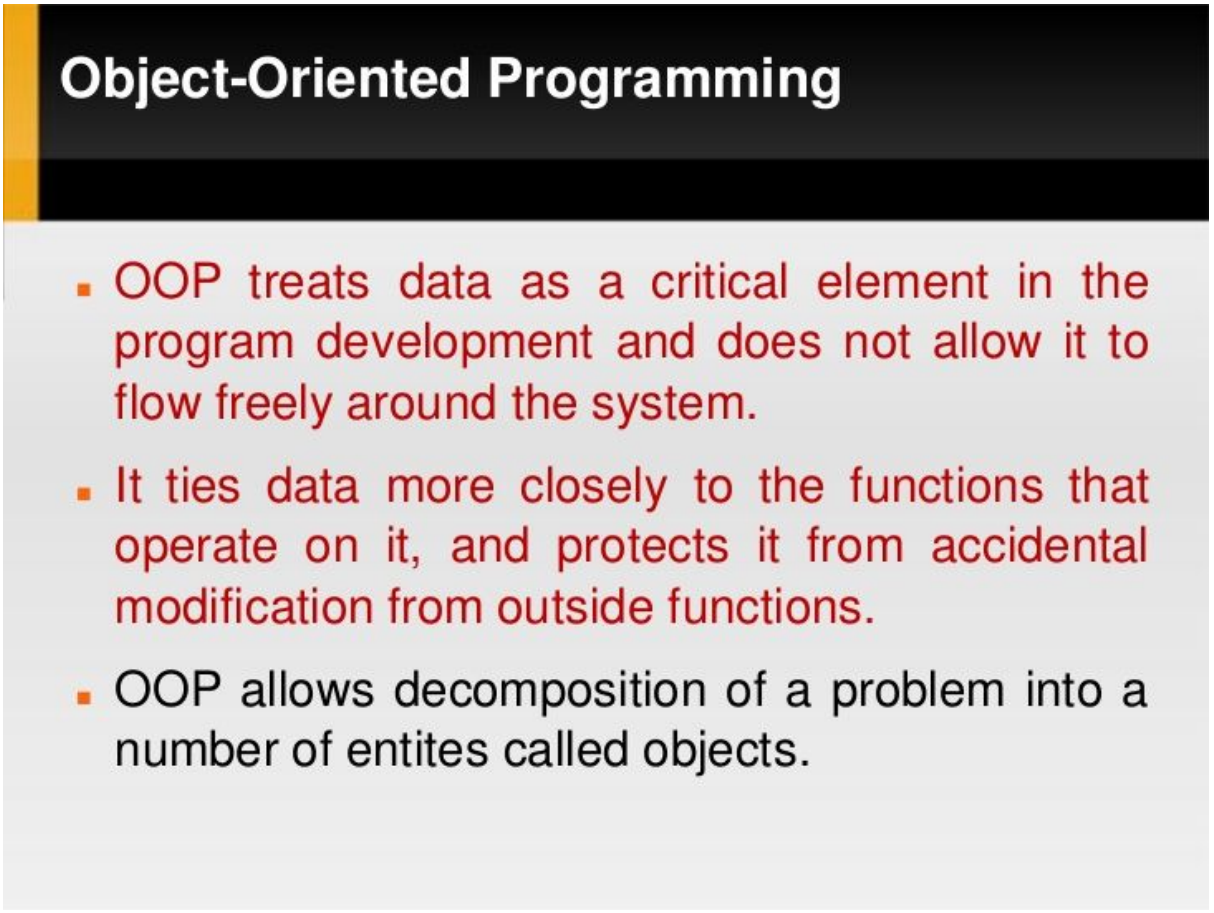
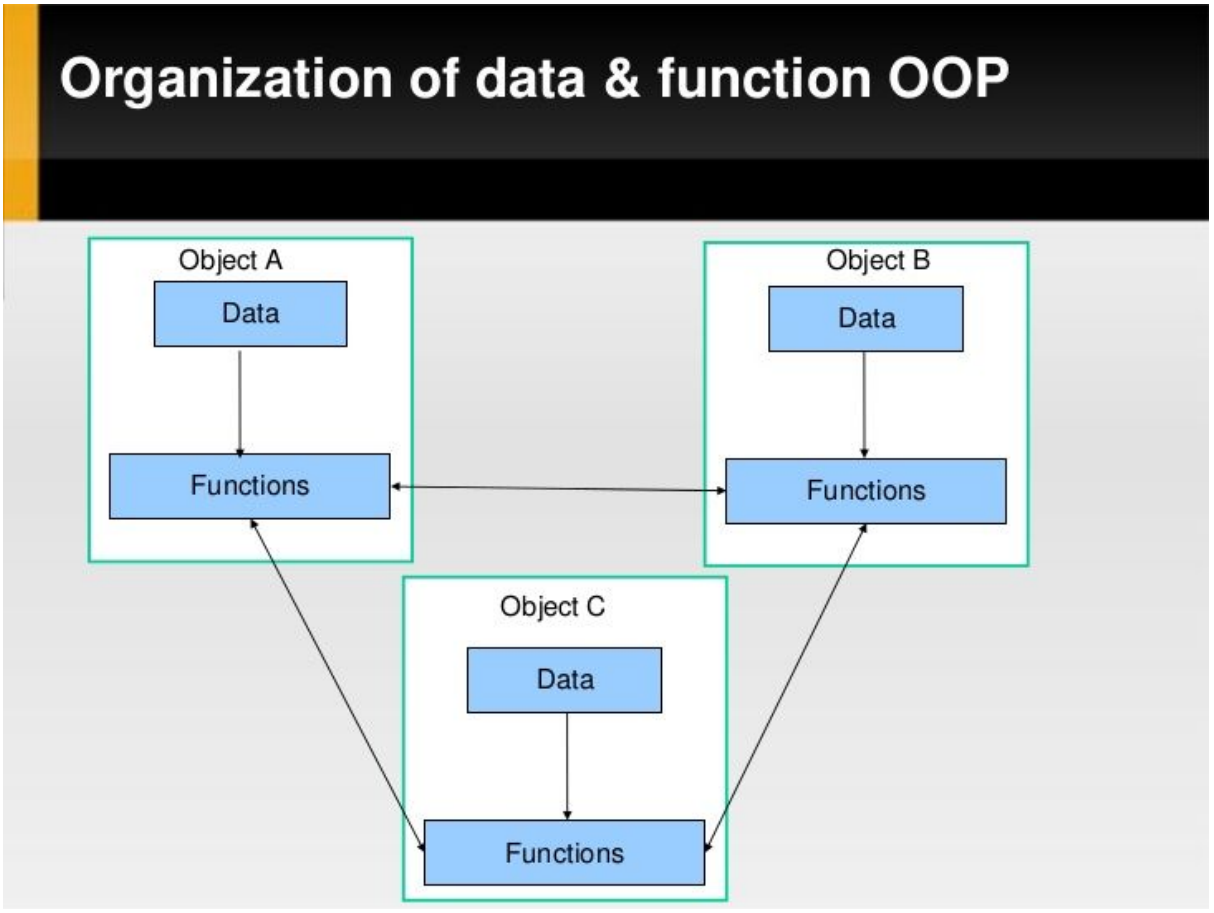
Slide 1 :

Object-oriented programming (OOP) is a programming paradigm based on the concept of objects, which may contain data, in the form of fields, (or attributes*)* and code, in the form of procedures, often known as methods. A feature of objects is that an object's procedures can access and often modify the data fields of the object with which they are associated

```
Class Car {
        String color;
        String model;

        String getColor(){
                return color;
        }

        void setColor(String color){
                this.color = color
        }

}
```



Slide 2 : Organization of data & function



Slide 3 : Objects, Methods and classes?

A class is an entity that determines how an object will behave and what the object will contain. In other words, it is a blueprint or a set of instruction to build a specific type of object.

```
class <class_name>{
    field;
    method;
  }
```

An object is nothing but a self-contained component which consists of methods and properties to make a particular type of data useful. Object determines the behavior of the class. When you send a message to an object, you are asking the object to invoke or execute one of its methods.

From a programming point of view, an object can be a data structure, a variable or a function. It has a memory location allocated. The object is designed as class hierarchies.

```
ClassName ReferenceVariable = new ClassName();
```

The difference

A **class** is a blueprint or prototype that defines the variables and the methods (functions) common to all objects of a certain kind.

An **object** is a specimen of a class. Software objects are often used to model real-world objects you find in everyday life.

https://www.guru99.com/java-oops-class-objects.html
https://www.slideshare.net/abhayjuneja/oops-ppt
https://www.slideshare.net/vinodthebest/basic-object-oriented-concepts/14
https://www.slideshare.net/thinkphp/object-oriented-programming-concepts
Slide 4 : **Interface**

In summary the Interface separates the implementation and defines the structure, and this concept is very useful in cases where you need the implementation to be interchangeable. Apart from that an interface is very useful when the implementation changes frequently. Some say you should define all classes in terms of interfaces, but I think recommendation seems a bit extreme.

Interface can be used to define a generic template and then one or more abstract classes to define partial implementations of the interface. Interfaces just specify the method declaration (implicitly public and abstract) and can contain properties (which are also implicitly public and abstract). Interface definition begins with the keyword interface. An interface like that of an abstract class cannot be instantiated.

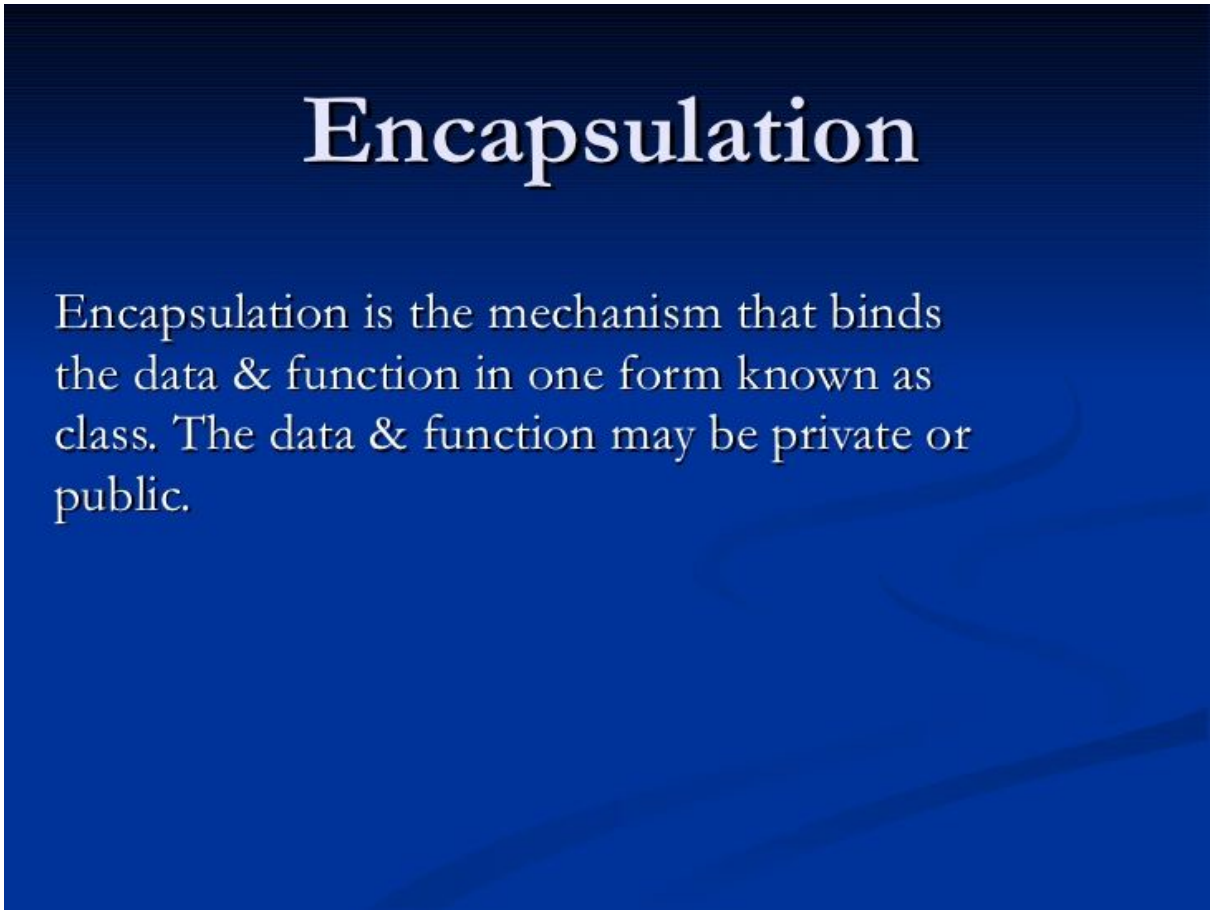
If a class that implements an interface does not define all the methods of the interface, then it must be declared abstract and the method definitions must be provided by the subclass that extends the abstract class. In addition to this an interfaces can inherit other interfaces.

Slide 4 a: **Abstraction & Encapsulation**

https://stackoverflow.com/questions/16014290/simple-way-to-understand-encapsulation-and-abstraction
Abstraction is a process where you show only “relevant” data and “hide” unnecessary details of an object from the user. Consider your mobile phone, you just need to know what buttons are to be pressed to send a message or make a call, What happens when you press a button, how your messages are sent, how your calls are connected is all abstracted away from the user.

Encapsulation is the process of combining data and functions into a single unit called class. In Encapsulation, the data is not accessed directly; it is accessed through the

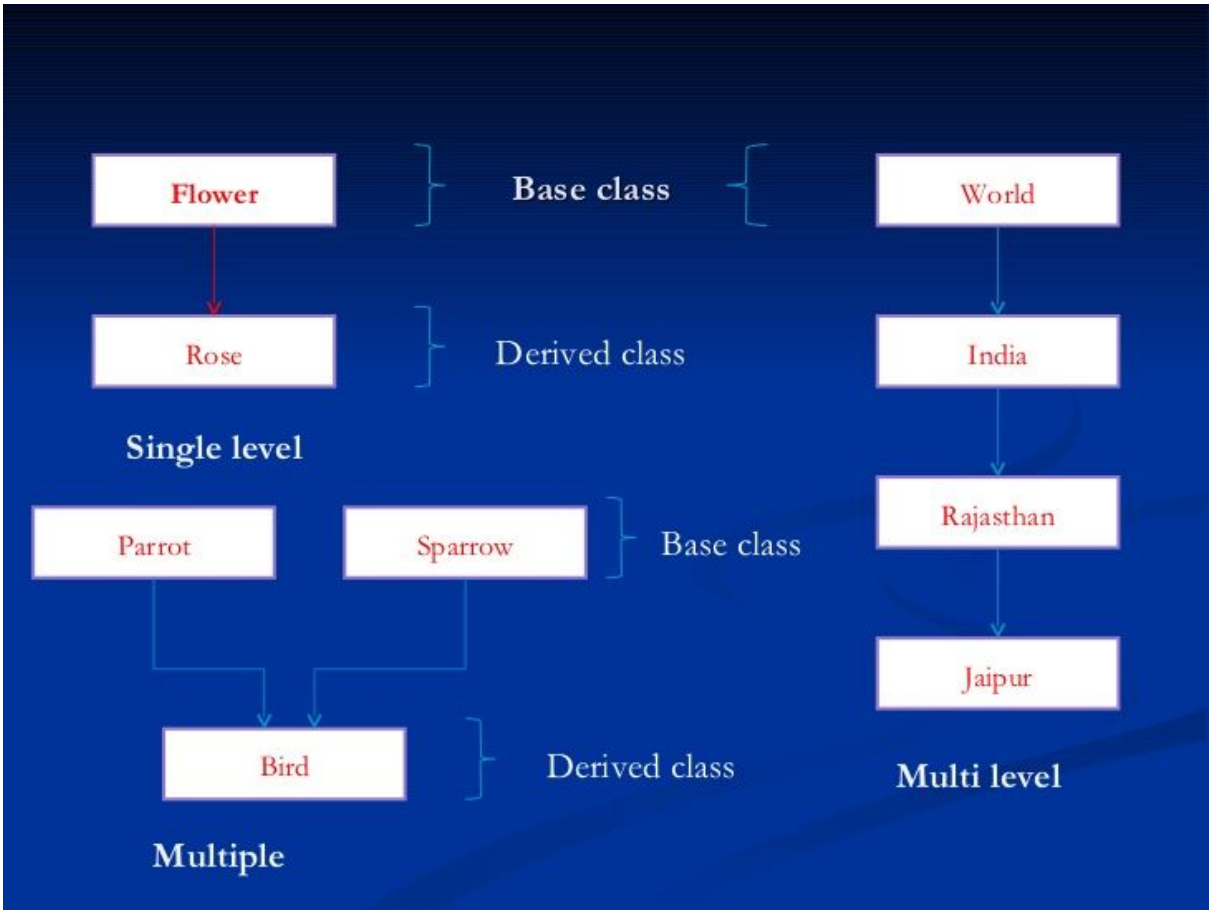
functions present inside the class. In simpler words, attributes of the class are kept private and public getter and setter methods are provided to manipulate these attributes. Thus, encapsulation makes the concept of data hiding possible.

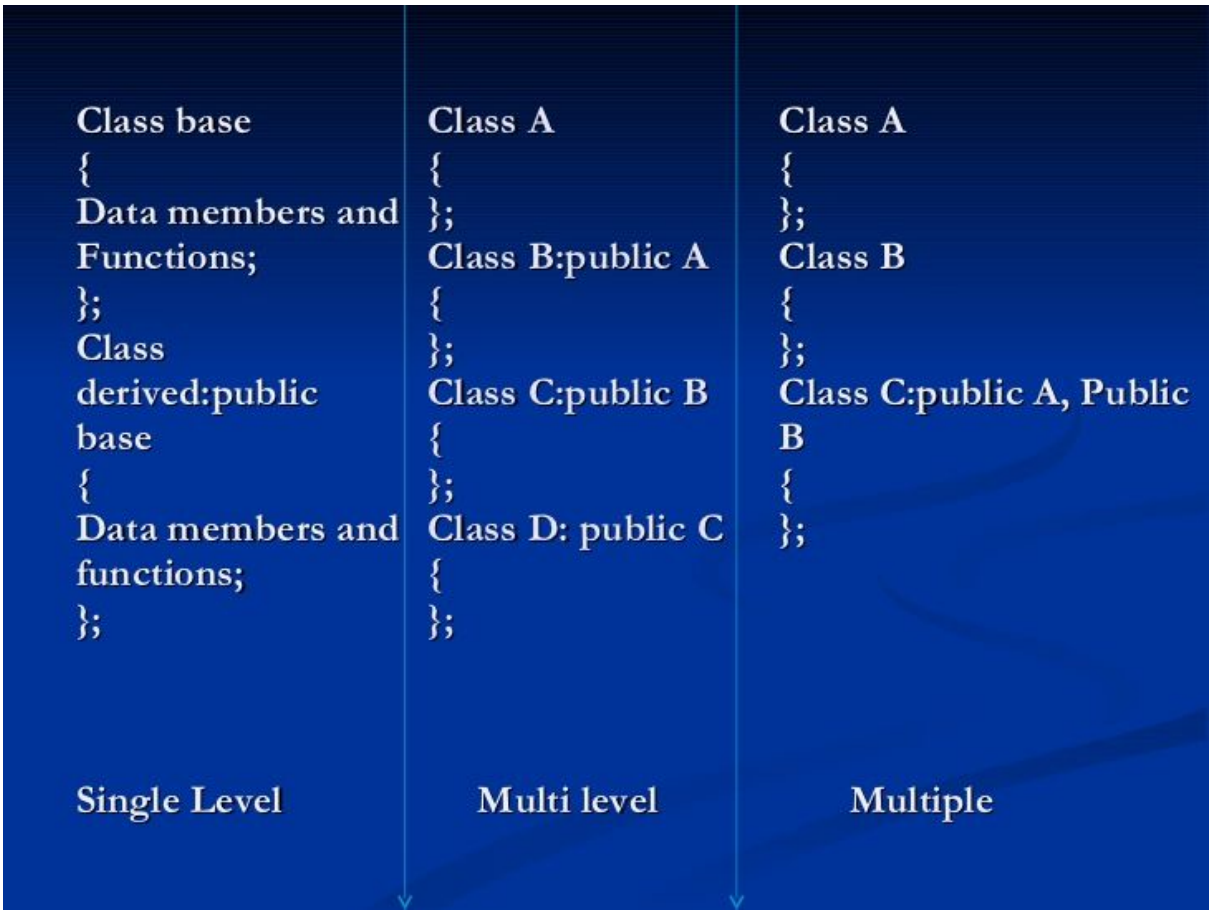


Slide 5: Inheritance

Inheritance

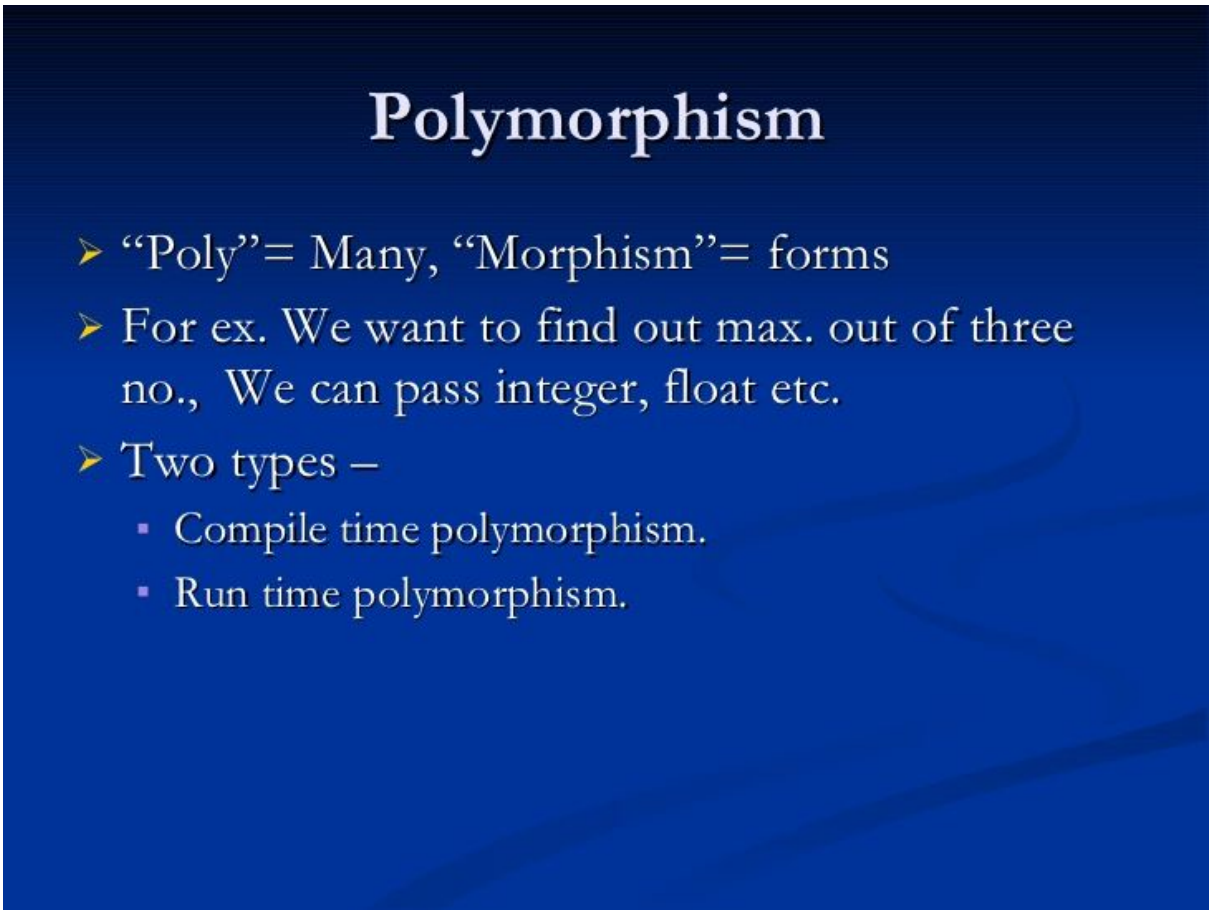
- Mechanism of deriving a new class from an already existing class.
- 5 types of inheritance
  - Single level
  - Multilevel
  - Multiple
  - Hierarchical
  - Hybrid





Slide 6: Polymorphism

Poly = “Many” morphism=”forms”



For example, given a base class *shape,*polymorphism enables the programmer to define different "*area"* methods for any number of derived classes, such as circles, rectangles and triangles. No matter what shape an object is, applying the *area* method to it will return the correct results. Polymorphism is considered to be a requirement of any true object-oriented programming language (OOPL).

Slide 7: The 3 pillere (Polymorphism, Inheritance & Encapsulation)