Chapter 4:

Basics of Object Oriented Programming

Informatics Practices

Class XII (CBSE Board)

Revised as per CBSE Curriculum 2015

"Open Teaching-Learning Material"

Visit www.ip4you.blogspot.com for more….

Authored By:- Rajesh Kumar Mishra, PGT (Comp.Sc.)
Kendriya Vidyalaya Upper Camp, Dehradun (Uttarakhand)
e-mail : rkmalld@gmail.com

What is Programming?

- Computer Programming is a process of designing, writing, testing, debugging and maintaining the source code written in a programming language to solve a real life problem.
- This Process requires knowledge of application domain (task), formal logic for solution and knowledge of syntax of the programming language, in which source code to be written.

There are two main approaches (methodologies) of programming-

1. Procedural Programming :

In this approach, a Programming task is broken into smaller and easily manageable modules (procedures). The stress is given to functionalities rather than data. Basic, COBOL, Pascal and C Languages supports Procedural Programming approach.

2. Object Oriented Programming:

In this approach, a programming task is broken into Objects, which contains data (properties) and its operating methods (behaviours). The main focus is given on Data rater than functions for better security and portability of data. C++, Java and other modern languages follows this methodology of programming.

What is Object Oriented Programming?

In general, Object Oriented Programming (OOP) refers '**Programming with objects**' , in which programming task is broken into objects.

The main features of OOP are:-

Encapsulation:

It refers the binding of Data and its associated Methods together in a single unit which is called Object.

Polymorphism:

A method can perform multiple functionalities (behaviour) depending on the input data. It is performed by Method Overloading and Operator Overloading.

Inheritance:

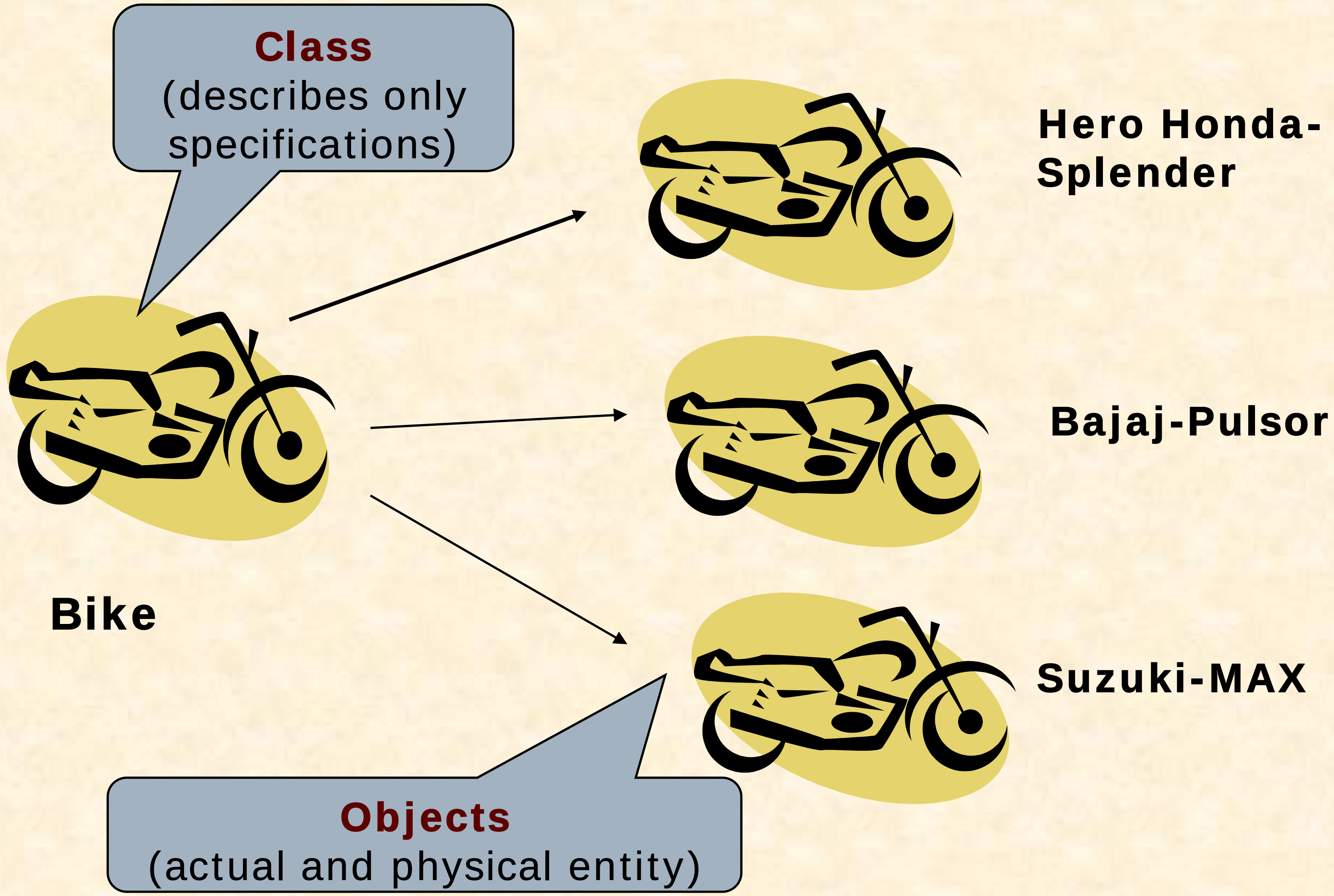
The process of deriving a new class (sub class) from existing classes (super class) is called Inheritance. The newly created class may contains properties and behaviour of its parent.

Object Oriented Programming in Java

JAVA is a pure Object Oriented Programming language, since each program in Java must contain at least one class definition.

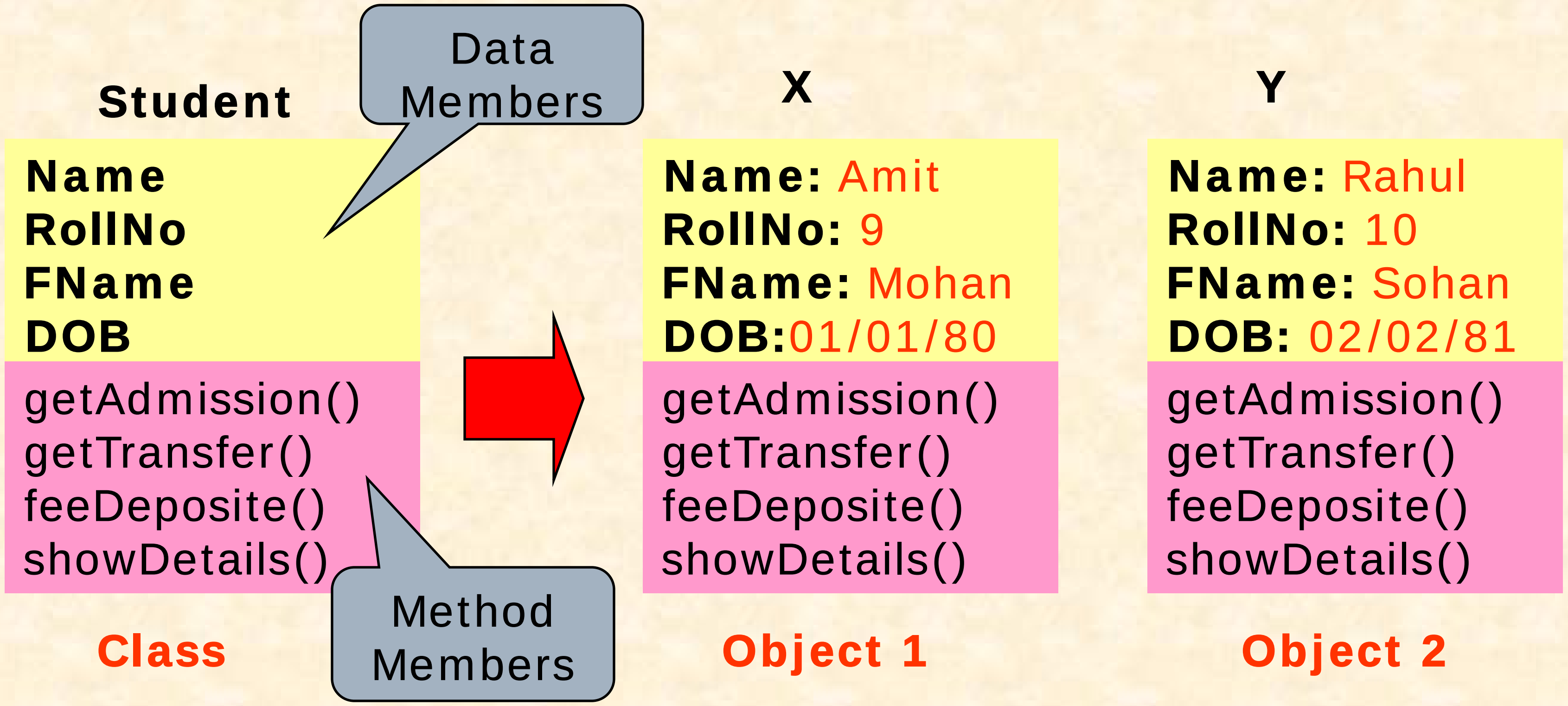
- The basic unit of OOP is the Class. A class represents a set of similar objects. It can be described as a <u>blue print of Objects</u>. In other words, an <u>Object is an instance of a class</u> that holds actual data in memory.
- An Object is an entity having a unique Identity, characteristics (Properties) and Behavior (Methods).
- JAVA is enriched with various ready-to-use class definitions, which are used in the Application. <u>Swing Controls are collection of such of classes.</u>
- For example, jButton control belongs to JButton Class of Swing Control Package. Each time, When you drag JButton on the Frame, an instance (Object) like jButton1, jButton2 etc. is created. Each object (jButton..) offers .setText(), .getText() etc. methods, which handles functionalities of the button.

Class and Objects



Data Member and Method Members

- A class is composed by a set of Attributes (Properties) and Behavior.
- Properties are represented by Data Members and Behavior is simulated by Method Members.



How to declare a Class in Java

```
public class Student
{ String Name;
  int RollNo;
  String FName;
  String DOB;
  void getAdmission()
   {……………………
    ……………………
   }
   void getTransfer()
   {……………………
    …………………….
   }
   void feeDeposit()
   { ………………….
     ………………….
   }
}
```

Data Members (Properties)

Method Members (Behavior)

In Java a Class is declared/defined by using class keyword followed by a class name.

Types of Members in a class

Data Members in a class may be categories into two types-

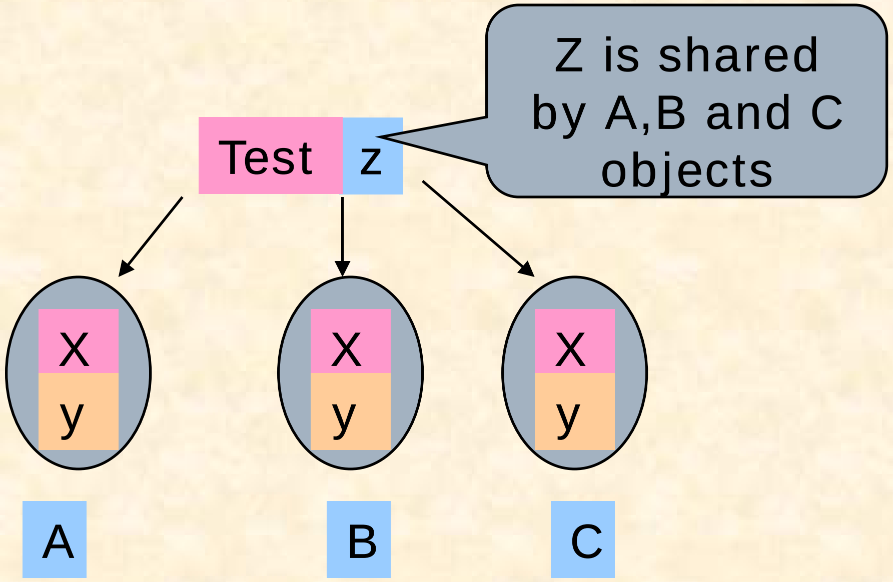
- **Instance Member (object Member)-**

  These member are created for every object of the class i.e. replicated with objects. They are declared without Static keyword.

- **Static Member (Class Member)-**

  These members that is declared once for each class and all objects share these members. Only a single copy is maintained in the memory. These are declared with static keyword.

```
public class test
{ int x;
  int y;
  static int z;
  ………………
}
```



How to create Objects

A class itself does not occupy any space, it defines a blue-print for the objects only. When object is instantiated then it occupies space in the memory. An object is created by the following steps-

- **Declaration:**

  Object is declared like an ordinary variables using class name as data type.

- **Instantiation:**

  An object is instantiated by using new keyword.

- **Initialization:**

  The **new** operator followed by a call to the constructor method (method having same name as class) is used to initialize and object.

```
student x;                 // x object is declared
x= new student();          // x is instantiated and initialized.
```

```
Student  x = new  student();    // all in a single step (alternate)
```

Creating Objects- An Example

```
/ / define a class
public class city
{ String name;
   long population;
   void display()
    { System.out.println(“City Name: ”+name);
       System.out.println(“Population: ”+population);
    }
}

/ / create an object and access it
private void jButton1ActionPerformed(…)
{ …………….
   city x= new city();    // creates an object x of class city
   x.name=“Jaipur”;       // Accessing data member of x
   x.population = 100000;
   System.out.println(“City Details”+’\n’);
   x.display();           // Accessing Method Member of x
 }
```

Understanding Methods

- A Method or function is sequence of statement which are written to perform a specific job in the application.
- In Object Oriented Programming, Method represents the behaviour of the object.
- You can create his own methods (User Defined Methods) apart from ready-to-use Library Methods in Java.

The following advantages describes that why we use methods.

☐ **To cope with complexity:**

When a programming task become more complex and big in size, it is broken into smaller and manageable module (Method). These modules can be easily designed and maintained.

☐ **Reusability of code:**

Once a method is implemented, it can be called from anywhere in the program when needed i.e. Method can be reused. This saves our time and effort.

Most of the library method like Math.sqrt() is available as ready to use methods which can be used anywhere in the application.

How to define a Method

A method must be defined before its use. The method always exist in a class. A Java Program must contain a main() method from where program execution starts. The general form of defining method is as-

```
[Access specifier]<return_type> <method_name>(<parameter(s)>)
        {……………. ;
          body of the method i.e. statement (s);
        }
```

- **Access Specifier:**

  It specified the access type and may be **public** or **protected** or **private**.

- **Return Type:**

  Specifies the return data type like **int**, **float** etc. **void** is used when nothing is to be returned.

- **Method Name:**

  Specified the name of method and must be a valid Java identifier.

- **Parameters List:**

  It is list of variable(s), also called Formal Parameter or Argument, which are used to catch the values when method is invoked. Also a method may have no parameters.

Working with Methods – An Example

- A method can be called by its name and parameters value (if any) in any other methods or Event method of jButtons.
- A method can be defined and called as –

```
// calling method/ function
private void jButton1ActionPerformed(.....)
{ int a,b,c,d;
   a=4;
   b=6;
   c=sum (a,b);    // function call
   d=sum(3,5);    // again function call
   jTextField1.setText(“”+c);
   System.out.println (“”+c);
   System.out.println(“”+d);
}
```

Actual parameters

Formal parameters

```
// called Method
int Sum (int x, int y)
{ int z= x+y;
   return(z);
}
```

Signature

```
int Sum (int x, int y)
```

Prototype

Note : The number of parameters and their data type must be matched during a method call and Formal Parameters must be variable.

What is Package ?

A group or collection of classes is called package.

- A Java a large program can be divided into smaller modules and compiled separately. Such modules are named Package.
- Each package may contain various class definitions under the same package name and stored in a folder.
- Java offers some ready-to-use packages as extensive library of pre-written classes like java.applet , java.io , java.lang , and java.net etc. java.io package. We can write our own packages also.
- We can use a package in a program by using import statement at top of the program.

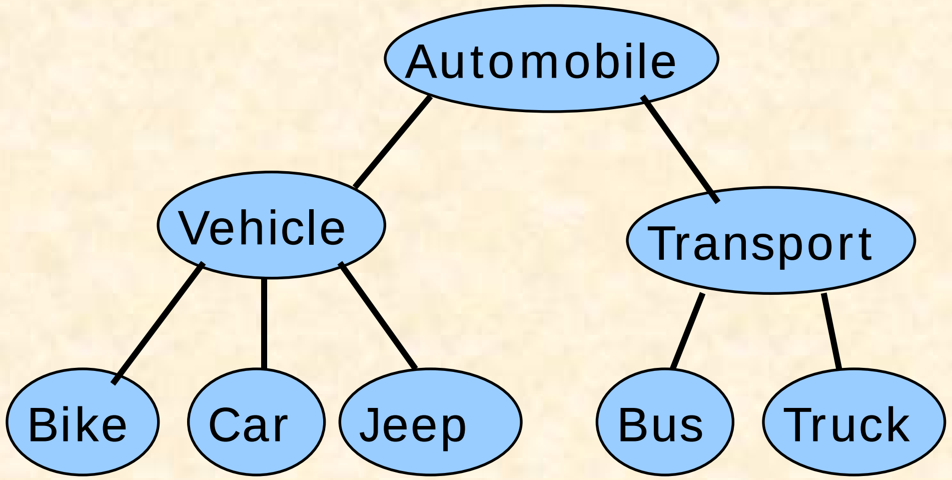
```
import javax.swing.JOptionPane;
import mypackage.*;              //to import all members//
import mypackage.myclass;  // to import selected class//
```

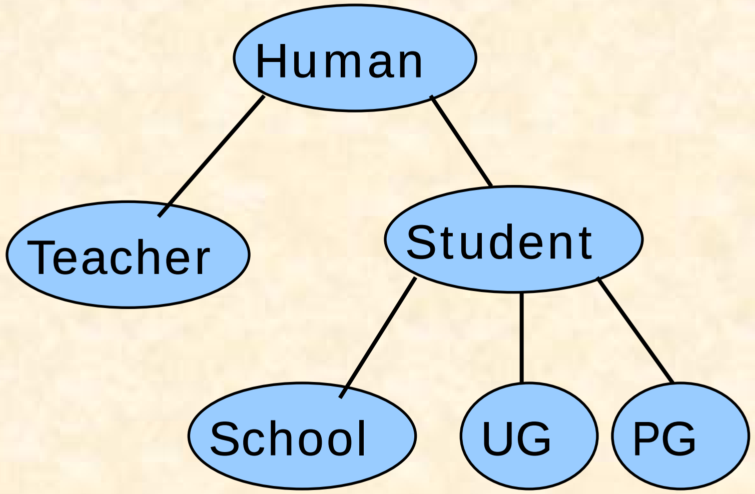
Note: java.lang package is imported by default i.e. no import statement is required.

Concept of Inheritance

- It is the process of creating new class (Derived Class or sub-classes) from existing class (Base Class or Super class).
- A Sub-class inherits all the properties (data members) and behaviour (method members) of the parent class (Super-class).
- The level of Inheritance may be extended i.e. A Sub-class may have their own sub-classes.

In Real-life most of the things exhibit Inheritance relationship.





Why Inheritance?

- **Modelling of Real-world:**

  By Inheritance, we can model the real-world inheritance relationships in easy way.

- **Reusability of codes:**

  Inheritance allows the derivation of a new class from existing classes. We can add new features to derived class to make a new one with minimal efforts.

- **Transitive nature of Inheritance:**

  If we modify the base class then the changes automatically inherit into derived classes. Since inheritance is transitive in nature. This offers faster and efficient way to maintain the large program.

How to derive a sub-class ?

In JAVA a new class (Sub-class) can derived from existing class (Super-class) by using **extends** keyword . A derived class may inherit all the data and method members from its parent. This principle known as Inheritance.

E.g. If human class is defined, we can derived student class by inheriting all the members of human class, since students are human beings.

```
//e.g. derivation of sub class//
class human
   { String name;
     int age;
     void method1()
     {……….}
   }
   class student extends human
    {int rollno;
     …………….
     void method2()
     {……….}
    }
```

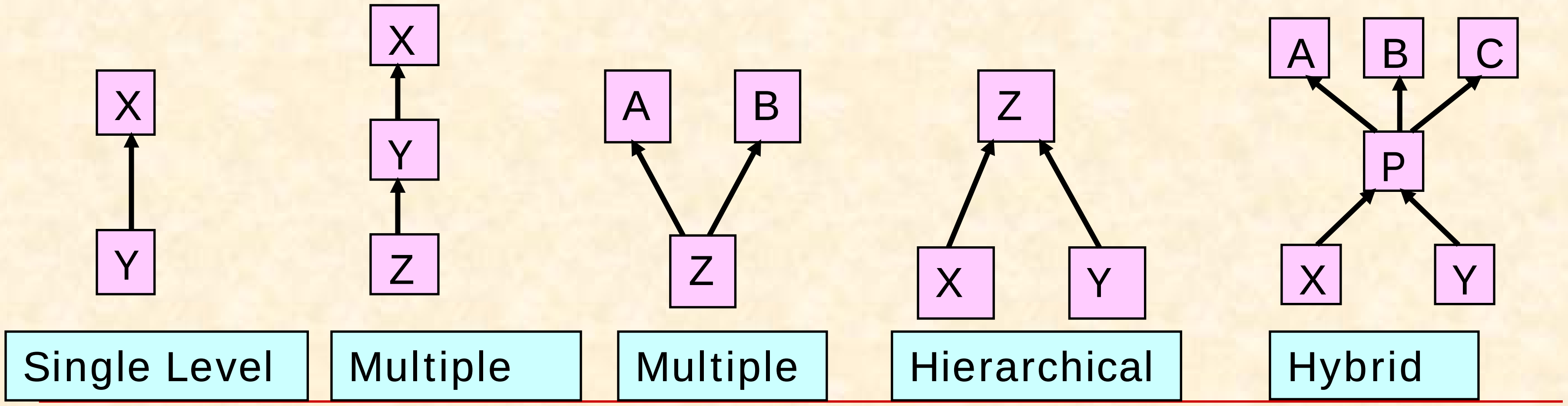
A sub-class may be derived in the same package or different package also. <u>The access of members of super class depends upon the Access Specifiers.</u>

Student class have name, age, rollno (data) and method1(), method2() as method members

Types of Inheritance

In OOP approach, the Inheritance are many types.

- Single Level Inheritance :
  When a sub-class inherits only one base-class.
- Multi-level Inheritance:
  When a sub-class is derived from sub-class of another base-class.
- Multiple Inheritance:
  When a sub-class derived from multiple base-classes.
- Hierarchical Inheritance:
  When Multiple sub-classes are derived from a single base class.
- Hybrid Inheritance: When a sub-class inherits from multiple base-classes and all its base-classes inherits one or more super-classes.



Access Specifiers for Class Members

The members (Data and Methods) of the Parent Class may be defined as Private, Public, Protected, Default, which may limit its accessibility or visibility in the derived (child) classes.

- **Private** : Members are accessible only inside their own class and no where else.
- **Protected**: Members are accessible in own class, all classes within package and all Sub-classes in different package.
- **Public**: Members are accessible everywhere in all the classes.
- **Package** (default): Members <u>without any specifier assumed package</u> level scope i.e. accessible to all classes inside the package only.

Type	**Inside own class**	**Inside Package**		**Outside Package**	
		Sub Class	**Non-Sub Class**	**Sub Class**	**Non-Sub Class**
Public	✓	✓	✓	✓	✓
Protected	✓	✓	✓	✓	✗
Package	✓	✓	✓	✗	✗
Private	✓	✗	✗	✗	✗

Abstract Classes

- An Abstract class simply represents a concept for its sub-classes.
- Sometimes, we need to define a super-class having general characteristics (data) and behaviour (generic methods) for its sub-classes.
- A Sub-class may re-define the methods or overridden to perform a task since super-class contains only the prototype (method with empty body).
- Abstract classes are normally used as base class in inheritance for which no object is required e.g. JOptionPane Class in Java is Abstract class because it requires no object. Whereas JTextField, JLabel classes etc. are called Concrete classes because they requires an object like ‘jTextField1’ for using them.
- The abstract keyword is used to define a such class.

Role of final keyword in Inheritance

The final keyword can be used with-

- ❖ **Variable**
- ❖ **Methods**
- ❖ **Class names**

The effect of final keywords is as follows.

- ☐ final variables works as constant i.e. the value of a final variables can’t be changed.
- ☐ final methods can’t be overridden by sub-class.
- ☐ final class can’t be extended.

Concept of Polymorphism

- In Simple term, Polymorphism means Multiple forms of behaviour. For example, a person may exhibit different behaviour in different places or situation.
- In Object Oriented Programming, a Method or Operator may exhibit different characteristics depending upon different sets of input given. Polymorphism makes your program code compact, smarter and faster.
- Polymorphism is implemented as Method Overloading and Operator Overloading i.e. overloaded with different functionalities.

e.g. Operator Overloading

e.g. In Java '+' operator is overloaded as-

2+3 gives 5 but "Hello"+"Java" gives "HelloJava"

Here, + operators exhibits different behaviour for numbers and string values i.e. + operator is overloaded.

Same as Math.round() function exhibit different behaviour for float and double type values i.e. Method Overloading.

Example : Method Overloading

Java allows functions to have same name but different in arguments (signature) in the same class.

```
class Test
  { int a;
    int area( int x, int y)
       { ………
         ………
       }
    int area( float a)
       { ………
         ………
       }
 …………
}
```

Two methods named **area** but different in arguments lists (signature) in the same class.

Two methods are considered Overloaded only if they are Same method name but Different in their Signature (i.e. either number of arguments or their type must be different)

A Method name having multiple definitions with different argument (s) is called **Overloaded Method**.