# PRACTICAL 2
# INHERITANCE AND POLYMORPHISM

**INHERITANCE:**

- ➔ Inheritance is an important pillar of OOP (Object-Oriented Programming). It is the mechanism in java by which one class is allowed to inherit the features (fields and methods) of another class.
- ➔ Important terminology:
  - **Super Class**: The class whose features are inherited is known as superclass (or a base class or a parent class).
  - **Sub Class**: The class that inherits the other class is known as a subclass (or a derived class, extended class, or child class). The subclass can add its own fields and methods in addition to the superclass fields and methods.
  - **Reusability**: Inheritance supports the concept of "reusability", i.e., when we want to create a new class and there is already a class that includes some of the code that we want, we can derive our new class from the existing class. By doing this, we are reusing the fields and methods of the existing class.
- ➔ Types on inheritance:
  - Single
    - When a class inherits another class, it is known as a single inheritance. In the example given below, Dog class inherits the Animal class, so there is the single inheritance.
  - Multilevel
    - When there is a chain of inheritance, it is known as multilevel inheritance. As you can see in the example given below, BabyDog class inherits the Dog class which again inherits the Animal class, so there is a multilevel inheritance.
  - Hierarchical
    - When two or more classes inherits a single class, it is known as hierarchical inheritance. In the example given below, Dog and Cat classes inherits the Animal class, so there is hierarchical inheritance.
  - Multiple
    - To reduce the complexity and simplify the language, multiple inheritance is not supported in java.
    - To fill this gap we have interface
    - The interface in Java is a mechanism to achieve abstraction. There can be only abstract methods in the Java interface, not method body. It is used to achieve abstraction and multiple inheritance in Java.

```
interface <interface_name>{

    // declare constant fields
    // declare methods that abstract
    // by default.
}
```

- Hybrid
  - Hybrid inheritance in Java is a combination of two or more types of inheritances. The purpose of using hybrid inheritance in Java is to modularize the codebase into well-defined classes and provide code reusability.
  -

➔ **Syntax**:

- class derived-class extends base-class
- {
-  //methods and fields
- }

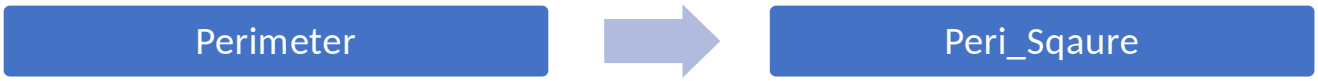
**POLYMORPHISM**

➔ The word polymorphism means having many forms. In simple words, we can define polymorphism as the ability of a message to be displayed in more than one form.

➔ In Java polymorphism is mainly divided into two types:

- Compile time polymorphism
  - It is also known as static polymorphism. This type of polymorphism is achieved by function overloading or operator overloading.
- Runtime polymorphism
  - It is also known as Dynamic Method Dispatch. It is a process in which a function call to the overridden method is resolved at Runtime. This type of polymorphism is achieved by Method Overriding.

**Q1.**




```
package practical2;

class Perimeter{

   public void Perimeter(){
      System.out.println("Perimeters will be printed: ");
   }
}

public class Peri_Square extends Perimeter{

   public void Peri_Square(){
      int a = 5;
      int peri_sq = a*4;
      System.out.println("Perimeter of the square is " + peri_sq);
   }

   public static void main(String[] args) {
      Peri_Square p1=new Peri_Square();
      p1.Perimeter();
      p1.Peri_Square();
   }

}
```

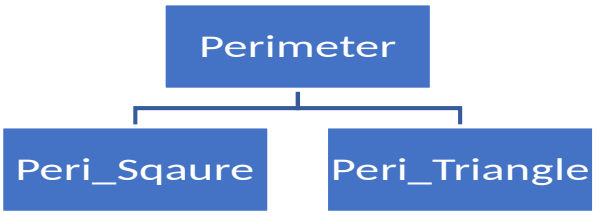
```
run:
Perimeters will be printed:
Perimeter of the square is 20
BUILD SUCCESSFUL (total time: 0 seconds)
```

Q2.


```
package practical2;

class Perimeter{
   public void Perimeter(){
      System.out.println("Perimeters will be printed: ");
   }
}

class Peri_Triangle extends Perimeter{
   public void Peri_Triangle(){
      int a = 5, b = 5, c = 6;
      int peri_tri = a + b + c;
      System.out.println("Perimeter of the triangle is " + peri_tri);
   }
}

public class Peri_Square extends Peri_Triangle {
   public void Peri_Square_2(){
      int a = 5;
      int peri_sq = a*4;
      System.out.println("Perimeter of the square is " + peri_sq);
   }

   public static void main(String[] args){
      Peri_Square p1 = new Peri_Square();
      p1.Perimeter();
      p1.Peri_Triangle();
      p1.Peri_Square();
   }
}
```

```
run:
Perimeters will be printed:
Perimeter of the triangle is 16
Perimeter of the square is 20
BUILD SUCCESSFUL (total time: 0 seconds)
```

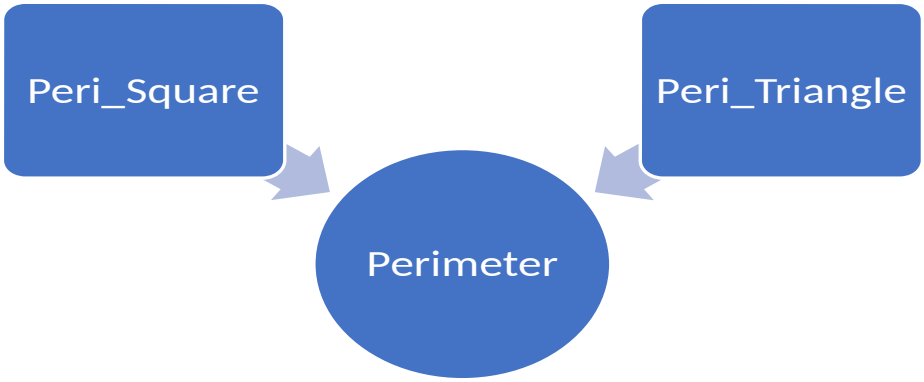
Q3.




```
package practical2;

class Perimeter{
   public void Perimeter(){
      System.out.println("Perimeters will be printed: ");
   }
}

class Peri_Triangle extends Perimeter{
   public void Peri_Triangle(){
      int a = 5, b = 5, c = 6;
      int peri_tri = a + b + c;
      System.out.println("Perimeter of the triangle is " + peri_tri);
   }
}

public class Peri_Square extends Perimeter {
   public void Peri_Square_3(){
      int a = 5;
      int peri_sq = a*4;
      System.out.println("Perimeter of the square is " + peri_sq);
   }

   public static void main(String[] args){
      Peri_Square ps = new Peri_Square();
      Peri_Triangle pt = new Peri_Triangle();

      ps.Perimeter();
      ps.Peri_Square();

      pt.Perimeter();
      pt.Peri_Triangle();
   }
}
```

```
run:
Perimeters will be printed:
Perimeter of the square is 20
Perimeters will be printed:
Perimeter of the triangle is 16
BUILD SUCCESSFUL (total time: 0 seconds)
```

**Q4.**




```
package practical2;

interface Peri_Square{
    default void Peri_Square(){
        int a = 5;
        int peri_sq = a*4;
        System.out.println("Perimeter of the square is " + peri_sq);
    }
}

interface Peri_Triangle{
    default void Peri_Triangle(){
        int a = 5, b = 5, c = 6;
        int peri_tri = a + b + c;
        System.out.println("Perimeter of the triangle is " + peri_tri);
    }
}

public class Perimeter implements Peri_Square, Peri_Triangle{
    public void PeriMeter(){
    System.out.println("Perimeters will be printed: ");
    Peri_Square.super.Peri_Square();
    Peri_Triangle.super.Peri_Triangle();
    }

    public static void main(String[] args){
        Perimeter p = new Perimeter();
        p.PeriMeter();
    }
}
```

```
run:
Perimeters will be printed:
Perimeter of the square is 20
Perimeter of the triangle is 16
BUILD SUCCESSFUL (total time: 0 seconds)
```

**Q5. Write a java program to create class Area and method named as calarea() to find area of a square and rectangle using method overloading.**

```
package practical2;

public class Area {
   void calarea(int a){
      int area = a*a;
      System.out.println("Area of Square is: " + area);
   }

   void calarea(int a, int b){
      int area = a*b;
      System.out.println("Area of Rectangle is: " + area);
   }

   public static void main(String[] args){
      Area a1 = new Area();
      a1.calarea(5);
      a1.calarea(4, 5);
   }
}
```

```
run:
Area of Square is: 25
Area of Rectangle is: 20
BUILD SUCCESSFUL (total time: 0 seconds)
```

**Q6. Write a java program to create class Calculator and method named as add()-two number and create another class Addition who inherit class Calculator and method named as add()-three number using overriding.**

```
package practical2;

class Calculator{
   void add(){
      int a, b, sum;
      a = 5;
      b = 5;
      sum = a+b;
      System.out.println("Addition of two numbers is: " + sum);
   }
}

public class Addition extends Calculator {
   void add(){
      int a, b, c, sum;
      a = 5;
      b = 5;
      c = 5;
      sum = a + b + c;
      System.out.println("Addition of three numbers is: " + sum);
   }

   public static void main(String[] args){
      Calculator c1 = new Calculator();
      c1.add();
      c1 = new Addition();
      c1.add();
   }
}
```

```
run:
Addition of two numbers is: 10
Addition of three numbers is: 15
BUILD SUCCESSFUL (total time: 0 seconds)
```