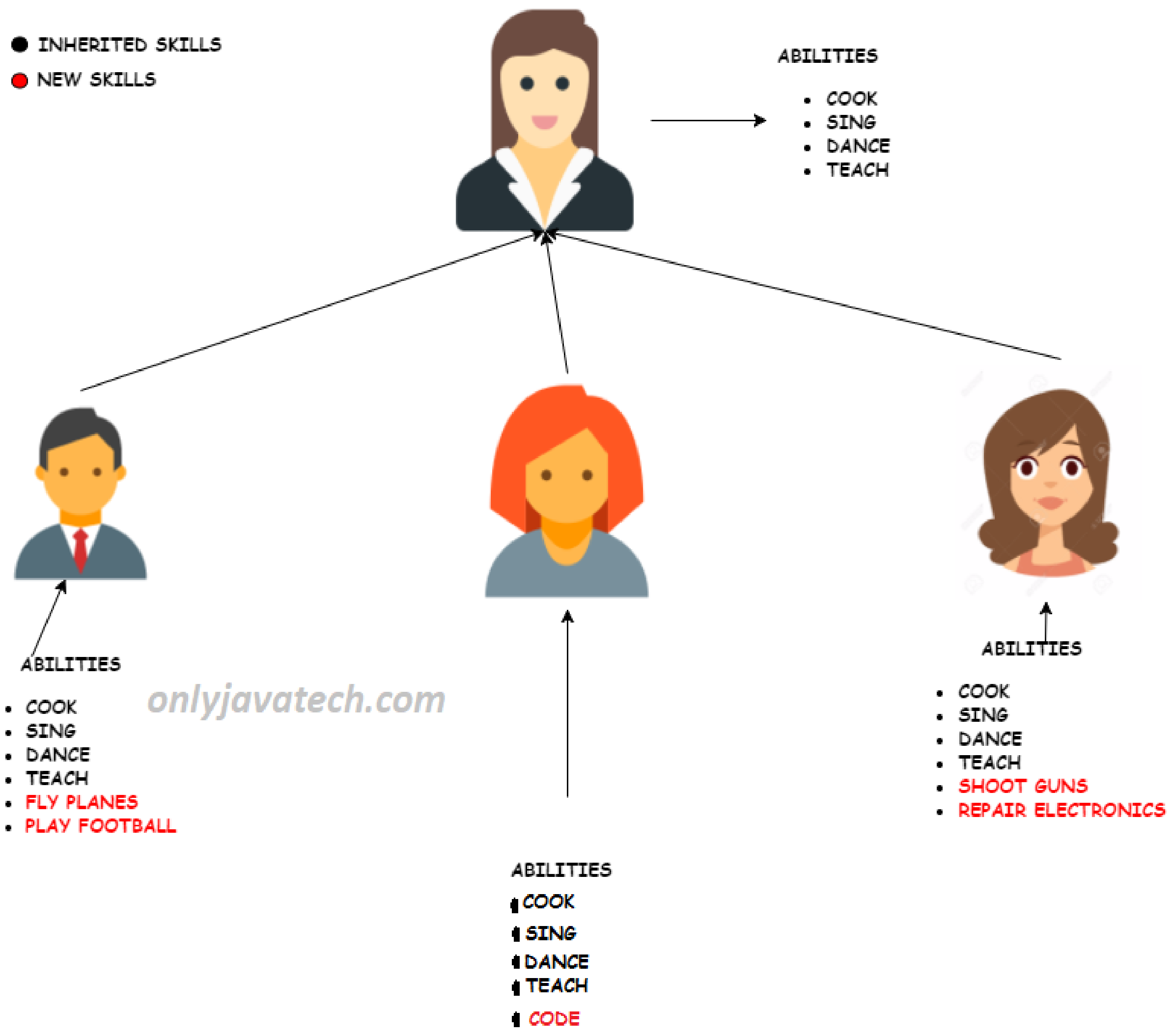
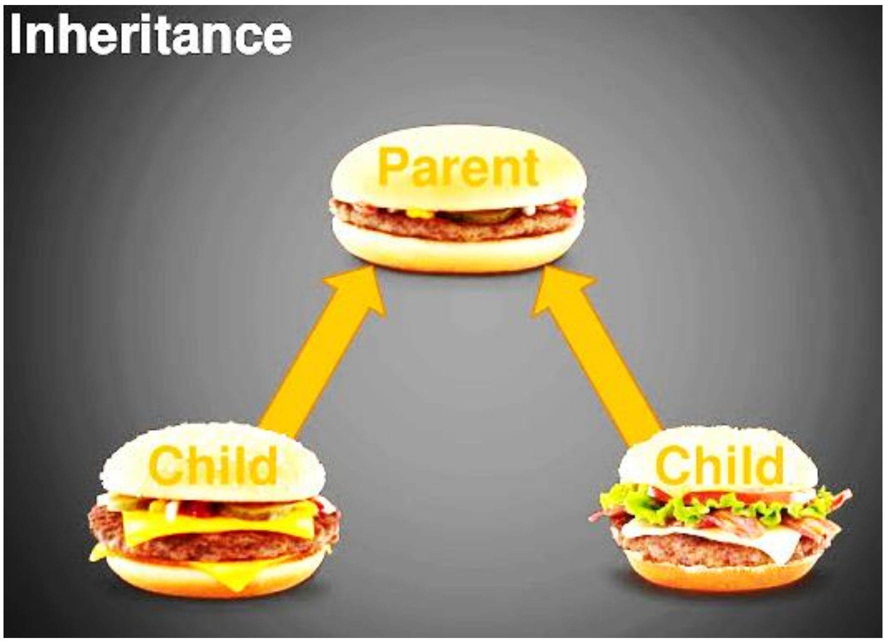
Inheritance in OOP – Java

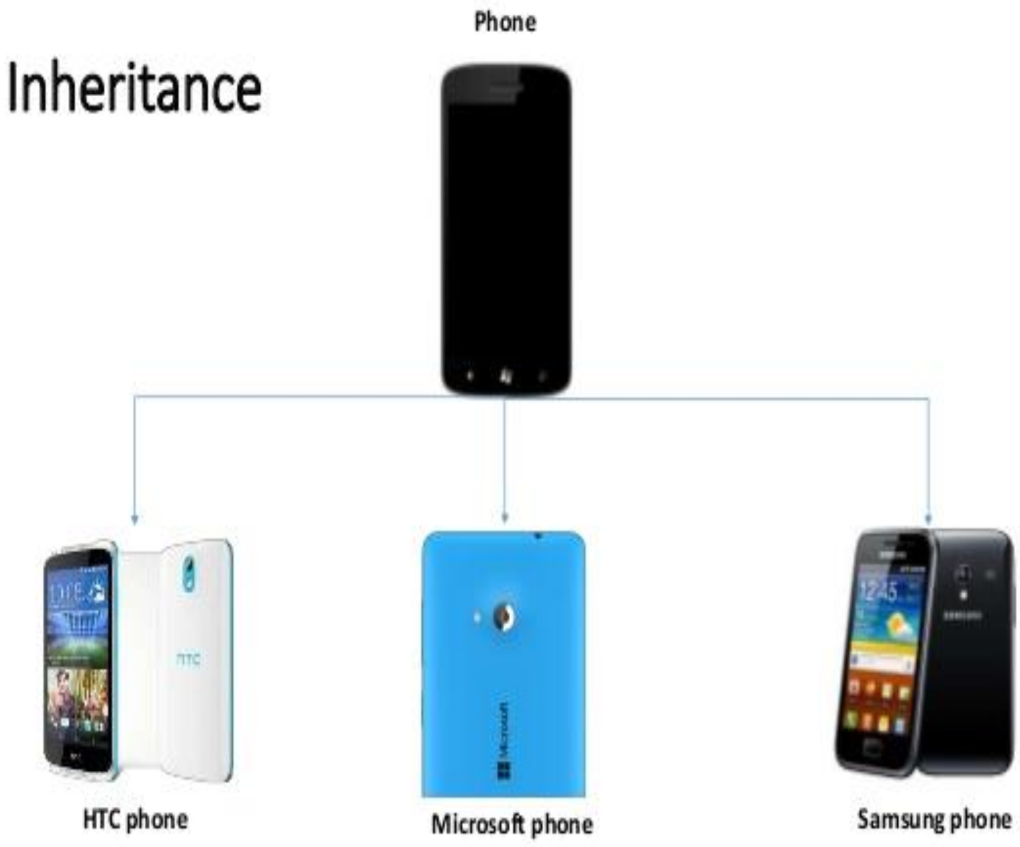
Inheritance is a mechanism in which one class acquires (يكتسب) the property of another class. For example, a child inherits the traits (سمات) of his/her parents.
Inheritance is an important pillar (دعامة) and the most powerful mechanisms of OOP. The inheritance mechanism is allowing a class to inherit the features (fields and methods) of another class.

Inheritance represents the **IS-A relationship,** also known as **parent-child relationship.**

It allows the **reuse** of the members of a class (called the superclass or the mother class) in another class (called subclass, child class or the derived class) that inherits from it. Below three visual examples of inheritance from Real World.




Inheritance
Parent
Child
Child


Inheritance
Phone
HTC phone
Microsoft phone
Samsung phone

Important terminology:

- **Super Class:** The class whose features are inherited is known as super class(or a base class or a parent class).
- **Sub Class:** The class that inherits the other class is known as sub class(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.

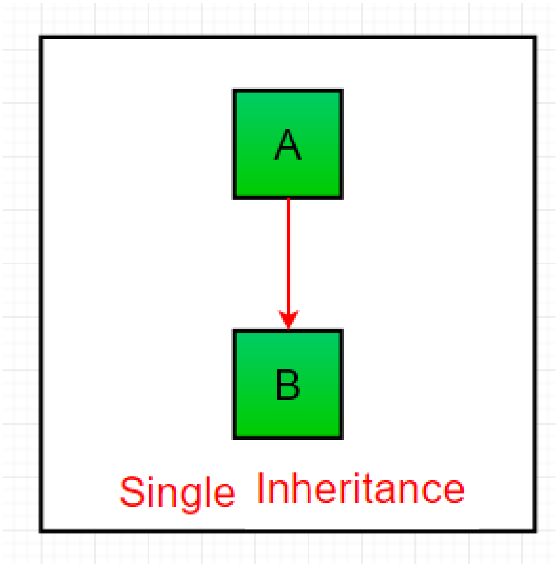
The keyword used for inheritance is **extends**. The syntax of inheritance in Java language is:

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

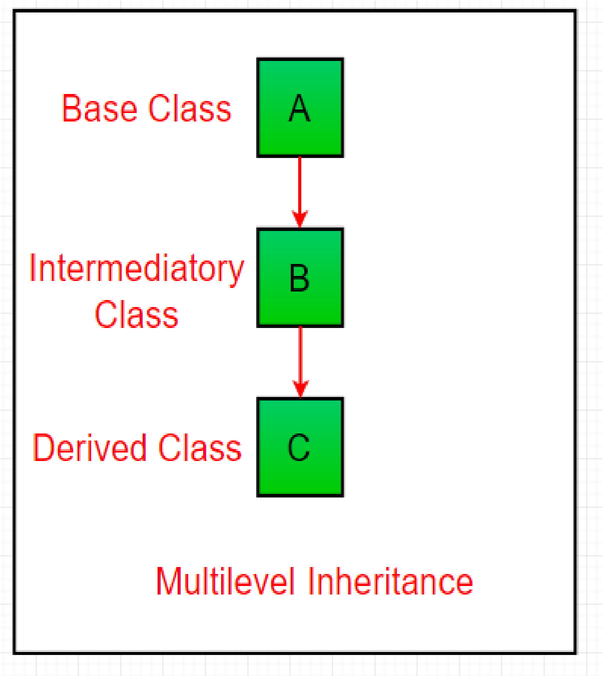
Types of Inheritance in Java

Below are the different types of inheritance which is supported by Java.

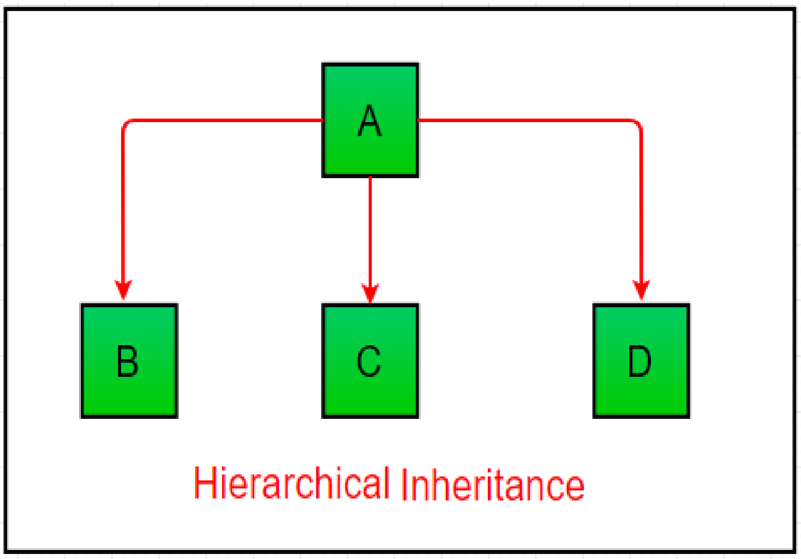
1. **Single Inheritance :** In single inheritance, subclasses inherit the features of one superclass. In image below, the class A serves as a base class for the derived class B.



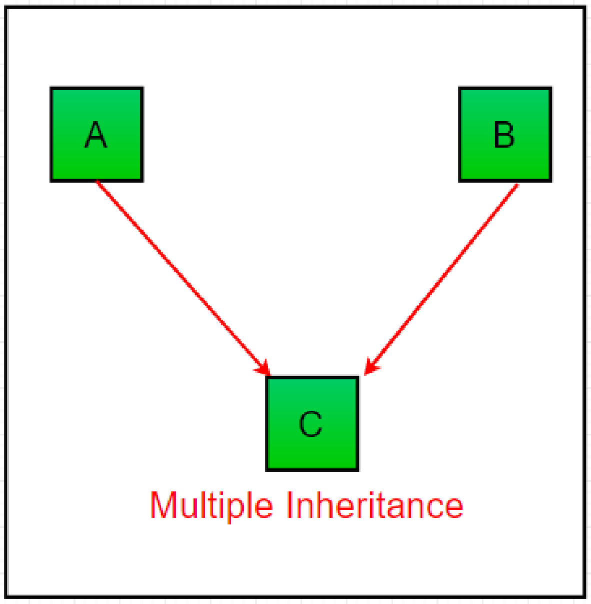
2. **Multilevel Inheritance:** In Multilevel Inheritance, a derived class will be inheriting a base class and as well as the derived class also act as the base class to other class. In below image, the class A serves as a base class for the derived class B, which in turn serves as a base class for the derived class C. In Java, a class cannot directly access the grandparent's members.



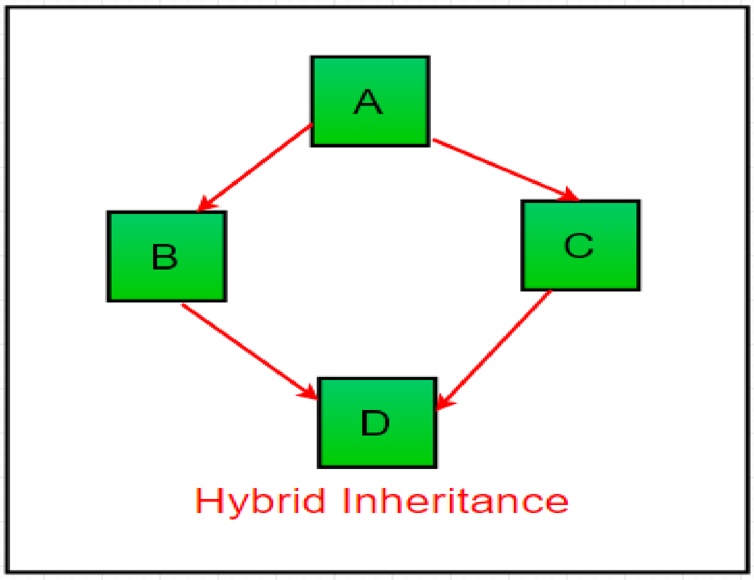
3. **Hierarchical Inheritance:** In Hierarchical Inheritance, one class serves as a superclass (base class) for more than one sub class.In below image, the class A serves as a base class for the derived class B,C and D.



4. **Multiple Inheritance (Through Interfaces):** In Multiple inheritance ,one class can have more than one superclass and inherit features from all parent classes. Please note that Java does not support multiple inheritance with classes. In java, we can achieve multiple inheritance only through Interfaces. In image below, Class C is derived from interface A and B.



5. **Hybrid Inheritance (Through Interfaces)** : It is a mix of two or more of the above types of inheritance. Since java doesn't support multiple inheritance with classes, the hybrid inheritance is also not possible with classes. In java, we can achieve hybrid inheritance only through Interfaces.



Important facts about inheritance in Java

- **Default superclass**: Except Object class, which has no superclass, every class has one and only one direct superclass (single inheritance). In the absence of any other explicit superclass, every class is implicitly a subclass of Object class.
- **Superclass can only be one:** A superclass can have any number of subclasses. But a subclass can have only **one** superclass. This is because Java does not support multiple inheritance with classes. Although with interfaces, multiple inheritance is supported by java.
- **Inheriting Constructors:** A subclass inherits all the members (fields, methods, and nested classes) from its superclass. **Constructors are not members**, so they are not inherited by subclasses, but the constructor of the superclass can be invoked (تستدعى)from the subclass.
- **Private member inheritance:** A subclass does not inherit the private members of its parent class. However, if the superclass has public or protected methods (like get and set) for accessing its private fields, these can also be used by the subclass.

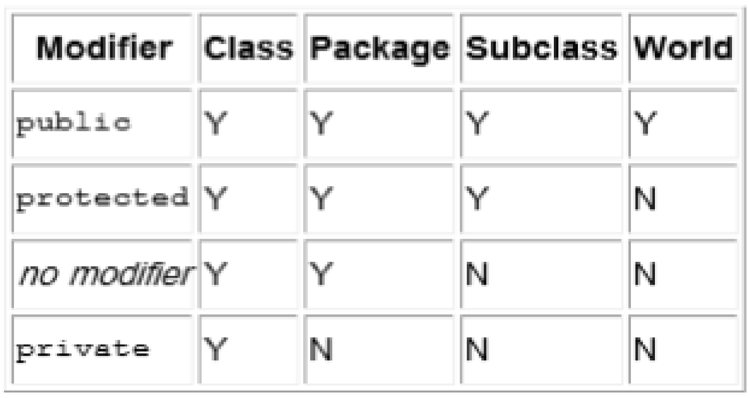
Controlling Access to Members of a Class

Access level modifiers determine whether other classes can use a particular field or invoke a particular method. There are two levels of access control:

- At the top level—public, or package-private (no explicit modifier).
- At the member level—public, private, protected, or package-private (no explicit modifier).

The following table shows the access to members permitted by each modifier.

Access Levels



Modifier	Class	Package	Subclass	World
`public`	Y	Y	Y	Y
`protected`	Y	Y	Y	N
no modifier	Y	Y	N	N
`private`	Y	N	N	N

Example1 (Trace)

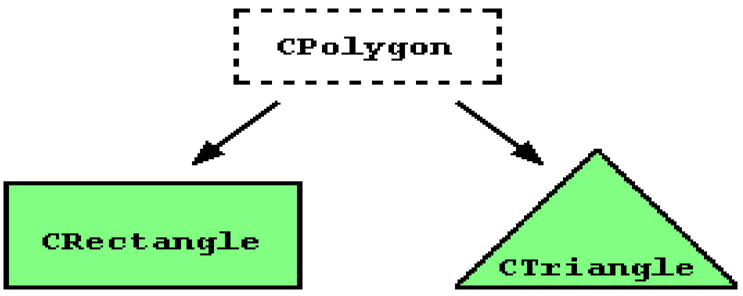
```
class Teacher {
 public   String designation = "Teacher";
  public String collegeName = "Beginners book";
  public void does(){
          System.out.println("Teaching");  } }

public class PhysicsTeacher extends Teacher{
  public String mainSubject = "Physics";}

public class Main{
   public static void main(String args[]){
          PhysicsTeacher obj = new PhysicsTeacher();
          System.out.println(obj.collegeName);       System.out.println(obj.designation);
          System.out.println(obj.mainSubject);       obj.does(); }}
```

Output:

```
Beginners book
Teacher
Physics
Teaching
```

We are going to suppose that we want to declare a series of classes that describe polygons like our **CRectangle**, or **CTriangle**. They have certain common features, such as both can be described by means of only two sides: height and base. This could be represented in the world of classes with a class **CPolygon** from which we would derive the two referred ones, **CRectangle** and **CTriangle**.

Base class / super class



Derived classes

The class **CPolygon** would contain members that are common for all polygons. In our case: **width** and **height**. And **CRectangle** and **CTriangle** would be its derived classes.

Example (Writing a program)

Define a base class called polygon. Use it to store two integer type values by method set values ; that could be used to compute the area of figure. Derive two specific classes called triangle and rectangle from the base polygon. Add to derived class a member function area ( ) to compute the area of figures.

```
package javaapplication45;
public class Polygon {

  protected int width, height;
  public void set_values (int a, int b)
     { width=a; height=b;}
  }


package javaapplication45;
public class Rectangle extends Polygon{

  public    int area (){
      return (width * height); }   }


package javaapplication45;
public class Triangle extends Polygon {
  public  int area (){
      return (width * height / 2); }   }


package javaapplication45;
 public class Main {
       public static void main(String[] args) {
       Rectangle rect=new Rectangle();
       Triangle trg=new Triangle();
  rect.set_values (4,5);
  trg.set_values (4,5);
  System.out.println(rect.area());
  System.out.println(trg.area()); }}
```

The output :

```
20
10
```

As you may see, objects of classes Rectangle and Triangle each contain members of Polygon, that are: width, height and set_values().

The protected specifier is similar to private, its only difference occurs when deriving classes. When we derive a class, protected members of the base class can be used by other members of the derived class, nevertheless private member cannot.

Since we wanted width and height to have the ability to be manipulated by members of the derived classes Rectangle and Triangle and not only by members of Polygon, we have used protected access instead of private.