Inheritance in Java

(Java OOPs Concepts)

Introduction

The word Inheritance is quite familiar with everyone. In common terms, the word means the bequeathing of property and characteristics from generation to generation. For example, the property or characteristics of parents are handed down to their children and the forthcoming generations.

Object Oriented Programming (commonly **OOP**) concepts are based on real life examples, where every entity in existence can be represented as an object. Thus, being one of the fundamental concepts in OOP, Inheritance is based on the example we discussed earlier – bequeathing of properties and characteristics from parents to their children.
Inheritance in Java, as derived from the concept of OOP, is defined as the process by which a child class or object (known as subclass) inherits the behaviors and properties(methods and variables) from its predecessors or parent class(known as **super class**). Let us delve a little deeper into the concepts of Inheritance in java with the following sections.

Basic Syntax

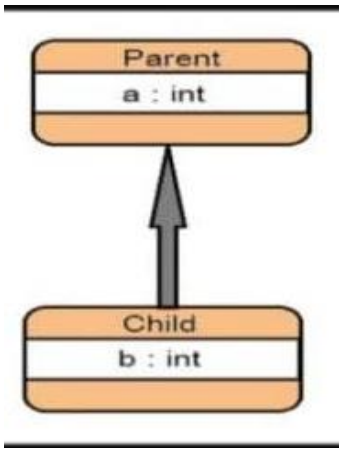
Inheritance in Java is implemented by the use of the keyword extends. This special word makes the Java compiler understand that the current class is

inheriting or extending another class. Let us look at the following snippet example to understand the basic syntax.

```
1
2 public class B extends A {
3
4 // some process or variables
5
6
7 }
```

The above snippet shows the use of the extends keyword. In the language of Java, the use of ‘extends’ indicates that the class B is a child or a subclass of the class A, which is known as the **super** class or parent. Inheritance represents an IS-A relationship between classes. Here, B is a child or subclass of A.

Example of Inheritance in Java



The picture given alongside displays a simple representation of inheritance in Java.

Here, the class Parent contains an integer **variable a** and is a super-class to

class Child which contains an integer **variable b**

Let us see the representation of this picture by means of a code example.

```
class Parent {

 int a = 5;
}

class Child extends Parent {
 int b= 7;
 public static void main(String args[]){
  Child c = new Child();
  System.out.println("Value of A ="+c.a);
  System.out.println("Value of B ="+c.b);
  System.out.println("Sum="+(c.a+c.b));
 }
}
```

The above code on execution, provides the following result:

Value of A =5
Value of B =7
Sum=12

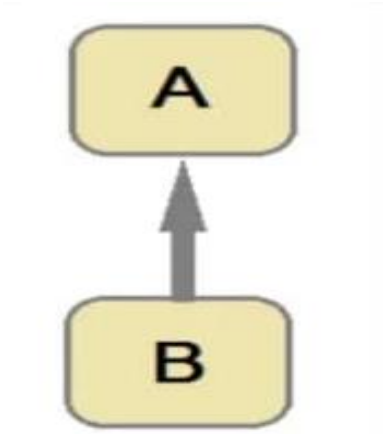
From the above example we see that the child class B is able to access the variable a of class Parent and use it in its own class. Thus we see that using inheritance, sub-classes may utilize variables and methods from their super-classes.

Types of Inheritance in Java

There are various forms of representing a Parent-Child relationship in Object Oriented Programming. Java supports three types of Inheritance on the basis of classes. The various types of inheritance shall be discussed further in this section and how they are realized using Java.

Single Inheritance

This is the simplest form of inheritance in Java and is a simple ONE to ONE relationship between two classes. A basic example of single inheritance has already been discussed in the section above, where a single Child class had inherited the attributes of its Parent class.



The picture given above represents a single inheritance between classes A(**super-class**) and B(**sub-class**)
Let us see a small code below as an example of Single Inheritance.
Code:

```
1
2  class MessageWriter {
3     public void display(){
4     System.out.println("You are viewing a method of A");
5     }
6
7  }
8
9  class MessageDisplayer extends MessageWriter {
```

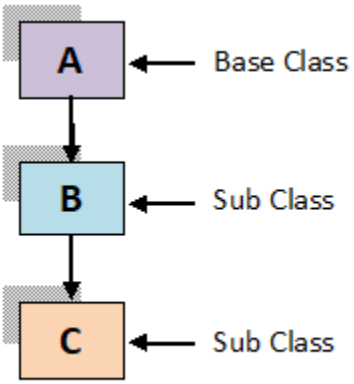
```
10      public static void main(String args[]) {
11          B obj=new B();
12          obj.display();
13      }
14  }
```

The above code executes to produce the following output:

You are viewing a method of A

Multi-Level Inheritance

Multi-Level Inheritance as its name suggests looks more like a tier-based structure. The picture given alongside represents a multi-level inheritance structure. In Multi-Level Inheritance, each class extends only a single class in the form of a multi-level or multi-tiered architecture.



For instance, from the figure alongside, we see that class C is a sub-class of class B and class B is a sub-class of class A. Thus, B will inherit the attributes and behavior of class A and C, in turn, will inherit from both.
Let us see a code snippet as an example of Multi-Level Inheritance.

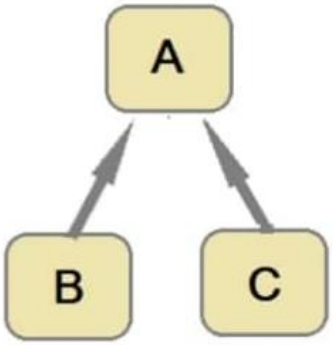
```
class Animal {
   void eat(){
     System.out.println(“Animals eat”);
  }
}

class Horse extends Animal {
  void call(){
    System.out.println(“Horses neigh”);
  }
}

class Colt extends Horse {
  void desc(){
    System.out.println(“Colts are baby horses”);
  }
}

class TestMultiLevel {
   public static void main(String args[]){
      Colt c = new Colt();
      c.desc();
      c.call();
      c.eat();
   }
}
```

On executing the above code, the following output is obtained:

Colts are baby horses
Horses neigh
Animals eat

Hierarchical Inheritance

Hierarchical Inheritance is a type of inheritance where many sub-classes extend a single super-class.



The picture given alongside represents a basic form of Hierarchical Inheritance. This form of inheritance is basically multiple single inheritances where all sub classes inherit from a single super class. Here, classes B and C are sub-classes of class A and inherit its attributes and behavior.
Let us see a code below as an example of Hierarchical Inheritance.
Code:

```
1
2  class Animal {
3    void eat(){
4      System.out.println(“Animals eat”);
5   }
6  }
7
```

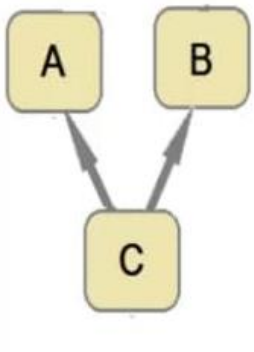
```
class Lion extends Animal {
   void call(){
   System.out.println(“Lions roar”);
  }
}

class Wolf extends Animal {
   void call(){
    System.out.println(“Wolves howl”);
 }
}

class TestHierarchy{
public static void main(String args[]){
     Wolf w = new Wolf();
     Lion l=new Lion();
     w.call();
     w.eat();
    l.call();
    l.eat();
  }
}
```

The above code, on execution, produces the following output:

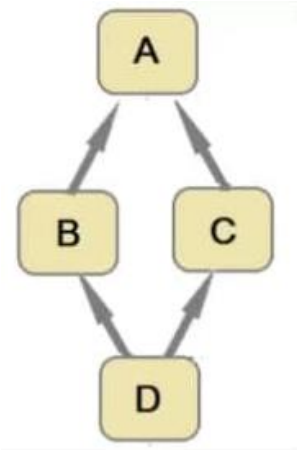
Wolves howl
Animals eat
Lions roar
Animals eat

Multiple and Hybrid Inheritance

Multiple Inheritance is a type of Inheritance in Object Oriented Programming where a single sub-class extends multiple super-classes. Hybrid Inheritance is a mixed form of inheritance comprising of Multiple and Multi-Level Inheritance. Multiple Inheritance is not supported by **Java**.



Above picture shows Multiple Inheritance



This picture depicts Hybrid Inheritance.

Why does Java not support Multiple Inheritance?

To reduce complexity and ambiguity, Java does not allow Multiple Inheritance using classes. Let us explain the ambiguity with an example.
Say, there is a class A having a method display() and class B having another method display(). A third class C, extends both A and B and as per the concept of inheritance, is able to access both the display() methods of A and B. Now, when creating an object of C and calling the display() method, the compiler will not be able to discern which display method to call as both classes A and B have a method having the same name and are extended by class C.

Code:

```
Class A (){
    public void display ()
    {
     //some code
    }
}
class B (){
  public void display ()
  {
   //some code
  }
}
class C extends A,B () ( Not supported in Java ){
public static void main (){
  C obj = new C ();
  obj.display(); ( Ambiguity between display() of class A and display of class B)
 }
}
```

In order to prevent this ambiguity, Java does not allow multiple inheritance between classes and throws a compile-time error when multiple inheritance is attempted.
Note: Java, however, simulates Multiple Inheritance by using **Interfaces**.

Conclusion

Inheritance is a strong weapon of Java that helps to make it a widely acceptable language. It helps to reduce code duplication and also cuts down on the bugs. With the code written in the parent class, you no longer need to write the same code for multiple child classes that has the same properties. In this way, inheritance in java implements code reusability to ensure better accessibility to users.

That’s all about inheritance in java.