Comparison of Inheritance in C++ and Java

Difficulty Level : Easy Last Updated : 22 Nov, 2021

The purpose of inheritance is the same in C++ and Java. Inheritance is used in both languages for reusing code and/or creating an 'is-a' relationship. The following examples will demonstrate the differences between Java and C++ that provide support for inheritance.

1) In Java, all classes inherit from the Object class directly or indirectly. Therefore, there is always a single inheritance tree of classes in Java, and the Object Class is the root of the tree. In Java, when creating a class it automatically inherits from the Object Class. In C++ however, there is a forest of classes; when we create a class that doesn't inherit from another, we create a new tree in a forest.

Following the Java example shows that the **Test class** automatically inherits from the Object class.

```
class Test {
    // members of test
}
class Main {
    public static void main(String[] args)
    {
        Test t = new Test();
        System.out.println("t is instanceof Object: "
                           + (t instanceof Object));
    }
}
```

Output

```
t is instanceof Object: true
```

2) In Java, members of the grandparent class are not directly accessible. (Refer to **this** article for more details).

3) The meaning of protected member access specifier is somewhat different in Java. In Java, protected members of a class “A” are accessible in other class “B” of the same package, even if B doesn’t inherit from A (they both have to be in the same package).

For example, in the following program, protected members of A are accessible in B.

```
class A {
    protected int x = 10, y = 20;
}

class B {
    public static void main(String args[])
    {
        A a = new A();
        System.out.println(a.x + " " + a.y);
    }
}
```

Output

```
10 20
```

4) Java uses ‘extends’ keywords for inheritance. Unlike C++, Java doesn’t provide an inheritance specifier like public, protected, or private. Therefore, we cannot change the protection level of members of the base class in Java, if some data member is public or protected in the base class then it remains public or protected in the derived class. Like C++, private members of a base class are not accessible in a derived class.

Unlike C++, in Java, we don’t have to remember those rules of inheritance which are a combination of base class access specifier and inheritance specifier.

5) In Java, methods are virtual by default. In C++, we explicitly use virtual keywords (Refer to **this** article for more details).

6) Java uses a separate keyword *interface* for interfaces and *abstract* keywords for abstract classes and abstract functions.

Following is a Java abstract class example,

```
// An abstract class example
abstract class myAbstractClass {

    // An abstract method
    abstract void myAbstractFun();

    // A normal method
    void fun() { System.out.println("Inside My fun"); }
}

public class myClass extends myAbstractClass {
    public void myAbstractFun()
    {
        System.out.println("Inside My fun");
    }
}
```

Following is a Java interface example,

```
// An interface example
public interface myInterface {

    // myAbstractFun() is public
     // and abstract, even if we
    // don't use these keywords
    void myAbstractFun();
    // is same as public abstract void myAbstractFun()
}

// Note the implements keyword also.
public class myClass implements myInterface {
    public void myAbstractFun()
    {
        System.out.println("Inside My fun");
    }
}
```

7) Unlike C++, Java doesn't support multiple inheritances. A class cannot inherit from more than one class. However, A class can implement multiple interfaces.

8) In C++, the default constructor of the parent class is automatically called, but if we want to call a parameterized constructor of a parent class, we must use the Initializer list. Like C++, the default constructor of the parent class is automatically called in Java, but if we want to call parameterized constructor then we must use super to call the parent constructor. See the following Java example.

```
package main;

class Base {
    private int b;
    Base(int x)
    {
        b = x;
        System.out.println("Base constructor called");
    }
}

class Derived extends Base {
    private int d;
    Derived(int x, int y)
    {
        // Calling parent class parameterized constructor
        // Call to parent constructor must be the first line
        // in a Derived class
        super(x);
        d = y;
        System.out.println("Derived constructor called");
    }
}

class Main {
    public static void main(String[] args)
    {
        Derived obj = new Derived(1, 2);
    }
}
```

Output

```
Base constructor called
Derived constructor called
```

Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above.