Comparison of Inheritance in C++ and Java

- 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 Object class is root of the tree. In Java, when create 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 forest.

```
//This is present in the different file named MyClass.java
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 will be

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

- In Java, members of the grandparent class are not directly accessible.

```
// filename Main.java
class Grandparent {
    public void Print()
    {
        System.out.println("Grandparent's Print()");
    }
}

class Parent extends Grandparent {
    public void Print()
    {
        System.out.println("Parent's Print()");
    }
}

class Child extends Parent {
    public void Print()
    {
        // Trying to access Grandparent's Print()
        super.super.Print();
        System.out.println("Child's Print()");
```

```
    }
}

public class Main {
    public static void main(String[] args)
    {
        Child c = new Child();
        c.Print();
    }
}
```

output will be

```
prog.java:20: error: <identifier> expected
    super.super.Print();
          ^
prog.java:20: error: not a statement
        super.super.Print();
```

- In Java, protected members of a class “A” are accessible in other class “B” of `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. In C++, protected members of a class can be accessed in derived class.

```
// filename B.java
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);
    }
}
```

- Java uses extends keyword for inheritence. Unlike C++, Java doesn’t provide an inheritance specifier like public, protected or private. We `cannot` change the protection level of members of base class in Java
- In Java, methods are `virtual` by default. In C++, we explicitly use virtual keyword.
- Java uses a separte keyword interface for interfaces, and abstract keyword for abstract classes and abstract functions.

```
// 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");
}
}
```

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");
}
}
```

- Unlike C++, Java doesn't support multiple inheritance;
- In C++, the default constructor of the parent class is automatically called, but if we want to call parametrized constructor of a parent class, we must use `Initializer list`. Like C++, default constructor of the parent class is automatically called in Java, but if we want to call parametrized constructor then we must use `super` to call the parent constructor. See 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
```