Constructors and Inheritance

constructor of sub class is invoked when we create the object of subclass, In Java, constructor of base class with no argument gets automatically called in derived class constructor. it by default invokes the default constructor of super class. Hence, in inheritance the objects are constructed top-down. The superclass constructor can be called explicitly using the super keyword, but it should be first statement in a constructor. The super keyword refers to the superclass, immediately above of the calling class in the hierarchy. The use of multiple super keywords to access an ancestor class other than the direct parent is not permitted.

```
// filename: Main.java
class Base {
Base() {
        System.out.println("Base Class Constructor Called ");
}
}
class Derived extends Base {
Derived() {
        System.out.println("Derived Class Constructor Called ");
}
}
public class Main {
public static void main(String[] args) {
        Derived d = new Derived();
}
}
```

But, if we want to call parameterized contructor of base class, then we can call it using super(). The point to note is **base class constructor call must be the first line in derived class constructor**. For example, in the following program, super(_x) is first line derived class constructor.

```
// filename: Main.java
class Base {
int x;
Base(int _x) {
            x = _x;
}
}
class Derived extends Base {
int y;
Derived(int _x, int _y) {
            super(_x);
            y = _y;
}
void Display() {
            System.out.println("x = "+x+", y = "+y);
}
}
public class Main {
public static void main(String[] args) {
            Derived d = new Derived(10, 20);
            d.Display();
} }
```

Inheritance and Method Overriding

When we declare the same method in child class which is already present in the parent class the this is called **method overriding**. In this case when we call the method from child class object, the child class version of the method is called. Method overriding is one of the way by which java achieve Run Time Polymorphism. If an object of a parent class is used to invoke the method, then the version in the parent class will be executed, but if an object of the subclass is used to invoke the method, then the version in the child class will be executed. In other words, *it is the type of the object being referred to* (not the type of the reference variable) that determines which version of an overridden method will be executed. However we can call the parent class method using super keyword as I have shown in the example below:

```
class ParentClass{
  //Parent class constructor
  ParentClass(){
                System.out.println("Constructor of Parent");
  }
  void disp(){
                System.out.println("Parent Method");
  }
}
class JavaExample extends ParentClass{
  JavaExample(){
                System.out.println("Constructor of Child");
  }
  void disp(){
                System.out.println("Child Method");
     //Calling the disp() method of parent class
                super.disp();
  }
  public static void main(String args[]){
                //Creating the object of child class
                JavaExample obj = new JavaExample();
                obj.disp(); }
}
```

Rules for method overriding:

Overriding and Access-Modifiers : The access modifier for an overriding method can allow more, but not less, access than the overridden method. For example, a protected instance method in the super-class can be made public, but not private, in the subclass. Doing so, will generate compile-time error.

```
// A Simple Java program to demonstrate
// Overriding and Access-Modifiers
class Parent {
            // private methods are not overridden
            private void m1()
            {
                        System.out.println("From parent m1()");
            }

            protected void m2()
            {
                        System.out.println("From parent m2()");
            } }
class Child extends Parent {
            // new m1() method
            // unique to Child class
            private void m1()
            {
                        System.out.println("From child m1()");
            }

            // overriding method
            // with more accessibility
            @Override
            public void m2()
            {
                        System.out.println("From child m2()");
            } }
// Driver class
class Main {
            public static void main(String[] args)
            {
                        Parent obj1 = new Parent();
                        obj1.m2();
                        Parent obj2 = new Child();
                        obj2.m2(); }}
```

Final methods can not be overridden : If we don’t want a method to be overridden, we declare it as final.

```
// A Java program to demonstrate that
// final methods cannot be overridden

class Parent {
        // Can't be overridden
        final void show() {}
}

class Child extends Parent {
        // This would produce error
        void show() {}
}
```

Static methods can not be overridden(Method Overriding vs Method Hiding) : When you define a static method with same signature as a static method in base class, it is known as method hiding.The following table summarizes what happens when you define a method with the same signature as a method in a super-class. **Private methods can not be overridden. The overriding method must have same return type (or subtype). Invoking overridden method from sub-class .**

	SUPERCLASS INSTANCE METHOD	**SUPERCLASS STATIC METHOD**
SUBCLASS INSTANCE METHOD	Overrides	Generates a compile-time error
SUBCLASS STATIC METHOD	Generates a compile-time error	Hides