Method Overriding :

Also known as Runtime Polymorphism/Dynamic Polymorphism/Late Binding.

In overriding method resolution always takes care by jvm based on runtime object and hence overriding is also considered as Polymorphism/Dynamic Polymorphism/Late Binding.

Class Parent{

Public void m1(){

Syso(“parent m1”);

}

Public void m2(){

Syso(“parent m2”)

}

}

Class Child extends Parent{

Public void m2(){

Syso(“Child m2”);

}

}

Public class Test{

P s v main(String args[])

{

Parent p = new Parent();

p.m2(); //parent m2

Child c = new Child();

c.m2();//Child m2

Parent p = new Child();

p.m2();//Child m2

//Because first compiler will check with reference (here reference is parent reference), whether reference type has m2() method or not. If yes no issues compiles files. But at run time JVM will decide based on run time object(in this case child object as new Child()).i.e. if Child class has any method called m2()(overriding) or not. If yes will executes child’s method

}

}

**Rules for overriding :**

1. In overriding method names and arguments types must be matched. I .e. method signatures must be same.
2. In overriding **return types** must be same, but this rule is applicable until 1.4 java version only.

From 1.5 version onwards we can take **covariant return types**. A/C to this child class method return type need not be same as parent class method return type, its child type also allowed.

Ex : Class p{

Public Object m1(){

return null;

}

}

Class C extends P{

Public String m1(){

return null;

}

}

Note : Covariant return type concept applicable only for Object types but not for primitive types.(int,double,float,char…).

1. Parent class **private methods** not available to the child and hence overriding concept not applicable for private methods.

Based on our requirement we can define exactly same private method in child class, it is valid but not overriding

Class p{

private void m1(){

}

Class c extends P{

private void m1(); // this is not overriding

}

1. We cannot override parent class **final methods** in child classes. If we are trying to override will get compile time error.
2. Parent class abstract method we should override in child class to provide implementation.
3. We can override non abstract method as abstract.

Ex : class p{

Public void m1(){

}

}

Abstract class c extends p{

Public abstract void m1();

}

Main advantage of this approach is we can stop the availability of parent method implementation to the next level child classes.

1. In overrirding following modifiers won’t keep any restricition.

synchronized,native,strictfp

1. If child class throws any checked exception, compulsory parentclass method should throw the same checked exception or its parent. Otherwise we will get compile time error. But there are no restrictions for Unchecked exceptions
2. While overriding we can’t reduce scope of **access modifier**, but we can increase the scope.

Private < default < protected <public

1. We Can’t override a **static** method as non-static. Otherwise we will get compile time error. Similarly we can’t override a nonstatic method as static.

If both parent and child class methods are static then we wont get any compile time error. It seems overriding concept applicable for static methods, but it is not overriding and it is method hiding.

**Static overriding :**

**Static methods can’t be override, if we do override i.e. not actually overriding, i.e. methodhiding**

**public** **class** Parent {

**public** **static** **void** m1() {

System.***out***.println("parent static m1");

}

**public** **void** m2() {

System.***out***.println("parent non-static m2");

}

}

**public** **class** OverRidingDemo **extends** Parent{

**public** **static** **void** m1() {

System.***out***.println("child static m1");

}

**public** **void** m2() {

System.***out***.println("child non-static m2");

}

**public** **void** m3() {

System.***out***.println("Child m3");

}

**public** **static** **void** main(String[] args) {

Parent p = **new** Parent();

p.*m1*();//Parent

p.m2();//Parent

OverRidingDemo c = **new** OverRidingDemo();

c.*m1*();//Overriding method will get execute i..e.Child

c.m2();//Parent

c.m3();//Child

Parent p1 = **new** OverRidingDemo();

p1.*m1*();//Parent

p1.m2();//Child

}

}