**Method overriding in Java** means redefining a method in a subclass to replace the functionality of superclass method.

**When to need Method overriding**

Program source code 1:

public class Marry

{

void engagementDate()

{

System.out.println("Engagement will be done on 23 Dec.");

}

// Overridden method.

void marryDate()

{

System.out.println("Marry will be on 25 Dec");

}

}

--

public class Change extends Marry

{

// Overriding method.

void marrydate()

{

System.out.println("Marry will be on 27 Dec");

}

}

public class MyClass

{

public static void main(String[] args)

{

Change obj = new Change();

obj.engagementDate();

obj.marrydate();

}

}

## Why do we need to create Subclass in Java?

There are mainly three reasons for which we need to create subclass of superclass in Java

1. To add a new feature or properties. For example, a student has properties like age and location.

But in the future, if we get a new requirement to add one more property “address” for that student, we should make a subclass of that class and add a new property address in the subclass.

2. To override or change the existing functionality of superclass method.

3. To inherit the existing functionality of superclass method.

## Features of Method overriding

There are the following features of method overriding in Java. They are as follows:

* Method overriding technique supports the runtime [polymorphism](https://www.scientecheasy.com/2020/07/polymorphism-in-java.html/).
* It allows a subclass to provide its own implementation of the method which is already provided by the superclass.
* Only the instance method can be overridden in Java.
* An instance variable can never be overridden in Java.
* The overriding method can not be more restrictive [access modifiers](https://www.scientecheasy.com/2020/06/access-modifiers-in-java.html/) than overridden method of the superclass.
* Overriding concept is not applicable for private, final, static, and main method in Java.
* From Java 5 onwards, method overriding can also be done by changing the covariant return type only.
* Overriding method cannot throw any checked exception.

## Method Overriding rules in Java

When you are overriding superclass method in a subclass, you need to follow certain rules

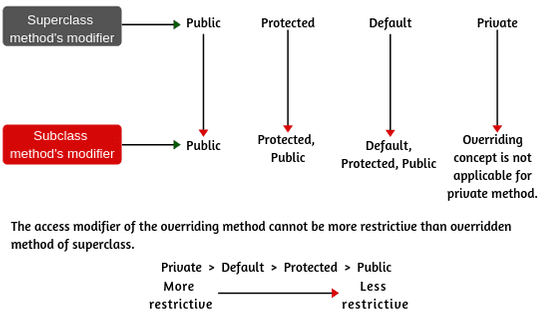
1. Subclass method name must be the same as superclass method name.

2. The parameters of subclass method must be the same as superclass method parameters. i.e. In overriding, method name and argument types must be matched. In other words, the method signature must be the same or matched.

3. Must be Is-A relationship (Inheritance).

4. Subclass method’s return type must be the same as superclass method return type. But this rule is applicable until Java 1.4 version only. From Java 1.5 version onwards, covariant return types are also allowed.

5. Subclass method’s access modifier must be the same or less than the superclass method access modifier. Look at the below figure to understand better.



**Key Points:**

1. While the overriding method, we can increase the visibility of the overriding method but cannot reduce it. For example, if superclass method is protected, we can override as a public method in the subclass.

2. Similarly, the default method of superclass can be overridden by default, protected, or public.

3. We cannot override a method if we do not inherit it. A private method cannot be overridden because it cannot be inherited in the subclass.

Program source code 2:

package overridingProgram;

public class A

{

void m1()

{

System.out.println("A-m1");

}

// Overridden method.

void m2()

{

System.out.println("A-m2");

}

}

public class B extends A

{

// Overriding method.

void m2()

{

System.out.println("B-m2");

}

// Newly defined method in class B.

void m3()

{

System.out.println("B-m3");

}

}

public class MyTest

{

public static void main(String[] args)

{

A a = new A();

a.m1();

a.m2();

B b = new B();

b.m1();

b.m2();

b.m3();

A a1 = new B();

a1.m1();

a1.m2();

}

}