**Method overriding in java with example**

Declaring a method in **subclass** which is already present in **parent class** is known as method overriding. Earlier we shared [**method overloading in java**](http://beginnersbook.com/2013/05/method-overloading/). In this tutorial we will see **method overriding with examples**.

**Example:**

One of the simplest example – Here Boy class extends Human class. Both the classes have a common method void eat(). Boy class is giving its own implementation to the eat() method or in other words it is overriding the method eat().

class Human{

public void eat()

{

System.out.println("Human is eating");

}

}

class Boy extends Human{

public void eat(){

System.out.println("Boy is eating");

}

public static void main( String args[]) {

Boy obj = new Boy();

obj.eat();

}

}

Output:

Boy is eating

**Advantage of method overriding**

The main advantage of method overriding is that the class can give its own specific implementation to a inherited method without even modifying the parent class(base class).

**Method Overriding in dynamic method dispatch**

Dynamic method dispatch is a technique which enables us to assign the base class reference to a child class object. As you can see in the below example that the base class reference is assigned to child class object.

class ABC{

public void disp()

{

System.out.println("disp() method of parent class");

}

public void abc()

{

System.out.println("abc() method of parent class");

}

}

class Test extends ABC{

public void disp(){

System.out.println("disp() method of Child class");

}

public void xyz(){

System.out.println("xyz() method of Child class");

}

public static void main( String args[]) {

//Parent class reference to child class object

ABC obj = new Test();

obj.disp();

obj.abc();

}

}

Output:

disp() method of Child class

abc() method of parent class

**Note**: In dynamic method dispatch the object can call the overriding methods of child class and all the non-overridden methods of base class but it cannot call the methods which are newly declared in the child class. In the above example the object obj was able to call the disp()(overriding method) and abc()(non-overridden method of base class). However if you try to call the xyz() method (which has been newly declared in Test class) [obj.xyz()] then it would give compilation error with the following message:

Exception in thread "main" java.lang.Error: Unresolved compilation

problem: The method xyz() is undefined for the type ABC

**Rules of method overriding in Java**

1. Argument list: The argument list of overriding method must be same as that of the method in parent class. The data types of the arguments and their sequence should be maintained as it is in the overriding method.
2. [**Access Modifier**](http://beginnersbook.com/2013/05/java-access-modifiers/): The Access Modifier of the overriding method (method of subclass) cannot be more restrictive than the overridden method of parent class.  For e.g. if the Access Modifier of base class method is public then the overriding method (child class method ) cannot have private, protected and default Access modifier as all of the three are more restrictive than public.  
   For e.g. This is not allowed as child class disp method is more restrictive(protected) than base class(public)
3. class MyBaseClass{
4. public void disp()
5. {
6. System.out.println("Parent class method");
7. }
8. }
9. class MyChildClass extends MyBaseClass{
10. protected void disp(){
11. System.out.println("Child class method");
12. }
13. public static void main( String args[]) {
14. MyChildClass obj = new MyChildClass();
15. obj.disp();
16. }

}

Output:

Exception in thread "main" java.lang.Error: Unresolved compilation

problem: Cannot reduce the visibility of the inherited method from MyBaseClass

However this is perfectly valid scenario as public is less restrictive than protected. Same access modifier is also a valid one.

class MyBaseClass{

protected void disp()

{

System.out.println("Parent class method");

}

}

class MyChildClass extends MyBaseClass{

public void disp(){

System.out.println("Child class method");

}

public static void main( String args[]) {

MyChildClass obj = new MyChildClass();

obj.disp();

}

}

Output:

Child class method

1. private, static and final methods cannot be overridden as they are local to the class. However static methods can be re-declared in the sub class, in this case the sub-class method would act differently and will have nothing to do with the same static method of parent class.
2. Overriding method (method of child class) can throw any [**unchecked exceptions**](http://beginnersbook.com/2013/04/java-checked-unchecked-exceptions-with-examples/), regardless of whether the overridden method(method of parent class) throws any exception or not. However the overriding method should not throw [**checked exceptions**](http://beginnersbook.com/2013/04/java-checked-unchecked-exceptions-with-examples/) that are new or broader than the ones declared by the overridden method. We will discuss this in detail with example in the upcoming tutorial.
3. Binding of overridden methods happen at runtime which is known as[**dynamic binding**](http://beginnersbook.com/2013/04/java-static-dynamic-binding/).
4. If a class is extending an [**abstract class**](http://beginnersbook.com/2013/05/java-abstract-class-method/) or implementing an [**interface**](http://beginnersbook.com/2013/05/java-interface/)then it has to override all the abstract methods unless the class itself is a abstract class.

**Super keyword in Overriding**

**super** keyword is used for calling the parent class method/constructor.super.methodname() calling the specified method of base class whilesuper() calls the [**constructor**](http://beginnersbook.com/2013/03/constructors-in-java/)of base class. Let’s see the use of super in Overriding.

class ABC{

public void mymethod()

{

System.out.println("Class ABC: mymethod()");

}

}

class Test extends ABC{

public void mymethod(){

//This will call the mymethod() of parent class

super.mymethod();

System.out.println("Class Test: mymethod()");

}

public static void main( String args[]) {

Test obj = new Test();

obj.mymethod();

}

}

Output:

Class ABC: mymethod()

Class Test: mymethod()