# **Method Overriding**

Declaring a method in **sub class** which is already present in **parent class** is known as method overriding. Overriding is done so that a child class can give its own implementation to a method which is already provided by the parent class. In this case the method in parent class is called overridden method and the method in child class is called overriding method.

Example:

Father.java

**package** FPPackage;

**public** **class** Father {

**public** **void** action() {

System.***out***.println("Walking");

}

}

Child.java

**package** FPPackage;

**public** **class** Child **extends** Father{

**public** **void** action() {

System.***out***.println("Crawling");

**super**.action();

}

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

Child c = **new** Child();

c.action();

}

}

Advantage of Method Overriding:

The main advantage of method overriding is that the class can give its own specific implementation to an inherited method **without even modifying the parent class code**.

**Method Overriding and Dynamic Method Dispatch:**

Method Overriding is an example of [runtime polymorphism](https://beginnersbook.com/2013/04/runtime-compile-time-polymorphism/). When a parent class reference points to the child class object then the call to the overridden method is determined at runtime, because during method call which method (parent class or child class) is to be executed is determined by the type of object. This process in which call to the overridden method is resolved at runtime is known as dynamic method dispatch.

**Example:**

Man.java

**package** FPPackage;

**public** **class** Man {

**public** **void** action() {

System.***out***.println("Crawl, Walk and Run");

}

}

Youngster.java

**package** FPPackage;

**public** **class** Youngster **extends** Man{

**public** **void** action() {

System.***out***.println("Walk and Run");

}

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

Man m = **new** Youngster();

m.action();

}

}

Child.java

**package** FPPackage;

**public** **class** Child **extends** Youngster{

**public** **void** action() {

System.***out***.println("Crawl");

}

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

Man m = **new** Man();

m.action();

Youngster y = **new** Youngster();

y.action();

Child c = **new** Child();

c.action();

Man m1 = **new** Child();

m1.action();

}

}

Rules of method overriding in Java:

Argument list: The argument list of overriding method (method of child class) must match the Overridden method (the method of parent class). The data types of the arguments and their sequence should exactly match.

[Access Modifier](https://beginnersbook.com/2013/05/java-access-modifiers/) 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 parent class method is public then the overriding method (child class method) cannot have private, protected and default Access modifier, because all of these three access modifiers are more restrictive than public.

**Overloading Vs Overriding:**

1. The basic difference is that overloading happens in the same class but for overriding to happen, we need to have a parent class and a child class.
2. Private and final methods can be overloaded but can not be overridden.

Ex: Change the parent class action method to private and see what happens.

1. Overloading happens at compile time and overriding happens at run time.
2. Performance: Overloading gives better performance compared to overriding. The reason is that the binding of overridden methods is being done at runtime.
3. Return type of method does not matter in case of method overloading, it can be same or different. However, in case of method overriding the overriding method can have more specific return type.
4. Argument list should be different while doing method overloading. Argument list should be same in method Overriding.