**Method Overriding in Java**

Whenever same method name is existing in both base class and derived class with same types of parameters or same order of parameters is known as **method Overriding**. Here we will discuss about **Overriding in Java**.

**Note:**Without Inheritance method overriding is not possible.

**Advantage of Java Method Overriding**

* Method Overriding is used to provide specific implementation of a method that is already provided by its super class.
* Method Overriding is used for Runtime Polymorphism

**Rules for Method Overriding**

* method must have same name as in the parent class.
* method must have same parameter as in the parent class.
* must be IS-A relationship (inheritance).

**Understanding the problem without method overriding**

Lets understand the problem that we may face in the program if we do not use method overriding.

**Example Method Overriding in Java**

**class** Walking

{

**void** walk()

{

System.**out**.println("Man walking fastly");

}

}

**class** OverridingDemo

{

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

{

Man obj = **new** Man();

obj.walk();

}

}

**Output**

Man walking

Problem is that I have to provide a specific implementation of walk() method in subclass that is why we use method overriding.

**Example of method overriding in Java**

In this example, we have defined the walk method in the subclass as defined in the parent class but it has some specific implementation. The name and parameter of the method is same and there is IS-A relationship between the classes, so there is method overriding.

**Example**

**class** Walking

{

**void** walk()

{

System.**out**.println("Man walking fastly");

}

}

**class** Man **extends** walking

{

**void** walk()

{

System.**out**.println("Man walking slowly");

}

}

**class** OverridingDemo

{

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

{

Man obj = **new** Man();

obj.walk();

}

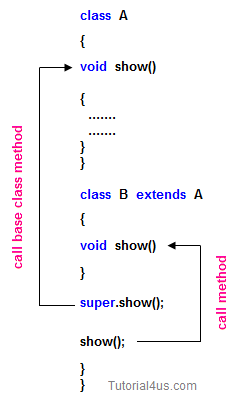
}

**Output**

Man walking slowly

**Note:**Whenever we are calling overridden method using derived class object reference the highest priority is given to current class (derived class). We can see in the above example high priority is derived class.

**Note:** super. (super dot) can be used to call base class overridden method in the derived class.



**Accessing properties of base class with respect to derived class object**

**class** A

{

**int** x;

**void** f1()

{

x=10;

System.**out**.println(x);

}

**void** f4()

{

System.**out**.println("this is f4()");

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

}

};

**class** B **extends** A

{

**int** y;

**void** f1()

{

**int** y=20;

System.**out**.println(y);

System.**out**.println("this is f1()");

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

}

};

**class** C **extends** A

{

**int** z;

**void** f1()

{

z=10;

System.**out**.println(z);

System.**out**.println("this is f1()");

}

};

**class** Overide

{

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

{

A a1=**new** B();

a1.f1();

a1.f4();

A c1=**new** C();

c1.f1();

c1.f4();

}

}

**Example of Implement overriding concept**

**class** Person

{

String name;

**void** sleep(String name)

{

**this**.name=name;

System.**out**.println(**this**.name +"is sleeping+8hr/day");

}

**void** walk()

{

System.**out**.println("this is walk()");

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

}

};

**class** Student **extends** Person

{

**void** writExams()

{

System.**out**.println("only student write the exam");

}

**void** sleep(String name)

{

**super**.name=name;

System.**out**.println(**super**.name +"is sleeping 6hr/day");

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

}

};

**class** Developer **extends** Person

{

**public** **void** designProj()

{

System.**out**.println("Design the project");

}

**void** sleep(String name)

{

**super**.name=name;

System.**out**.println(**super**.name +"is sleeping 4hr/day");

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

}

};

**class** OverideDemo

{

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

{

Student s1=**new** Student();

s1.writExams();

s1.sleep("student");

s1.walk();

Developer d1=**new** Developer();

d1.designProj();

d1.sleep("developer");

}

}

**Difference between Overloading and Overriding**

|  |  |  |
| --- | --- | --- |
|  | **Overloading** | **Overriding** |
| 1 | Whenever same method or Constructor is existing multiple times within a class either with different number of parameter or with different type of parameter or with different order of parameter is known as Overloading. | Whenever same method name is existing multiple time in both base and derived class with same number of parameter or same type of parameter or same order of parameters is known as Overriding. |
| 2 | Arguments of method must be different at least arguments. | Argument of method must be same including order. |
| 3 | Method signature must be different. | Method signature must be same. |
| 4 | Private, static and final methods can be overloaded. | Private, static and final methods can not be override. |
| 5 | Access modifiers point of view no restriction. | Access modifiers point of view not reduced scope of Access modifiers but increased. |
| 6 | Also known as compile time polymorphism or static polymorphism or early binding. | Also known as run time polymorphism or dynamic polymorphism or late binding. |
| 7 | Overloading can be exhibited both are method and constructor level. | Overriding can be exhibited only at method label. |
| 8 | The scope of overloading is within the class. | The scope of Overriding is base class and derived class. |
| 9 | Overloading can be done at both static and non-static methods. | Overriding can be done only at non-static method. |
| 10 | For overloading methods return type may or may not be same. | For overriding method return type should be same. |

**Note:**In overloading we have to check only methods names (must be same) and arguments types (must be different) except these the remaining like return type access modifiers etc. are not required to check   
But in overriding every things check like method names arguments types return types access modifiers etc.