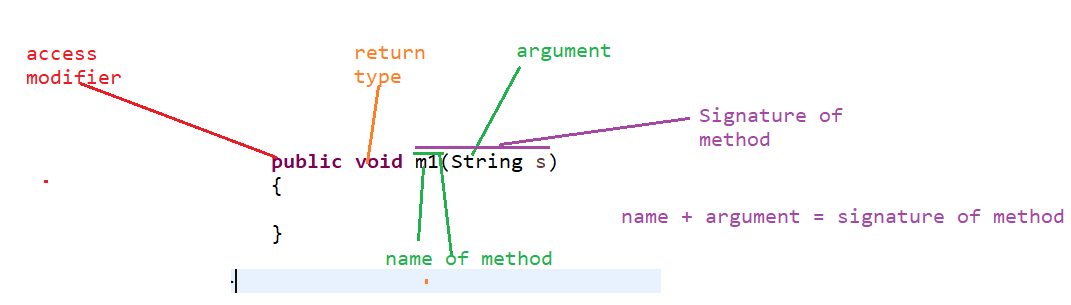
Polymorphism: Method with same name but different implementation represents polymorphism.



There are 2 types of polymorphism:

1. Overloading.

2. Overriding.

1. Overloading: Whenever we have same method name but different signature then those methods are called as overloaded methods

Example:

**public** **class** Test {

**public** **void** m1(String s)

{

System.***out***.println("String argument method");

}

**public** **void** m1()

{

System.***out***.println("no argument method");

}

**public** **void** nameChange(String name, **int** Documentid)

{

System.***out***.println("name has been updated");

}

**public** **void** nameChange(String name, **int** Documentid, String marriagecert)

{

System.***out***.println("name has been updated as per certificate");

}

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

Test t = **new** Test();

t.m1();

t.m1("abc");

t.nameChange("Robin", 50);

t.nameChange("Hariet", 25, "Hariet Johnson");

}

Output:

no argument method

String argument method

name has been updated

name has been updated as per certificate

2. Type of method (static or non-static) doesn’t make any difference for making a method as overloaded. It should follow name of method – same and signature – different.

3. Return type of the method doesn’t matter to call a method as overloaded method.

Example:

**public** **static** **void** m1(**boolean** b)

{

System.***out***.println("Boolean argument method");

}

**public** **int** m1(**double** d)

{

System.***out***.println("m1 method with int return type");

**return** 50;

}

The above two methods are overloaded methods..

4. Access modifier: Access modifier of any method doesn’t matter to make two methods as overloaded.

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

{

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

}

**private** **void** m3(**int** i)

{

System.***out***.println("private method m3");

}

The above methods are overloaded as their names are same but arguments are different.

5. In Overloading method resolution (execution) is based on reference variable. That’s why it is also known as compile time polymorphism.

6. Overloading is also known as Early binding.

2. Overriding: Whenever method inside the parent is by default available to child through inheritance. If Child class is not satisfied with the implementation then child class is allowed to redefine the method then that method is called as Overridden method.

Rules for a method to be called as overridden:

1. In overriding the method name and argument type must be same for both the class.

2. Return type for both the methods (parent and child class) must be same.

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

**public** **int** m1()

{

System.***out***.println("m1 method from Parent class");

**return** 80;

}

**public** **class** Child **extends** Parent{

**public** **void** m1()

{

System.***out***.println("m1 method from child class");

}

The above combination is not valid because for overriding the return types for both the methods must be same.

3. Type of the method must be non-static to make the method overridden i.e we cannot have one method as static and other method as non static. This combination will generate an error.

**Public class** Parent {

**public** **static** **void** m2()

{

System.***out***.println("m2 static method ");

}

}

**public** **class** Child **extends** Parent{

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

{

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

}

}

The above combination is invalid we cannot have an overridden method which is static in one class and non static in another class. To override a method we must use only non static method because static method cannot be overridden.

4. With respect to access modifier the access modifier from parent class should be same into the child class or it should improve in the child class but it cannot reduce in the child.

Example:

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

**void** m3()

{

System.***out***.println("public m3 method");

}

**public** **class** Child **extends** Parent{

**protected** **void** m3()

{

System.***out***.println("public m3 method");

}

Note: in the above example the scope of access modifier gets improve hence it is valid.

We can have an alternate way to create the object:

Parentclass referencevariable = new Childclass();

If A is the Parent class and B is a Child class then we can create an object like:

A a1 = new B();

The method which got overridden in which the method resolution (execute) is based on runtime object. That’s why overriding is also known as Run time polymorphism.

Overriding is also known as Late-binding.

**public** **class** A {

**public** **void** m1()

{

System.***out***.println("m1 method from A class");

}

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

{

System.***out***.println("m2 method from A class");

}

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

{

System.***out***.println("m3 method from A class");

}

**public** **void** m4()

{

System.***out***.println("m4 method from A class");

}

}

**public** **class** B **extends** A{

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

{

System.***out***.println("m2 method from B class");

}

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

{

System.***out***.println("m3 method from B class");

}

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

B b = **new** B();

b.m2();// B class m2 method

b.m1();// A class m1 method

A a = **new** A();

a.m2();// A class m2 method

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

aa.m2();// B class method

aa.m4();// A class method

}

}

Static method vs overriding: Static method doesn’t follow overriding but it follows **method** **hiding** in which if we call the static method which is present in both Parent and child by same name and argument and if we call it by reference variable then it always gets call on the basis of reference variable only.

Example:

**public** **class** A {

**public** **static** **void** m5()

{

System.***out***.println("static method m5 from Class A");

}

**public** **class** B **extends** A{

**public** **static** **void** m5()

{

System.***out***.println("static method m5 from Class B");

}

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

B b = **new** B();

// calling static method

b.*m5*();// B class static method

A a = **new** A();

a.*m5*();//A class Static method

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

aa.*m5*();// A class static method

**Variable with respect to overriding**: Variable doesn’t follow overriding they will get call on the basis of reference variable but not object.

Example:

**public** **class** A {

String s = "abc";

**public** **class** B **extends** A{

String s = "def";

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

B b = **new** B();

System.***out***.println(b.s);// def

A a = **new** A();

System.***out***.println(a.s);//abc

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

System.***out***.println(aa.s);//abc

}

}

Note : main method can be overloaded but cannot be overridden because static method cannot be override they can only follow method hiding.

Difference between Overloading and overriding:

|  |  |  |  |
| --- | --- | --- | --- |
| **Sr no** | **Property** | **Overloading** | **Overriding** |
| 1 | Method name | Must be same | Must be same. |
| 2 | Argument type | Must be different.(atleast order different) | Must be same (including order). |
| 3 | Method signature | Must be different | Must be same. |
| 4 | Return type | No restrictions | Must be same. |
| 5 | Static method | Method type doesn’t matter | Static method doesn’t follow overriding it follows method hiding. |
| 6 | Access modifier | No restrictions | Must be same or increasing in the child class. |
| 7 | Method resolution | Based on reference variable | Based on runtime object. |
| 8 | Known as | Compile time polymorphism, static binding and Early binding | Runtime polymorphism or Late binding or dynamic binding. |

Through polymorphism we achieve portability into the code.