**Method Overloading**

Developing multiple methods with same method name but different method signatures is known a method overloading.

The signature should change either in terms of

1. Number of arguments.
2. Datatype of arguments.
3. Sequence of argument.

Note: Method overloading cannot differ from return type.

Example:

**package** methodOverloading;

**public** **class** Demo2 {

**void** print(){

System.***out***.println("print() no argument constructor");

}

/\*boolean print(){

System.out.println("print() no argument constructor");

return false;

}\*/ //print() method having same method name and same signature this cannot be //method-overloading, we should not consider return type.

String print(**int** i){

System.***out***.println("print() one argument constructor");

**return** "success";

}

String print(**int** i, **int** j){

System.***out***.println("print() two argument constructor");

**return** "success2";

}

String print(**long** l,**long** m){

System.***out***.println("print() two long argument constructor");

**return** "success3";

}

String print(String s,**int** i){

System.***out***.println("print() two STring and int argument constructor");

**return** "success4";

}

String print(**int** i,String s){

System.***out***.println("print() two int and string argument constructor");

**return** "success5";

}

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

String res;

Demo2 d=**new** Demo2();

d.print();

res=d.print(555);

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

res=d.print(123,466);

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

res=d.print(100l,200l);

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

res=d.print(555,"hello");

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

res=d.print("helloandhie",4654);

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

}

}

Output:

print() no argument constructor

print() one argument constructor

success

print() two argument constructor

success2

print() two long argument constructor

success3

print() two int and string argument constructor

success5

print() two STring and int argument constructor

success4

**Important**

1. Method overloading applies to both static and non-static methods.
2. Method can be overloaded in the same class (usually) or sub class.

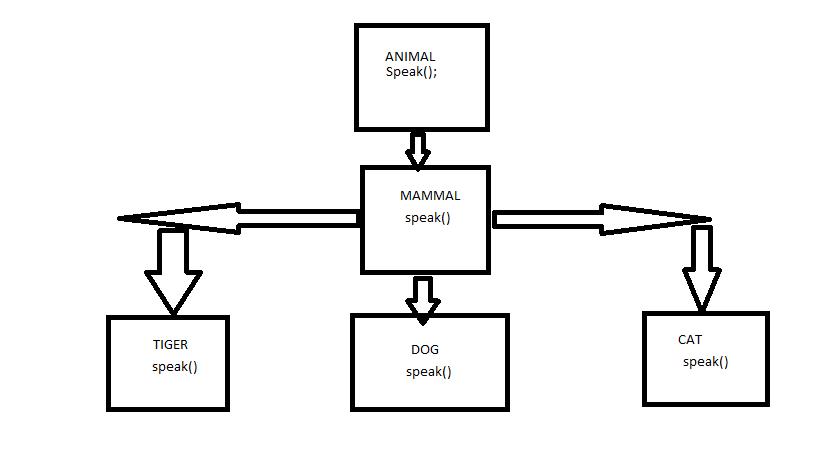
**Method Overriding**

**Definition:**

Subclass providing new implementation of already defined method in the super class i.e., redefining the non-static method which is already defined in the super class.

**Rules for method overriding:**

1. There should be inheritance (“is a relationship”).
2. Method should be non-static
3. Method should have same name as in superclass with same signature and same sequence and return type (for primitive) should also be same. For non-primitive return type can change
4. Static methods are not over ridden in subclass because static member belongs to class but it is hidden.



package methodOverriding;

class Animal{

void speak()

{

System.out.println("Animal Speaking");

}

}

class Mammal extends Animal{

void speak()

{

System.out.println("Mammal Speaking");

}

}

class Dog extends Mammal{

void speak()

{

System.out.println("BOW BOW");

}

}

public class Demo50 {

public static void main(String args[])

{

Mammal m1=new Mammal();

m1.speak();

Animal a1=new Animal();

a1.speak();

Dog d1=new Dog();

d1.speak();

}

}

**Can we override Static methods?**

No, Static members cannot be over ridden. They belong to class , though it get inherited , if we redefine . The original method gets hidden.

**What is hiding?**

When you have a static method in the subclass which has same method name and signature and return type as the method in super class the then sub class method is set to hide the super class inherited method.

This applies for only static method

**Note:**

Static method in super class cannot be re-defined in sub class as non-static method and same vice versa

Program:

package com.qsp.pack1;

class A{

static int i=100;

static void print(){

System.out.println("hello world");

}

}

class B extends A{

static int i=200;

static void print(){

System.out.println("I’m hiding");

}

}

class Demo63{

public static void main(String...args){

B.print();

System.out.println(A.i);

System.out.println(B.i);

A.print();

}

}

**Note:** print() method of super class is hidden by the subclass print() method but still you can able to access print() method of super class and print() method of sub class and also here we are redefining the static variable in sub class as 200 which is already defined in the super class as 100, when we access variable of sub class it prints 200 and when we access the variable of super class it will print 100. That means variable of super class is shadowed by the sub class variable.

Program:

package com.qsp.pack1;

class StaticHiding{

void print(){

System.out.println("Hiya");

}

static void test(){

System.out.println("Static method");

}

}

class B extends StaticHiding{

void print(){// method over riding

System.out.println("Huya");

}

static void test()// method hiding{

System.out.println("HIDING");

}

}

class C{

public static void main(String args[])

{

B b1=new B();

b1.test();//we are accessing test() method and print() method.

b1.print();

}

}

O/P:

C:\practice\_java\bin>java com.qsp.pack1.C

HIDING

Huya

**Important:**

We cannot change from static method to non-static method in sub class & vice versa.

We usually overload method in same class but method can be overloaded in subclass, if method is overload in sub class then it should be non-static method only.

Overriding applies to only non-static method.

Over loading can be applied to both static and non-static method.

**Note:**

This and Super are keywords

This- refers to current object and can be used within a non-static method or a constructor, we can refer to any members of the current object.

Super-refers to super class non static members, we can call super class non static members from sub class non static method using “super.member\_name”

We can also invoke super class constructor from sub class constructor using super().

super() and this() can be used only in constructors

“this.” and “super.” can be used only in non-static methods.

super() and this() cannot be used together.

**Note:** we cannot override constructor as they are not members of class so they are not inherited.

Example for co variant return type

package com.qsp.pack1;

class Apple()

{  
Apple get()

{  
return this;

}

}

class OotyApple extends Apple

{

OotyApple get()//method over ridden because for non primitive variable returntype might change

{

return this;

}

void print()

{

System.out.println(“co variant return type of demo”);

}

public static void main(String args)

{

new OotyApple().get().print();

}

}

Note:

Co variant applies for non-primitive and non-final return types.

We can change to bigger access specifier in the sub-class; we cannot change from bigger to smaller access specifier in the sub class

Program:

class A

{

void print()

{

System.out.println("print method called ");

}

private void disp()

{

System.out.println("disp method called");

}

public void paint()

{

System.out.println("print method called");

}

}

class B extends A// inheriting

{

void print()//this is possible because access specifier in both classes is same since scope is same

{

System.out.println("print method over ridden called ");

}

private void disp()//this we are not overridding since its not inherited this method belongs to this class

{

System.out.println("disp metohod of current class");

}

void paint()//this is not possible the scope in above method is bigger than this.

{

System.out.println("paint method called");

}

}

O/P:

C:\practice\_java\src>javac -d ..\bin Accesss.java

Accesss.java:26: error: paint() in B cannot override paint() in A

void paint()

^

attempting to assign weaker access privileges; was public

1 error