Methods

- A group of statement into a single logical unit .
- Methods are used to do task.
- There are 4 ways to use methods
    i. Methods with Argument & with return type.
    ii. Methods with Argument & without return type.
    iii. Methods without Argument & with return type.
    iv. Methods without Argument & without return type.

i. Methods with Argument & with return type

Syntax

```
ReturnType  methodName (arg1,arg2,…..argn)
{                                         Primitive or Reference type
  code       // Task
}
```

Ex.

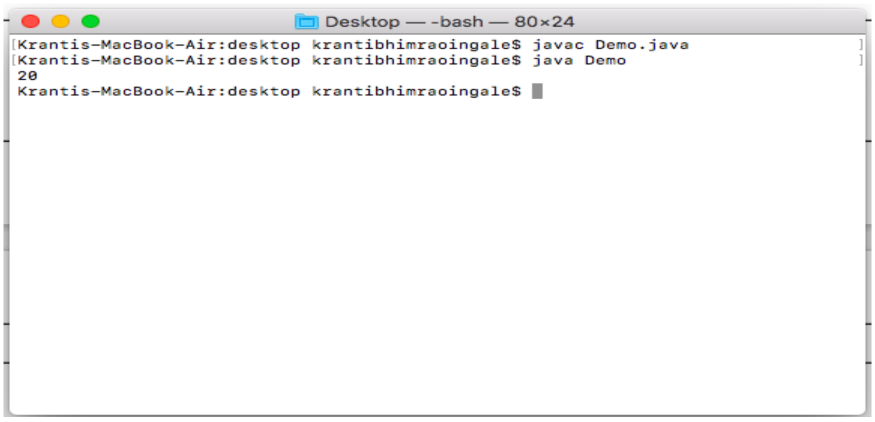
//To find max of two number	**//Sort an array []**

int max(int a , int b)
{
 	
//code
 	
}
	int [] sort (int a[ ])
	{
	// code
	}
	Here it returns only hash code not whole array
	As method returns only one value. By using hash code we can refer whole array

Example

```
class Demo
{
int max(int a, int b)
 {
if (a>b)
return a;
else
return b;
}
public static void main(String[] args)
 {
int max;
Demo d =new Demo(); // call the  default constructor of Demo
max=d.max(10,20);  //based on condition max will have maximum value among a and b
System.out.println(max);
}
}
```

Output:



```
Krantis-MacBook-Air:desktop krantibhimraoingale$ javac Demo.java
Krantis-MacBook-Air:desktop krantibhimraoingale$ java Demo
20
Krantis-MacBook-Air:desktop krantibhimraoingale$
```

ii.Methods with Argument & without return type

Syntax

```
void  methodName (arg1,arg2,…..argn)
{
  code       // Task
}
```

Primitive or Reference type

Ex.

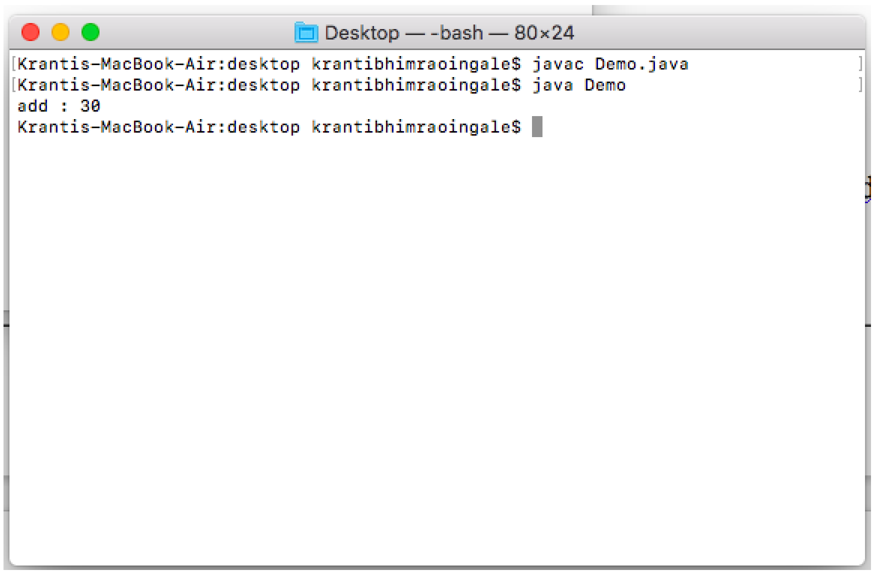
//addition of two number	**//Concatenate two string**
void add(int a , int b)	void concate (String s1.String s2)
{	{
//code	// code
}	}

Example

```
class Demo
{
void  add (int a, int b)
{
int c;
c=a+b;
System.out.println("add : "+c);
}

public static void main(String[] args)
{
int max;
Demo d =new Demo(); // calls default constructor
d.add(10,20);
}
}
```

Output:



```
Krantis-MacBook-Air:desktop krantibhimraoingale$ javac Demo.java
Krantis-MacBook-Air:desktop krantibhimraoingale$ java Demo
add : 30
Krantis-MacBook-Air:desktop krantibhimraoingale$
```

iii.Methods without Argument & with return type

Syntax

```
ReturnType methodName ()
{
  code       // Task
}
```

Ex.

//to get an integer as return	**//get an array**

int get( )
{
//code
return <integer varaible or value>
}
	int[] show ( )
	{
	// code
	return <array variable>
	}

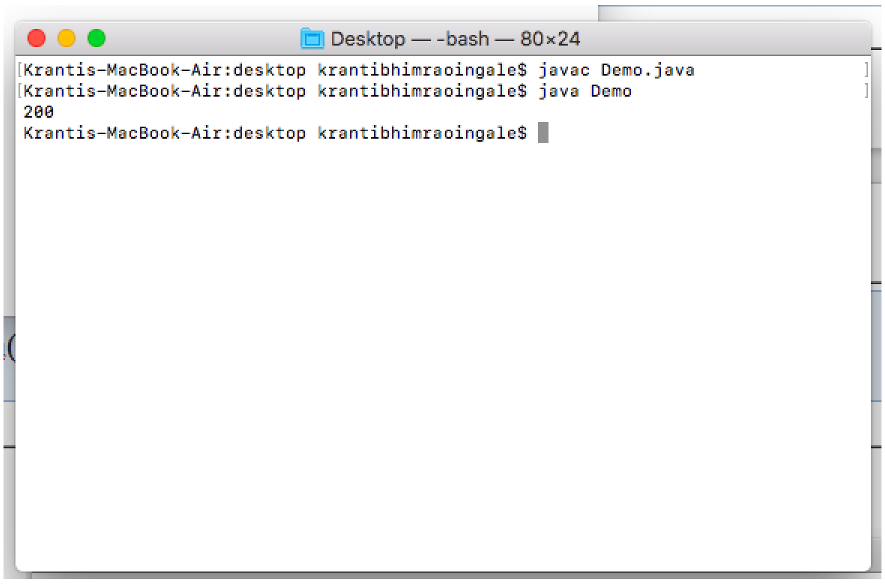
Example

```
class Demo
{
int   get ()
{
int c=200;
return c;    // c must be of type 'int' here , as get() have 'int' as return type
}

public static void main(String[] args)
{
int m;
Demo d =new Demo(); // calls default constructor
m=d.get();   // c's value get copied into local variable m in main method()

System.out.println(m);
}
}
```

Output:



```
Krantis-MacBook-Air:desktop krantibhimraoingale$ javac Demo.java
Krantis-MacBook-Air:desktop krantibhimraoingale$ java Demo
200
Krantis-MacBook-Air:desktop krantibhimraoingale$
```

4.Methods without Argument & without return type

Syntax

```
void methodName ()
{
   code       // Task
}
```

Ex.

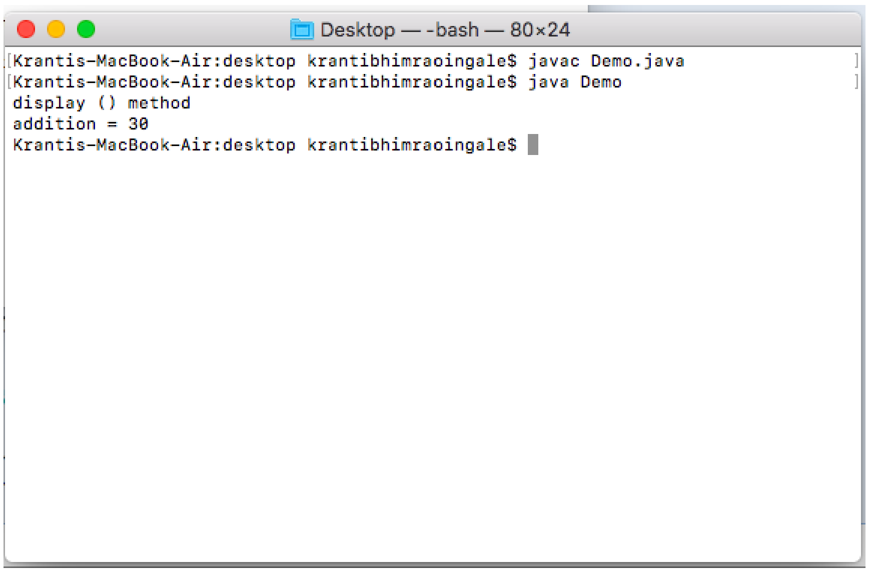
//to display result
void display( )
{
//code
 	
}
	// to show
	void show ( )
	{
	// code
	
	}

Example

```
class Demo
{
void  display ()
{
System.out.println("display () method");
}

 void  add ()
  {
 int a=10,b=20,c;
 c=a+b;
 System.out.println("addition = "+c);
  }
 public static void main(String[] args)
  {
 Demo d =new Demo(); // calls default constructor
 d.display();
 d.add();
}
}
```

Output:



```
Krantis-MacBook-Air:desktop krantibhimraoingale$ javac Demo.java
Krantis-MacBook-Air:desktop krantibhimraoingale$ java Demo
display () method
addition = 30
Krantis-MacBook-Air:desktop krantibhimraoingale$
```

- **Advantage of Methods**
  1. Reusability
  2. Modularity

Polymorphism

- One thing can exhibits more than one form called polymorphism.
- Polymorphism is a Greek word poly means many and polymorphism means many forms.
- The ability to take more than one form

Polymorphism is is of two types:

1. Compile Time Polymorphism (Early binding/Static Binding)
   Ex:-Method Overloading

2. Run Time Polymorphism (Late binding/ Dynamic Programming)
   Ex:-Method Overriding

Overloading Methods

In Java it is possible to define two or more methods within the **same class** that share the same name, as long as their parameter declarations are different. When this is the case, the methods are said to be overloaded, and the process is referred to as **method overloading.**

Method overloading is one of the ways that Java implements **polymorphism**.

When Java encounters a call to an overloaded method, it simply executes the version of the method whose parameters match the arguments used in the call.

Three ways to overload a method
In order to overload a method, the parameter lists of the methods must differ in either of these:

1. Number of parameters.

```
add ( int )                    // number of parameter =1
add ( int , int )             // number of parameter =2
add ( int , int , int )       // number of parameter =3
```

2. Data type of parameters

```
add ( int ,int)               // number of parameter =2 , both are integer
add ( float , int)            // number of parameter =2 , one is float and other is integer
```

3. Sequence of Data type of parameters.

```
add ( int ,float )            // number of parameter =2 , first is int  and second is float
add ( float , int)            // number of parameter =2 , first is float and second is integer
```

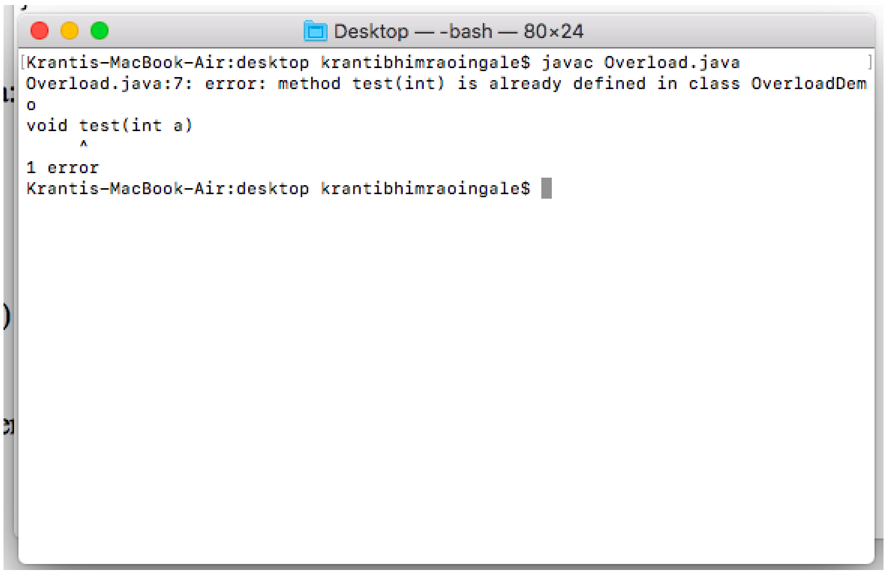
Invalid case of method overloading:

If two methods have same name, same parameters and have different return type, then this is **not a valid** method overloading

```
class OverloadDemo
{
int test(int a)
{
return a;
}
void test(int a)
{
System.out.println("One parameter a:"+a);
}

}
class Overload
{
public static void main(String args[])
{
int m;
OverloadDemo ob=new OverloadDemo();
m=ob.test(10);
ob.test(10);
}
}
```

Output:



```
Krantis-MacBook-Air:desktop krantibhimraoingale$ javac Overload.java
Overload.java:7: error: method test(int) is already defined in class OverloadDem
o
void test(int a)
     ^
1 error
Krantis-MacBook-Air:desktop krantibhimraoingale$
```

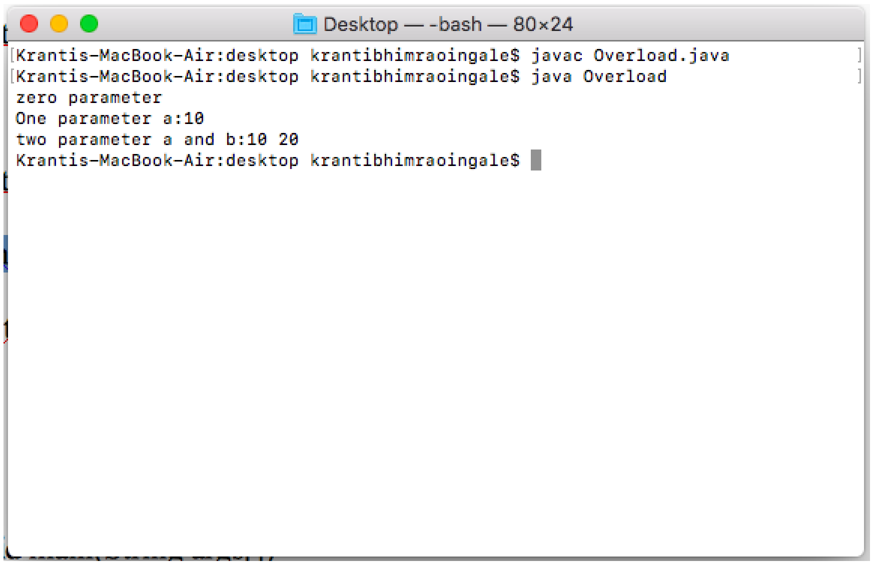
here we want to overload test() method.
Both test() is having same number of parameter and order of parameter is also same.
Only the difference is in return type of test().
Which is not valid hence we got **COMPILE TIME ERROR ,** saying method test already define define in class OverloadDemo

Case 1 : Method overloading with different number of parameters

```
class OverloadDemo
{
void test()
{
System.out.println("zero parameter");
}
void test(int a)    //Overload test for one integer parameter.
{
System.out.println("One parameter a:"+a);
}
void test(int a,int b)  //Overload test for two integer parameters.
{
System.out.println("two parameter a and b:"+a+" "+b);
}
}

class Overload
{
public static void main(String args[])
{
OverloadDemo ob=new OverloadDemo();
double result;
ob.test();
ob.test(10);
ob.test(10,20);
}
}
```

Output



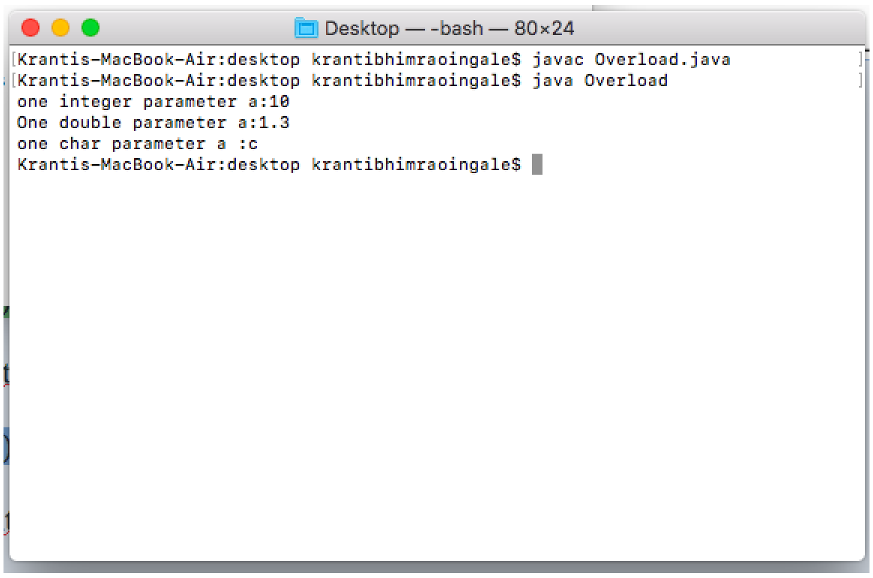
```
Krantis-MacBook-Air:desktop krantibhimraoingale$ javac Overload.java
Krantis-MacBook-Air:desktop krantibhimraoingale$ java Overload
zero parameter
One parameter a:10
two parameter a and b:10 20
Krantis-MacBook-Air:desktop krantibhimraoingale$
```

Case 2 : Method overloading with difference in datatype of parameter

```
class OverloadDemo
{
void test(int a)  //Overload test with one parameter named "int "
{
System.out.println("one integer parameter a:"+a);
}
void test(double a)    //Overload test with one parameter named "double "
{
System.out.println("One double parameter a:"+a);
}
void test(char b)  //Overload test with one parameter named "char "
{
System.out.println("one char parameter a :"+b);
}
}

class Overload
{
public static void main(String args[])
{
OverloadDemo ob=new OverloadDemo();
double result;
ob.test(10);
ob.test(1.3);
ob.test('c');
}
}
```

Output



```
Krantis-MacBook-Air:desktop krantibhimraoingale$ javac Overload.java
Krantis-MacBook-Air:desktop krantibhimraoingale$ java Overload
one integer parameter a:10
One double parameter a:1.3
one char parameter a :c
Krantis-MacBook-Air:desktop krantibhimraoingale$
```

Case 3 : Method overloading with difference in order of parameter

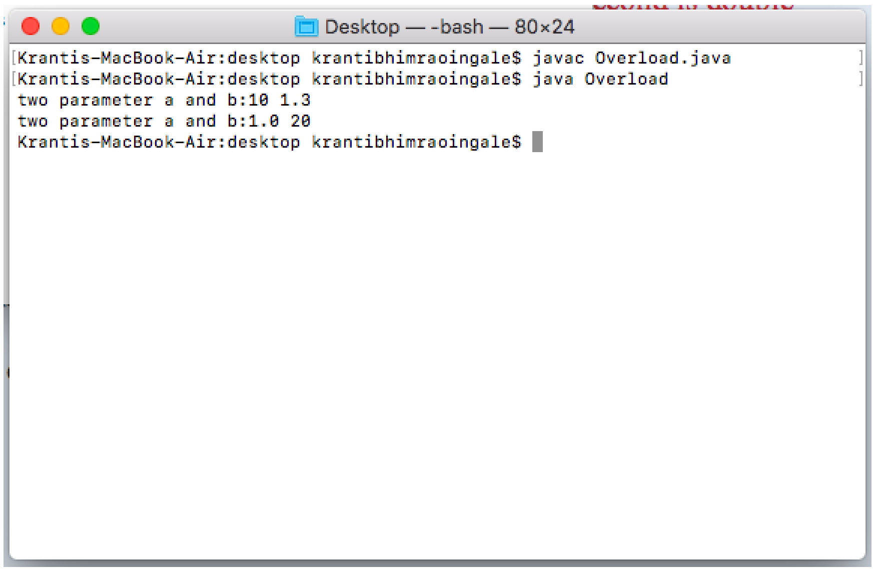
```
class OverloadDemo
{

void test(double b,int a)    //Overload test first arg is double , second is integers
{
System.out.println("two parameter a and b:"+a+" "+b);
}
void test(int a, double b)  //Overload test first arg is integer , second is double
{
System.out.println("two parameter a and b:"+a+" "+b);
}
}

class Overload
{
public static void main(String args[])
{
OverloadDemo ob=new OverloadDemo(); //Calls the default constructor


ob.test(10,1.3);
ob.test(1.0,20);
}
}
```

Output



```
Krantis-MacBook-Air:desktop krantibhimraoingale$ javac Overload.java
Krantis-MacBook-Air:desktop krantibhimraoingale$ java Overload
two parameter a and b:10 1.3
two parameter a and b:1.0 20
Krantis-MacBook-Air:desktop krantibhimraoingale$
```

- **Advantage**

  We don’t have to create and remember different names for functions doing the same thing.
  For example, in our code, if overloading was not supported by Java, we would have to create method names like sum1, sum2, … etc.

Method Overloading and Type Promotion

When a data type of smaller size is promoted to the data type of bigger size than this is called **Type promotion.**

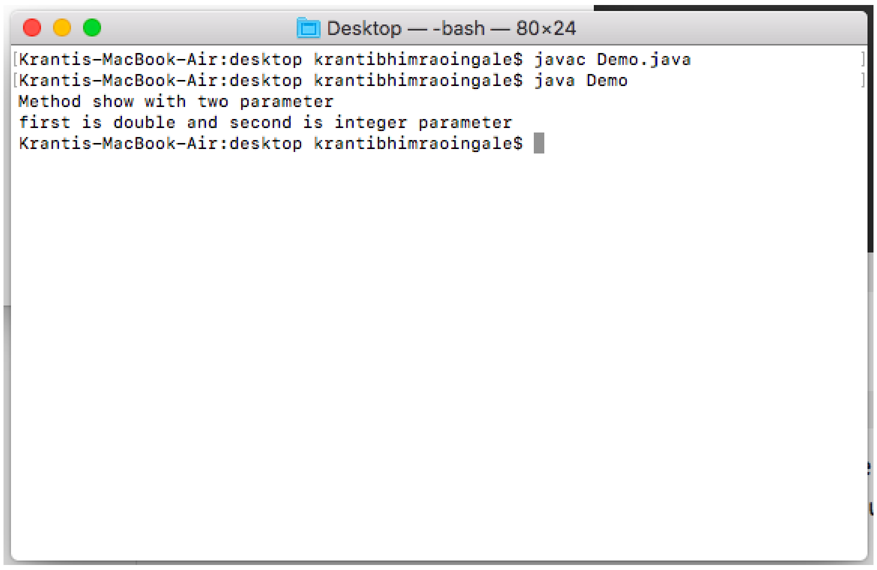
for example:

byte data type can be promoted to short, a short data type can be promoted to int, long, double etc.

What it has to do with method overloading?
Well, it is very important to understand type promotion else you will think that the program will throw compilation error but in fact that program will run fine because of type promotion.

```
class Demo
{
void show(double b, int a)
{
System.out.println("Method show with two parameter");
System.out.println("first is double and second is integer parameter");
}
void show(int a, int b){
System.out.println("Method show with two parameter");
System.out.println("both are integer parameter");
}
public static void main(String args[]){
Demo obj = new Demo();
obj.show(20.67f , 200); //20.67f is float which gets converted into double as we don't have
   }                             // specified method in our program
}
```

Output :



```
Krantis-MacBook-Air:desktop krantibhimraoingale$ javac Demo.java
Krantis-MacBook-Air:desktop krantibhimraoingale$ java Demo
Method show with two parameter
first is double and second is integer parameter
Krantis-MacBook-Air:desktop krantibhimraoingale$
```

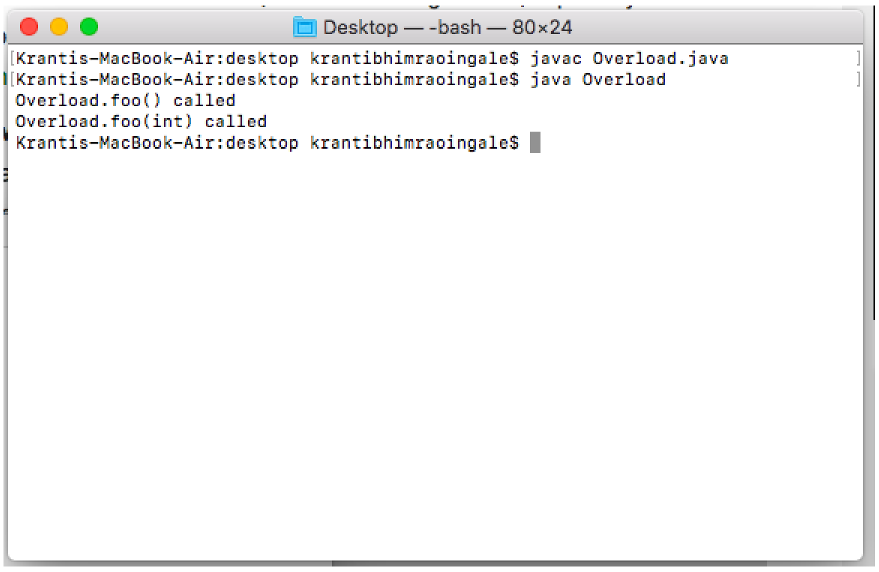
Most IMP Questions

Q.1 What if the exact prototype does not match with arguments.
Ans : Type Conversion takes place.

Q.2 Can we overload static methods?
Ans: YES

```
class Overload
{
static void foo()
{
System.out.println("Overload.foo() called ");
}
static void foo(int a)
{
System.out.println("Overload.foo(int) called ");
}
public static void main(String args[])
{
 Overload.foo();
  Overload.foo(10);
}
}
```

Output:

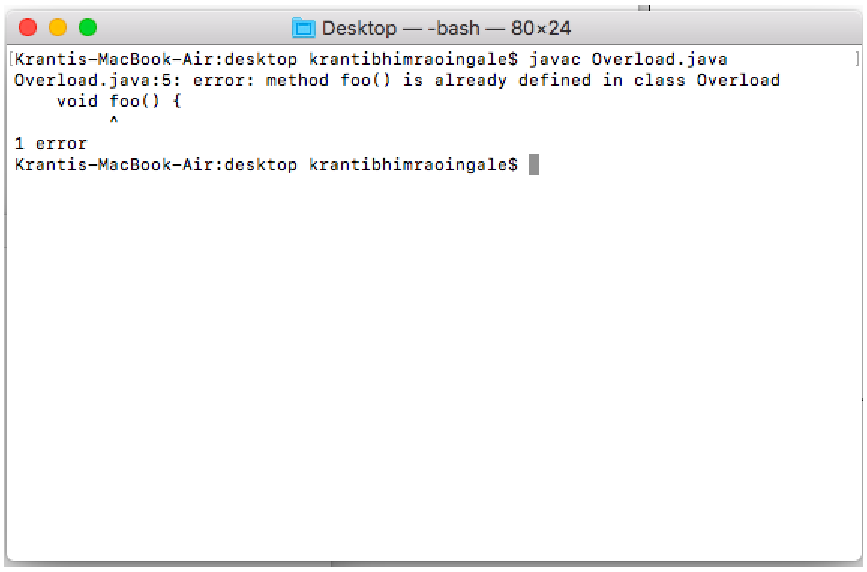


```
Krantis-MacBook-Air:desktop krantibhimraoingale$ javac Overload.java
Krantis-MacBook-Air:desktop krantibhimraoingale$ java Overload
Overload.foo() called
Overload.foo(int) called
Krantis-MacBook-Air:desktop krantibhimraoingale$
```

Q.3 Can we overload methods that differ only by static keyword?
Ans : NO

```
class Overload
{
static void foo()
{
     System.out.println("static foo() called ");
}
static void foo(int a)
{
      System.out.println("non static foo() called ");
}
public static void main(String args[])
{
 Overload ob= new Overload();
  ob.foo();
  Overload.foo();
}
}
```

Output:



```
Krantis-MacBook-Air:desktop krantibhimraoingale$ javac Overload.java
Overload.java:5: error: method foo() is already defined in class Overload
    void foo() {
         ^
1 error
Krantis-MacBook-Air:desktop krantibhimraoingale$
```

we want to overload foo() method.
One foo() is with static and other is without static.
This is not a valid method overloading.
Hence we got **Compile time error** , saying foo() is already define.