|  |
| --- |
| class hello1  {  String name;  hello1() //constructor  {  this.name="good morning";  }  }  public class hello {  public static void main(String args[])  {  hello1 obj=new hello1();  System.out.println(obj.name);    }    } |

Constructor has same name as the class and looks like this in a java code.

public class MyClass{

//This is the constructor

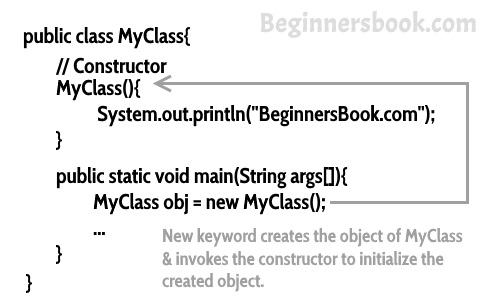
MyClass(){

}

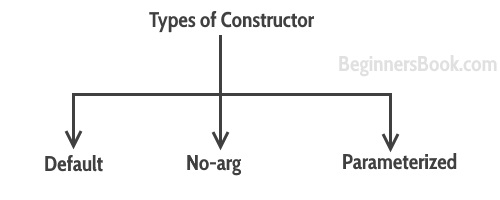
..

}

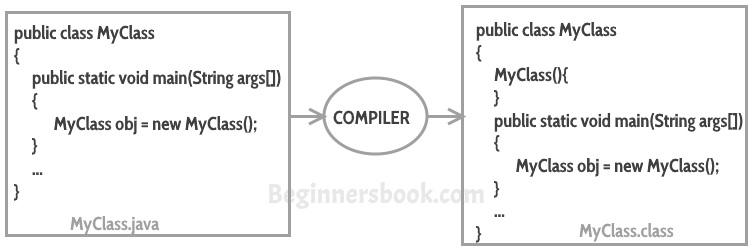
Note that the constructor name matches with the class name and it doesn’t have a return type.



**Types of Constructors**

There are three types of constructors: Default, No-arg constructor and Parameterized.  


**Default constructor**

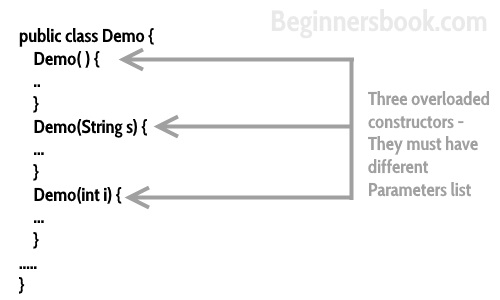
If you do not implement any constructor in your class, Java compiler inserts a [**default constructor**](https://beginnersbook.com/2014/01/default-constructor-java-example/) into your code on your behalf. This constructor is known as default constructor. You would not find it in your source code(the java file) as it would be inserted into the code during compilation and exists in .class file. This process is shown in the diagram below:  


If you implement any constructor then you no longer receive a default constructor from Java compiler.

Parameterized constructor:

|  |
| --- |
| class emp  {  int empid;  String empname;  emp(int id,String name)  {  this.empid=id;  this.empname=name;  }  void display()  {  System.out.println("employee id is : "+empid);  System.out.println("employee name is : "+empname);  }    }  public class employee {  public static void main(String args[])  {  emp obj1=new emp(100,"sharmila");  obj1.display();  }  } |

**Constructor Overloading**

Constructor overloading is a concept of having more than one constructor with different parameters list, in such a way so that each constructor performs a different task.  


Method overloading

Method Overloading is a feature that allows a class to have more than one method having the same name, if their argument lists are different.

Three ways to overload a method:

1. Number of parameters

Eg: add(int,int)

add(int, int, int)

1. Data type of parameters

Eg : add(int,int)

add(int,float)

1. Sequence of data type of parameters:

Eg : add(int , float)

add(flaot,int)

invalid case of method overloading

return type is not considered as method overloading. For example:

int add(int,int)

float add(int,int)

will throw error:

overloading – different number of parameters in argument list

|  |
| --- |
| class overloaddemo  {  void add(int a)  {  a=a+a;  System.out.println("the added value is "+a);  }  void add(int a,int b) //change in data type & different number of parameter  {  int c;  c=a+b;  System.out.println("the added value is "+c);  }  void add(int a,double b) //change in data type  {  double c;  c=a+b;  System.out.println("the added value is "+c);  }  void add(int a,float b) //change in data type  {  float c;  c=a+b;  System.out.println("the added value is "+c);  }  void add(double a,int b) //change in sequence  {  double c;  c=a+b;  System.out.println("the added value is "+c);  }    }  public class overloadeg1 {  public static void main(String args[])  {  overloaddemo obj1=new overloaddemo();  obj1.add(5);  obj1.add(10,20);  obj1.add(50,3.5);  obj1.add(90,9.5f);  obj1.add(100.5,5);  }  } |

Output

|  |
| --- |
| the added value is 10  the added value is 30  the added value is 53.5  the added value is 99.5  the added value is 105.5  Process completed. |

## few Valid/invalid cases of method overloading

Case 1:

int mymethod(int a, int b, float c)

int mymethod(int var1, int var2, float var3)

Result: Compile time error. Argument lists are exactly same. Both methods are having same number, data types and same sequence of data types.

Case 2:

int mymethod(int a, int b)

int mymethod(float var1, float var2)

Result: Perfectly fine. Valid case of overloading. Here data types of arguments are different.

Case 3:

int mymethod(int a, int b)

int mymethod(int num)

Result: Perfectly fine. Valid case of overloading. Here number of arguments are different.

Case 4:

float mymethod(int a, float b)

float mymethod(float var1, int var2)

Result: Perfectly fine. Valid case of overloading. Sequence of the data types of parameters are different, first method is having (int, float) and second is having (float, int).

Case 5:

int mymethod(int a, int b)

float mymethod(int var1, int var2)

Result: Compile time error. Argument lists are exactly same. Even though return type of methods are different, it is not a valid case. Since return type of method doesn’t matter while overloading a method.

Static keyword use

Static Class, Block, Methods and Variables

**Java Static Variables**

A static variable is common to all the instances (or objects) of the class because it is a class level variable. In other words you can say that only a single copy of static variable is created and shared among all the instances of the class. Memory allocation for such variables only happens once when the class is loaded in the memory.  
Few Important Points:

* Static variables are also known as Class Variables.
* Unlike **non-static variables**, such variables can be accessed directly in static and non-static methods.

Static keyword can be used with class, variable, method and block. Static members can be accessed without object. Static members are common for all the instances(objects) of the class but non-static members are separate for each instance of class.

Example:

|  |
| --- |
| class simplestatic  {  static void disp()  {  System.out.println("welcome");  }    public static void main(String args[])  {  disp();  }    } |

Static Block

Static block is used for initializing the static variables.This block gets executed when the class is loaded in the memory. A class can have multiple Static blocks, which will execute in the same sequence in which they have been written into the program.

|  |
| --- |
| class staticblockeg1  {  static int num;  static String mystr;  static  {  num=97;  mystr="static keyword in java";  }  public static void main(String args[])  {  System.out.println("value of static num is :"+num);  System.out.println("value of static mystr is :"+mystr);  }      } |

Multiple static blocks

Lets see how multiple static blocks work in Java. They execute in the given order which means the first static block executes before second static block. That’s the reason, values initialized by first block are overwritten by second block.

|  |
| --- |
| public class multiplestaticblock {  static int num;  static String mystr;  //first static block  static  {  System.out.println("first static block");  num=97;  mystr="block1";  }  //second static block  static  {  System.out.println("second static block");  num=87;  mystr="block2";  }  public static void main(String args[])  {  System.out.println("value of static num is :"+num);  System.out.println("value of static mystr is :"+mystr);  }    } |

Output

|  |
| --- |
| first static block  second static block  value of static num is :87  value of static mystr is :block2  Process completed. |

### Static variables can be accessed directly in Static method

Here we have a static method disp() and two static variables var1 and var2. Both the variables are accessed directly in the static method.

class JavaExample3{

static int var1;

static String var2;

//This is a Static Method

static void disp(){

System.out.println("Var1 is: "+var1);

System.out.println("Var2 is: "+var2);

}

public static void main(String args[])

{

disp();

}

}

**Output:**

Var1 is: 0

Var2 is: null

### Static variables are shared among all the instances of class

In this example, String variable is non-static and integer variable is Static. As you can see in the output that the non-static variable is different for both the objects but the static variable is shared among them, thats the reason the changes made to the static variable by object ob2 reflects in both the objects.

class JavaExample{

//Static integer variable

static int var1=77;

//non-static string variable

String var2;

public static void main(String args[])

{

JavaExample ob1 = new JavaExample();

JavaExample ob2 = new JavaExample();

ob1.var1=88;

ob1.var2="I'm Object1";

ob2.var1=99;

ob2.var2="I'm Object2";

System.out.println("ob1 integer:"+ob1.var1);

System.out.println("ob1 String:"+ob1.var2);

System.out.println("ob2 integer:"+ob2.var1);

System.out.println("ob2 STring:"+ob2.var2);

}

}

Output:

ob1 integer:99

ob1 String:I'm Object1

ob2 integer:99

ob2 STring:I'm Object2

### Static method accessed directly in static and non-static method

class JavaExample{

static int i = 100;

static String s = "Beginnersbook";

//Static method

static void display()

{

System.out.println("i:"+i);

System.out.println("i:"+s);

}

//non-static method

void funcn()

{

//Static method called in non-static method

display();

}

//static method

public static void main(String args[])

{

JavaExample obj = new JavaExample();

//You need to have object to call this non-static method

obj.funcn();

//Static method called in another static method

display();

}

}

Output:

i:100

i:Beginnersbook

i:100

i:Beginnersbook

|  |
| --- |
| public class staticeg2 {  static int a=3;  static int b;  static void method(int x)  {  System.out.println("x= "+x);  System.out.println("a= "+a);  System.out.println("b= "+b);  }  static  {  System.out.println("static block initialized");  b=a\*4;  }  public static void main(String args[])  {  method(43);  }    } |

Output

|  |
| --- |
| static block initialized  x= 43  a= 3  b= 12  Process completed. |

Outside of the class in which they are defined, static methods and variables can be used independently of any object. It can be done by specifying the name of the class followed by the dot operator and the static method.

Eg

Classname.method()

|  |
| --- |
| class static\_eg3  {  static int x=5;  static void disp()  {  System.out.println("welcome");  System.out.println("the value of x is "+x);  }  }  class staticeg3  {    public static void main(String args[])  {  static\_eg3.disp();  }    } |

Output

|  |
| --- |
| welcome  the value of x is 5 |