Methods in Java

A method is a collection of statements that perform some specific task and return result to the caller. A method can perform some specific task without returning anything. Methods allow us to **reuse** the code without retyping the code. In Java, every method must be part of some class which is different from languages like C, C++ and Python.  
Methods are **time savers**and help us to **reuse** the code without retyping the code.

**Method Declaration**

In general, method declarations have six components:

* **Modifier**-: Defines **access type** of the method i.e. from where it can be accessed in your application. In Java, there 3 types of the access specifiers and 4 types of access levels.
  + **public**: accessible in all class in your application.
  + **protected**: accessible within the class in which it is defined and, in its subclass,**(es)**
  + **private**: accessible only within the class in which it is defined.
  + **default/package access** **Level** (declared/defined without using any modifier): accessible within same class and package within which its class is defined.
* **The return type**: The data type of the value returned by the method or void if does not return a value.
* **Method Name**: the rules for field names apply to method names as well, but the convention is a little different.
* **Parameter list:** Comma separated list of the input parameters are defined, preceded with their data type, within the enclosed parenthesis. If there are no parameters, you must use empty parentheses ().
* **Exception list:** The exceptions you expect by the method can throw, you can specify these exception(s).
* **Method body:** it is enclosed between braces. The code you need to be executed to perform your intended operations.

[](http://cdncontribute.geeksforgeeks.org/wp-content/uploads/methods-in-java.png)

**Method signature**: It consists of method name and parameter list (number of parameters, type of the parameters and order of the parameters). Return type and exceptions are not considered as part of it.  
Method Signature of above function:

max (int x, int y)

**How to name a Method?** A method name is typically a single word that should be a **verb** in lowercase or multi-word, that begins with a **verb** in lowercase followed by **adjective, noun…..**After the first word, first letter of each word should be capitalized. For example, findSum,  
computeMax, setX and getX

Generally, A method has a unique name within the class in which it is defined but sometime a method might have same name as other method name within the same class as method overloading is allowed in Java.

**Calling a method**

The method needs to be called for using its functionality. There can be three situations when a method is called:  
A method returns to the code that invoked it when:

* It completes all the statements in the method
* It reaches a return statement
* Throws an exception

|  |
| --- |
| // Program to illustrate methods in java  import java.io.\*;    class Calculator {        int sum = 0;        public int add(int a, int b){            // adding two integer value.          sum = a + b;            //returning summation of two values.          return sum;      }    }    class TestCalculator {      public static void main (String[] args) {            // creating an instance of Calculator class          Calculator c1 = new Calculator ();            // calling add() method to add two integer using instance created          // in above step.          int s = c1.add(1,2);          System.out.println("Sum of two integer values :"+ s);        }  } |

Output:

Sum of two integer values :3

See the below example to understand method call in detail:

|  |
| --- |
| // Java program to illustrate different ways of calling a method  import java.io.\*;    class Test  {      public static int i = 0;      // constructor of class which counts      //the number of the objects of the class.      Test()      {          i++;        }      // static method is used to access static members of the class      // and for getting total no of objects      // of the same class created so far      public static int get ()      {          // statements to be executed....          return i;      }        // Instance method calling object directly      // that is created inside another class 'GFG'.      // Can also be called by object directly created in the same class      // and from another method defined in the same class      // and return integer value as return type is int.      public int m1()      {          System.out.println("Inside the method m1 by object of GFG class");            // calling m2() method within the same class.          this.m2();            // statements to be executed if any          return 1;      }        // It doesn't return anything as      // return type is 'void'.      public void m2()      {            System.out.println("In method m2 came from method m1");      }  }    class GFG  {      public static void main(String[] args)      {          // Creating an instance of the class          Test obj = new Test();            // Calling the m1() method by the object created in above step.          int i = obj.m1();          System.out.println("Control returned after method m1 :" + i);            // Call m2() method          // obj.m2();          int no\_of\_objects = Test.get();            System.out.print("No of instances created till now : ");          System.out.println(no\_of\_objects);        }  } |

Output:

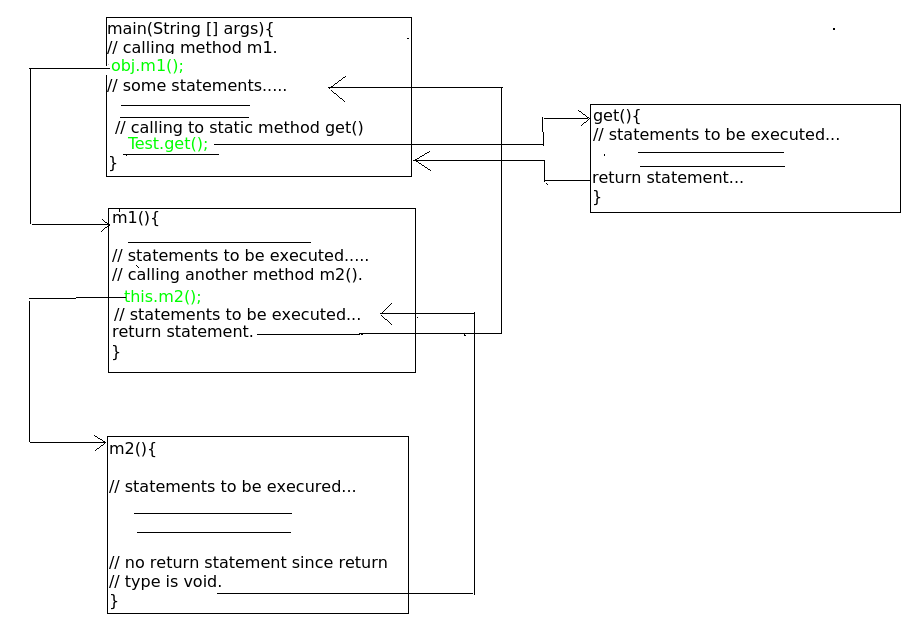
Inside the method m1 by object of GFG class

In method m2 came from method m1

Control returned after method m1 :1

No of instances created till now : 1

Control flow of above program:

[](http://cdncontribute.geeksforgeeks.org/wp-content/uploads/methods-in-java2.png)

**Memory allocation for methods calls**

Methods calls are implemented through stack. Whenever a method is called a stack frame is created within the stack area and after that the arguments passed to and the local variables and value to be returned by this called method are stored in this stack frame and when execution of the called method is finished, the allocated stack frame would be deleted. There is a stack pointer register that tracks the top of the stack which is adjusted accordingly.

**Reference:** <https://docs.oracle.com/javase/tutorial/java/javaOO/methods.html>

# Java Static Method, Variable and Block

## What is Static Variable in Java?

Static variable in Java is variable which belongs to the class and initialized only once at the start of the execution.

* It is a variable which belongs to the class and not to object(instance)
* Static variables are initialized only once, at the start of the execution. These variables will be initialized first, before the initialization of any instance variables
* A single copy to be shared by all instances of the class
* A static variable can be accessed directly by the class name and doesn’t need any object

Syntax:

<**class-name>.<variable-name>**

## What is Static Method in Java?

Static method in Java is a method which belongs to the class and not to the object. A static method can access only static data.

* It is a method which belongs to the class and not to the object(instance)
* A static method can access only static data. It cannot access non-static data (instance variables)
* A static method can call only other static methods and cannot call a non-static method from it.
* A static method can be accessed directly by the class name and doesn’t need any object
* A static method cannot refer to "this" or "super" keywords in anyway

Syntax:

<**class-name>.<method-name>**

**Note:** main method is static, since it must be accessible for an application to run, before any instantiation takes place.

Let’s learn the nuances of the static keywords by doing some exercises!  
  
**Example: How to call static variables & methods**  
  
**Step 1)** Copy the following code into an editor

public class Demo {

public static void main(String args[]){

Student s1 = new Student();

s1.showData();

Student s2 = new Student();

s2.showData();

//Student.b++;

//s1.showData();

}

}

class Student {

int a; //initialized to zero

static int b; //initialized to zero only when class is loaded not for each object created.

Student(){

//Constructor incrementing static variable b

b++;

}

public void showData(){

System.out.println("Value of a = "+a);

System.out.println("Value of b = "+b);

}

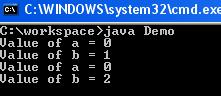
//public static void increment(){

//a++;

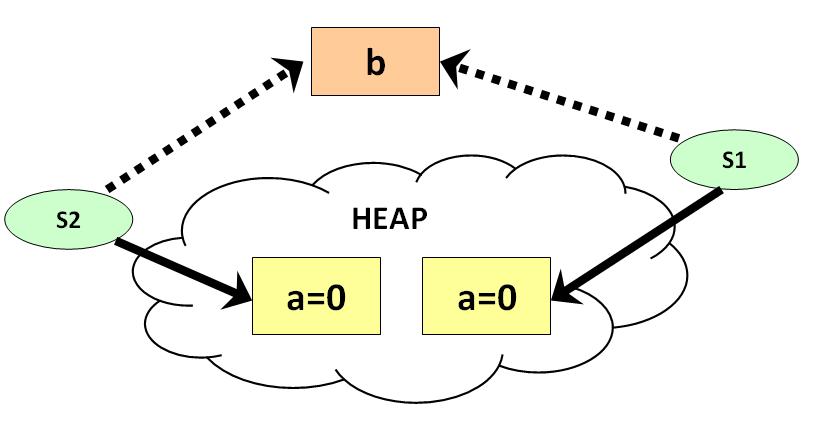
//}

}

**Step 2)** Save & Compile the code. Run the code as, **java Demo**.  
  
**Step 3)** Expected output show below

[](https://www.guru99.com/images/uploads/2012/07/java-static-variable.jpg)

Following diagram shows, how reference variables & objects are created, and static variables are accessed by the different instances.

[](https://www.guru99.com/images/uploads/2012/07/static.jpg)

**Step 4)** It is possible to access a static variable from outside the class using the syntax **ClassName.Variable\_Name**. Uncomment line # 7 & 8. Save, Compile & Run. Observe the output.

Value of a = 0

Value of b = 1

Value of a = 0

Value of b = 2

Value of a = 0

Value of b = 3

**Step 5)** Uncomment line 25,26 & 27. Save, Compile & Run.

error: non-static variable a cannot be referenced from a static context a++;

**Step 6)** Error =? This is because it is not possible to access instance variable "**a**" from java static class method "**increment**".

**Important Points:**

* Static method(s) are associated to the class in which they reside i.e. they can be called even without creating an instance of the class i.e. ClassName.methodName(args).
* They are designed with aim to be shared among all Objects created from the same class.
* Static methods cannot be overridden. But can be overloaded since they are resolved using **static binding** by compiler at compile time.

Note: Static variables and their values (primitives or references) defined in the class are stored in **PermGen** space of memory.

**When to use static methods??**

* When you have code that can be shared across all instances of the same class, put that portion of code into static method.
* They are basically used to access static field(s) of the class.

**Instance method vs Static method**

* Instance method can access the instance methods and instance variables directly.
* Instance method can access static variables and static methods directly.
* Static methods can access the static variables and static methods directly.
* Static methods can’t access instance methods and instance variables directly. They must use reference to object. And static method can’t use this keyword as there is no instance for ‘this’ to refer to.

References

* <https://docs.oracle.com/javase/tutorial/java/javaOO/classvars.html>

## **Java Static Block**

The static block is a block of statement inside a Java class that will be executed when a class is first loaded into the JVM

class Test {

static {

//Code goes here

}

}

A **static block helps to initialize the static data members**, just like constructors help to initialize instance members

Following program is the example of java static block.

**Example: How to access static block**

public class Demo {

static int a;

static int b;

static {

a = 10;

b = 20;

}

public static void main (String args []) {

System.out.println("Value of a = " + a);

System.out.println("Value of b = " + b);

}

}

you will get following output of the program.

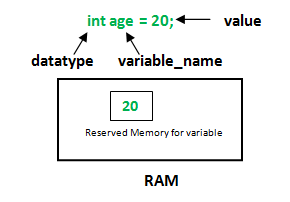
Value of a = 10

Value of b = 20

Variables in Java

A variable is the name given to a memory location. It is the basic unit of storage in a program.

* The value stored in a variable can be changed during program execution.
* A variable is only a name given to a memory location, all the operations done on the variable effects that memory location.
* In Java, all the variables must be declared before use.

**How to declare variables?**  
We can declare variables in java as follows:  
[](http://cdncontribute.geeksforgeeks.org/wp-content/uploads/Variables-in-Java.png)

**datatype**: Type of data that can be stored in this variable.  
**variable\_name**: Name given to the variable.  
**value**: It is the initial value stored in the variable.

**Examples**:

float simpleInterest; //Declaring float variable

int time = 10, speed = 20; //Declaring and Initializing integer variable

char var = 'h'; // Declaring and Initializing character variable

**Types of variables**

There are three types of variables in Java:

* Local Variables
* Instance Variables
* Static Variables

Let us now learn about each one of these variables in detail.

1. **Local Variables**: A variable defined within a block or method or constructor is called local variable.
   * These variables are created when the block in entered or the function is called and destroyed after exiting from the block or when the call returns from the function.
   * The scope of these variables exists only within the block in which the variable is declared. i.e. we can access these variables only within that block.

**Sample Program 1:**

|  |
| --- |
| public class StudentDetails  {      public void studentAge ()      {   //local variable age          int age = 0;          age = age + 5;          System.out.println("Student age is : " + age);      }        public static void main(String args[])      {          StudentDetails obj = new StudentDetails();          obj.studentAge();      }  } |

Copy Code Run on IDE

Output:

Student age is : 5

In the above program the variable age is local variable to the function StudentAge(). If we use the variable age outside StudentAge() function, the compiler will produce an error as shown in below program.

**Sample Program 2:**

|  |
| --- |
| public class StudentDetails  {      public void studentAge()      {   //local variable age          int age = 0;          age = age + 5;      }        public static void main(String args[])      {          //using local variable age outside it's scope          System.out.println("Student age is : " + age);      }  } |

Copy Code Run on IDE

Output:

error: cannot find symbol

" + age);

1. **Instance Variables**: Instance variables are non-static variables and are declared in a class outside any method, constructor or block.
   * As instance variables are declared in a class, these variables are created when an object of the class is created and destroyed when the object is destroyed.
   * Unlike local variables, we may use access specifiers for instance variables. If we do not specify any access specifier then the default access specifier will be used.

**Sample Program**:

|  |
| --- |
| import java.io.\*;  class Marks  {      //These variables are instance variables.      //These variables are in a class and are not inside any function      int engMarks;      int mathsMarks;      int phyMarks;  }    class MarksDemo  {      public static void main(String args[])      {   //first object          Marks obj1 = new Marks();          obj1.engMarks = 50;          obj1.mathsMarks = 80;          obj1.phyMarks = 90;            //second object          Marks obj2 = new Marks();          obj2.engMarks = 80;          obj2.mathsMarks = 60;          obj2.phyMarks = 85;            //displaying marks for first object          System.out.println("Marks for first object:");          System.out.println(obj1.engMarks);          System.out.println(obj1.mathsMarks);          System.out.println(obj1.phyMarks);            //displaying marks for second object          System.out.println("Marks for second object:");          System.out.println(obj2.engMarks);          System.out.println(obj2.mathsMarks);          System.out.println(obj2.phyMarks);      }  } |

Copy Code Run on IDE

Output:

Marks for first object:

50

80

90

Marks for second object:

80

60

85

As you can see in the above program the variables, *engMarks*, *mathsMarks,* *phyMarks*are instance variables. In case we have multiple objects as in the above program, each object will have its own copies of instance variables. It is clear from the above output that each object will have its own copy of instance variable.

1. **Static Variables**: Static variables are also known as Class variables.
   * These variables are declared similarly as instance variables, the difference is that static variables are declared using the static keyword within a class outside any method constructor or block.
   * Unlike instance variables, we can only have one copy of a static variable per class irrespective of how many objects we create.
   * Static variables are created at start of program execution and destroyed automatically when execution ends.

To access static variables, we need not to create any object of that class, we can simply access the variable as:

class\_name.variable\_name;

**Sample Program**:

|  |
| --- |
| import java.io.\*;  class Emp {       // static variable salary     public static double salary;     public static String name = "Harsh";  }    public class EmpDemo  {       public static void main(String args[]) {          //accessing static variable without object        Emp.salary = 1000;        System.out.println(Emp.name + "'s average salary:" + Emp.salary);     }    } |

Copy Code Run on IDE

output:

Hersh’s average salary:1000.0

**Instance variable Vs Static variable**

* Each object will have its **own copy** of instance variable whereas We can only have **one copy** of a static variable per class irrespective of how many objects we create.
* Changes made in an instance variable using one object will **not be reflected** in other objects as each object has its own copy of instance variable. In case of static, changes**will be reflected** in other objects as static variables are common to all object of a class.
* We can access instance variables **through object references** and Static Variables can be accessed **directly using class name.**
* Syntax for static and instance variables:
* class Example
* {
* static int a; //static variable
* int b; //instance variable
* }

**Practice Quiz for Methods, Variables, Modifiers in java**

<https://www.proprofs.com/quiz-school/story.php?title=java-methods-introduction>

<https://www.geeksforgeeks.org/java-gq/functions-2-gq/>

<https://study.com/academy/practice/static-vs-non-static-methods-in-java.html>

<https://www.geeksforgeeks.org/output-java-programs-set-48-static-keyword/>

<http://java.meritcampus.com/core-java-questions/Scope-of-variables?t=66>

<https://study.com/academy/practice/quiz-worksheet-variable-types-in-java.html>

<http://www.sarmaroof.com/java-quiz-3-the-difference-between-static-and-instance-variables/>

<https://javaconceptoftheday.com/java-practice-questions-on-access-modifiers/>

<https://study.com/academy/practice/java-keywords-protected-public-private.html>