**static Keyword in Java**

Last Updated : 04 Oct, 2024

The **static keyword** in Java is mainly used for memory management. The static keyword in Java is used to share the same variable or method of a given class. The users can apply static keywords with variables, methods, blocks, and nested classes. The static keyword belongs to the class rather than an instance of the class. The static keyword is used for a constant variable or a method that is the same for every instance of a class.

**The *static* keyword is a non-access modifier in Java that is applicable for the following:**

1. Blocks
2. Variables
3. Methods
4. Classes

***Note:*** *To create a static member(block, variable, method, nested class), precede its declaration with the keyword static.*

**Characteristics of static keyword:**

The static keyword is used for memory management and helps create class-level variables and methods. If you’re looking to explore more about how to use static effectively, the [**Java Programming Course**](https://gfgcdn.com/tu/S62/) provides detailed explanations with real-world examples.

Here are some characteristics of the static keyword in Java:

* **Shared memory allocation**: Static variables and methods are allocated memory space only once during the execution of the program. This memory space is shared among all instances of the class, which makes static members useful for maintaining global state or shared functionality.
* **Accessible without object instantiation:** Static members can be accessed without the need to create an instance of the class. This makes them useful for providing utility functions and constants that can be used across the entire program.
* **Associated with class, not objects:** Static members are associated with the class, not with individual objects. This means that changes to a static member are reflected in all instances of the class, and that you can access static members using the class name rather than an object reference.
* **Cannot access non-static members:** Static methods and variables cannot access non-static members of a class, as they are not associated with any particular instance of the class.
* **Can be overloaded, but not overridden**: Static methods can be overloaded, which means that you can define multiple methods with the same name but different parameters. However, they cannot be overridden, as they are associated with the class rather than with a particular instance of the class.

When a member is declared static, it can be accessed before any objects of its class are created, and without reference to any object. For example, in the below java program, we are accessing static method *m1()* without creating any object of the *Test* class.

Java

1

// Java program to demonstrate that a static member

2

// can be accessed before instantiating a class

3

​

4

class Test

5

{

6

// static method

7

static void m1()

8

{

9

System.out.println("from m1");

10

}

11

​

12

public static void main(String[] args)

13

{

14

// calling m1 without creating

15

// any object of class Test

16

m1();

17

}

18

}

**Output**

from m1

**Static blocks**

If you need to do the computation in order to initialize your **static variables**, you can declare a static block that gets executed exactly once, when the class is first loaded.

Consider the following java program demonstrating the use of static blocks.

Java

1

// Java program to demonstrate use of static blocks

2

​

3

class Test

4

{

5

// static variable

6

static int a = 10;

7

static int b;

8

9

// static block

10

static {

11

System.out.println("Static block initialized.");

12

b = a \* 4;

13

}

14

​

15

public static void main(String[] args)

16

{

17

System.out.println("from main");

18

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

19

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

20

}

21

}

**Output**

Static block initialized.

from main

Value of a : 10

Value of b : 40

For a detailed article on static blocks, see [static blocks](https://www.geeksforgeeks.org/static-blocks-in-java/)

**Static variables**

When a variable is declared as static, then a single copy of the variable is created and shared among all objects at the class level. Static variables are, essentially, global variables. All instances of the class share the same static variable.

**Important points for static variables:**

* We can create static variables at the class level only. See [here](https://www.geeksforgeeks.org/g-fact-47/)
* static block and static variables are executed in the order they are present in a program.

Below is the Java program to demonstrate that static block and static variables are executed in the order they are present in a program.

Java

1

// Java program to demonstrate execution

2

// of static blocks and variables

3

​

4

class Test

5

{

6

// static variable

7

static int a = m1();

8

9

// static block

10

static {

11

System.out.println("Inside static block");

12

}

13

14

// static method

15

static int m1() {

16

System.out.println("from m1");

17

return 20;

18

}

19

20

// static method(main !!)

21

public static void main(String[] args)

22

{

23

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

24

System.out.println("from main");

25

}

26

}

**Output**

from m1

Inside static block

Value of a : 20

from main

**Static methods**

When a method is declared with the *static* keyword, it is known as the static method. The most common example of a static method is the *main( )* method. As discussed above, Any static member can be accessed before any objects of its class are created, and without reference to any object. Methods declared as static have several restrictions:

* They can only directly call other static methods.
* They can only directly access static data.
* They cannot refer to [this](https://www.geeksforgeeks.org/this-reference-in-java/) or [super](https://www.geeksforgeeks.org/super-keyword/) in any way.

Below is the java program to demonstrate restrictions on static methods.

Java

1

// Java program to demonstrate restriction on static methods

2

​

3

class Test

4

{

5

// static variable

6

static int a = 10;

7

8

// instance variable

9

int b = 20;

10

11

// static method

12

static void m1()

13

{

14

a = 20;

15

System.out.println("from m1");

16

17

// Cannot make a static reference to the non-static field b

18

b = 10; // compilation error

19

20

// Cannot make a static reference to the

21

// non-static method m2() from the type Test

22

m2(); // compilation error

23

24

// Cannot use super in a static context

25

System.out.println(super.a); // compiler error

26

}

27

28

// instance method

29

void m2()

30

{

31

System.out.println("from m2");

32

}

33

34

35

36

public static void main(String[] args)

37

{

38

// main method

39

}

40

}

**Output:**

prog.java:18: error: non-static variable b cannot be referenced from a static context  
 b = 10; // compilation error  
 ^  
prog.java:22: error: non-static method m2() cannot be referenced from a static context  
 m2(); // compilation error  
 ^  
prog.java:25: error: non-static variable super cannot be referenced from a static context  
 System.out.println(super.a); // compiler error   
 ^  
prog.java:25: error: cannot find symbol  
 System.out.println(super.a); // compiler error   
 ^  
 symbol: variable a  
4 errors

**When to use static variables and methods?**

Use the static variable for the property that is common to all objects. For example, in class Student, all students share the same college name. Use static methods for changing static variables.

Consider the following java program, that illustrates the use of *static* keywords with variables and methods.

Java

1

// A java program to demonstrate use of

2

// static keyword with methods and variables

3

​

4

// Student class

5

class Student {

6

String name;

7

int rollNo;

8

​

9

// static variable

10

static String cllgName;

11

​

12

// static counter to set unique roll no

13

static int counter = 0;

14

​

15

public Student(String name)

16

{

17

this.name = name;

18

​

19

this.rollNo = setRollNo();

20

}

21

​

22

// getting unique rollNo

23

// through static variable(counter)

24

static int setRollNo()

25

{

26

counter++;

27

return counter;

28

}

29

​

30

// static method

31

static void setCllg(String name) { cllgName = name; }

32

​

33

// instance method

34

void getStudentInfo()

35

{

36

System.out.println("name : " + this.name);

37

System.out.println("rollNo : " + this.rollNo);

38

​

39

// accessing static variable

40

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

41

}

42

}

43

​

44

// Driver class

45

public class StaticDemo {

46

public static void main(String[] args)

47

{

48

// calling static method

49

// without instantiating Student class

50

Student.setCllg("XYZ");

51

​

52

Student s1 = new Student("Alice");

53

Student s2 = new Student("Bob");

54

​

55

s1.getStudentInfo();

56

s2.getStudentInfo();

57

}

58

}

**Output**

name : Alice

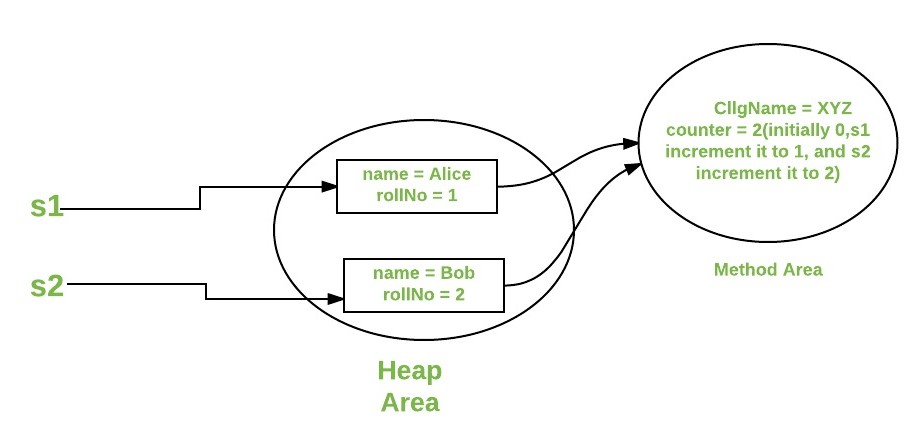
rollNo : 1

cllgName : XYZ

name : Bob

rollNo : 2

cllgName : XYZ



**Static Classes**

A class can be made **static** only if it is a nested class. We cannot declare a top-level class with a static modifier but can declare [nested classes](https://www.geeksforgeeks.org/nested-classes-java/) as static. Such types of classes are called Nested static classes. Nested static class doesn’t need a reference of Outer class. In this case, a static class cannot access non-static members of the Outer class.

***Note:*** *For static nested class, see a* [*static nested class in java*](https://www.geeksforgeeks.org/static-class-in-java/)

**Implementation:**

Java

1

// A java program to demonstrate use

2

// of static keyword with Classes

3

​

4

import java.io.\*;

5

​

6

public class GFG {

7

​

8

private static String str = "GeeksforGeeks";

9

​

10

// Static class

11

static class MyNestedClass {

12

13

// non-static method

14

public void disp(){

15

System.out.println(str);

16

}

17

}

18

19

public static void main(String args[])

20

{

21

GFG.MyNestedClass obj

22

= new GFG.MyNestedClass();

23

obj.disp();

24

}

25

}

**Output**

GeeksforGeeks

**Here’s an example Java program that demonstrates the use of the static keyword:**

Java

1

public class ExampleClass {

2

public static int count = 0;

3

public int id;

4

​

5

public ExampleClass() {

6

count++;

7

id = count;

8

}

9

​

10

public static void printCount() {

11

System.out.println("Number of instances: " + count);

12

}

13

​

14

public void printId() {

15

System.out.println("Instance ID: " + id);

16

}

17

​

18

public static void main(String[] args) {

19

ExampleClass e1 = new ExampleClass();

20

ExampleClass e2 = new ExampleClass();

21

ExampleClass e3 = new ExampleClass();

22

​

23

e1.printId();

24

e2.printId();

25

e3.printId();

26

​

27

ExampleClass.printCount();

28

}

29

}

**Output**

Instance ID: 1

Instance ID: 2

Instance ID: 3

Number of instances: 3

**Advantages:**

* **Memory efficiency**: Static members are allocated memory only once during the execution of the program, which can result in significant memory savings for large programs.
* **Improved performance**: Because static members are associated with the class rather than with individual instances, they can be accessed more quickly and efficiently than non-static members.
* **Global accessibility:** Static members can be accessed from anywhere in the program, regardless of whether an instance of the class has been created.
* **Encapsulation**of utility methods: Static methods can be used to encapsulate utility functions that don’t require any state information from an object. This can improve code organization and make it easier to reuse utility functions across multiple classes.
* **Constants**: Static final variables can be used to define constants that are shared across the entire program.
* **Class-level functionality**: Static methods can be used to define class-level functionality that doesn’t require any state information from an object, such as factory methods or helper functions.