**Java static keyword**

**Static modifier modifies the access and allows to call without the help of an object**.

"**Static**" keyword can be applied to instance variables and methods and [not with classes](http://way2java.com/oops-concepts/java-made-simple-is-it-possible-static-class-in-java/) and also [not with local variables.](http://way2java.com/oops-concepts/does-java-support-static-local-variables-inside-method/).

The **static keyword** in java is used for memory management mainly. We can apply java static keyword with variables, methods, blocks and nested class. The static keyword belongs to the class than instance of the class.

**The static can be:**

1. Variable (also known as class variable)
2. Method (also known as class method)
3. Block
4. Nested class

## Applicable to

The Static keyword can be applied to

* **Method**
* **Variable**
* **Class nested within another Class**
* **Initialization Block**

## Not Applicable to

The Static keyword can not be applied to

* ***Class (Not Nested)***
* ***Constructor***
* ***Interfaces***
* ***Method Local Inner Class(Difference then  nested class)***
* ***Inner Class methods***
* ***Instance Variables***
* ***Local Variables***

## 1) Java static variable

If you declare any variable as static, it is known **static variable**.

* The static variable can be used to refer the common property of all objects (that is not unique for each object)

**Example:** company name of employees, college name of students etc.

* The static variable gets memory only once in class area at the time of class loading.

### Advantage of static variable

It makes your program **memory efficient** ( i.e it saves memory).

Using static variable we make our program memory efficient (i.e it saves memory).

#### Understanding problem without static variable

***class****Student{*

***int****rollno;*

*String name;*

*String college="ITS";*

*}*

Suppose there are 500 students in my college, now all instance data members will get memory each time when object is created. All student have its unique rollno and name so instance data member is good.

Here, college refers to the common property of all objects. If we make it static, this field will get memory only once.

#### Note: Java static property is shared to all objects.

### Example of static variable

***//Program of static variable***

***class*** *Student8{*

***int*** *rollno;*

*String name;*

***static*** *String college ="ITS";*

*Student8(****int*** *r,String n){*

*rollno = r;*

*name = n;*

*}*

***void*** *display (){*

*System.out.println(rollno+" "+name+" "+college);*

*}*

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

*Student8 s1 =* ***new*** *Student8(111,"Karan");*

*Student8 s2 =* ***new*** *Student8(222,"Aryan");*

*s1.display();*

*s2.display();*

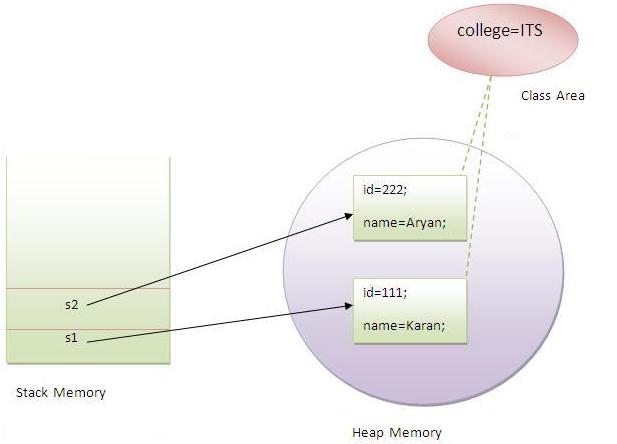
*}*

*}*

**Output**:

*111 Karan ITS*

*222 Aryan ITS*



### Example of static variable.

In the below example College\_Name is always same, and it is declared as static.

## Examp*le*

***class*** *Student{*

***int*** *roll\_no;*

*String name;*

***static*** *String College\_Name="ITM";*

*}*

***class*** *StaticDemo{*

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

*Student s1=****new*** *Student();*

*s1.roll\_no=100;*

*s1.name="abcd";*

*System.****out****.println(s1.roll\_no);*

*System.****out****.println(s1.name);*

*System.****out****.println(Student.College\_Name);*

*Student s2=****new*** *Student();*

*s2.roll\_no=200;*

*s2.name="zyx";*

*System.****out****.println(s2.roll\_no);*

*System.****out****.println(s2.name);*

*System.****out****.println(Student.College\_Name);*

*}*

*}*

## Example

***Output:***

*100*

*abcd*

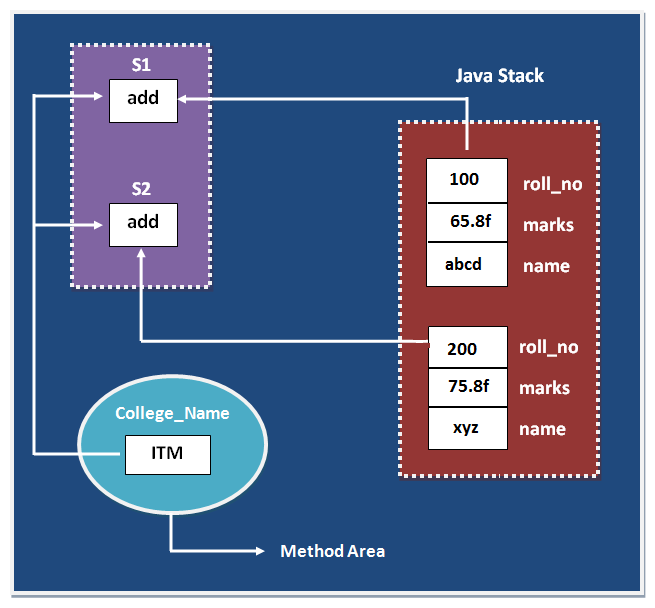
*ITM*

*200*

*zyx*

*ITM*

In the above example College\_Name variable is commonly sharable by both S1 and S2 objects.



In the above image static data variable are store in method are and non static variable is store in java stack.

**Java permits to call a static variable or method in three ways.**

1. Can be called without the help of an object
2. Can be called with the help of an object
3. Can be called with class name

The following program explains the**static**keyword with variables.

*public class StaticDemo{*

*static int price = 100;*

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

*System.out.println(price);* ***// without object***

*StaticDemo sd1 = new StaticDemo();*

*System.out.println(sd1.price);* ***// with object***

*System.out.println(StaticDemo.price);* ***// with class name***

*}*

*}*

### Java static Variable Method

The instance variable **price** is declared as static and is called in three ways – without object, with object and with class name.

###### **Example on Static methods usage (of Java static Variable Method)**

The same style of static variables is followed with static methods also and can be called in three ways. Following program explains.

*public class StaticDemo2{*

*public static void show() {*

*System.out.println("Hello World");*

*}*

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

*show();* ***// without object***

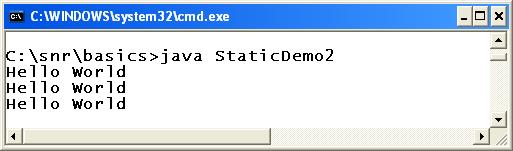
*StaticDemo2 sd1 = new StaticDemo2();*

*sd1.show();* ***// with object***

*StaticDemo2.show();*  ***// with class name***

*}*

*}*



The **show()** method is declared as static and is called in three ways. It is allowed with class name also to facilitate to call many methods of Java API classes without the help of an object.

Many methods are declared static in Java classes like **[java.lang.Character](http://way2java.com/java-lang/class-character/)** and **[java.lang.Math](http://way2java.com/java-lang/class-math/)** etc.

###### **What is Class Variable? (of Java static Variable Method)**

A static variable is also known as "**class variable**" as all the objects of the whole class refer (or share) the same variable (location); that is, one variable is used by all the objects (in encapsulation, each object will have separate location of a variable). As all the objects refer the same location (variable), no encapsulation exist with static variable.

**Can a local variable be declared static? (of Java static Variable Method)**

**A local variable cannot be static**. If static the meaning of static is lost. A static variable scope is for the objects of whole class and if declared local as static, the purpose of static is lost and compiler raises error.

**Why the main() is static? (of Java static Variable Method)**

main() method is declared as static to allow the JVM to call the main() without the help of an object. JVM is a separate process and to call the method of another process (your program), it requires an object. As the execution starts from the main(), the JVM cannot create an object without entering the main() method. If the main() method is allowed to call without the need of an object, the JVM can create hundreds of objects once enters the main(). For this reason, main() is declared as static.

###### **Memory Allocation for static Variables**

We know earlier every object will have its own copy of variables. But with static variables, it is very different.

**A static variable location is shared (or accessed or pointed) by all the objects of the class**. The result is, if one object changes the value, all the other objects also get affected. With static variables, no encapsulation exists. Observe the following program.

*public class Demo{*

*static int x = 10;*

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

*System.out.println(x); // 10*

*Demo d1 = new Demo();*

*Demo d2 = new Demo();*

*System.out.println(d1.x); // 10*

*System.out.println(d2.x); // 10*

*d1.x = 20;*

*System.out.println(d2.x); // 20*

*d2.x = 30;*

*System.out.println(d1.x); // 30*

*System.out.println(x); // 30*

*}*

*}*



Variable **x** is declared as static. Objects **d1** and **d2** point to the same **x** location. Observe, if **d1**changes **d2**and if **d2**changes **d1**are getting affected. Because all the objects of the whole class share the same location, the static variable is known as "**class variable**".

### Program of counter without static variable

In this example, we have created an instance variable named count which is incremented in the constructor. Since instance variable gets the memory at the time of object creation, each object will have the copy of the instance variable, if it is incremented, it won't reflect to other objects. So each objects will have the value 1 in the count variable.

**Example**:

***class*** *Counter{*

***int*** *count=0;//will get memory when instance is created*

*Counter(){*

*count++;*

*System.out.println(count);*

*}*

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

*Counter c1=****new*** *Counter();*

*Counter c2=****new*** *Counter();*

*Counter c3=****new*** *Counter();*

*}*

*}*

**Output**:

1

1

1

### Program of counter by static variable

|  |
| --- |
| As we have mentioned above, static variable will get the memory only once, if any object changes the value of the static variable, it will retain its value. |
| ***class*** *Counter2{*  ***static******int*** *count=0;//will get memory only once and retain its value*  *Counter2(){*  *count++;*  *System.out.println(count);*  *}*  ***public******static******void*** *main(String args[]){*  *Counter2 c1=****new*** *Counter2();*  *Counter2 c2=****new*** *Counter2();*  *Counter2 c3=****new*** *Counter2();*  *}*  *}* |
|  |

**Output:**

1

2

3

2) **Java static method**

If you apply static keyword with any method, it is known as **static method**.

* A static method belongs to the class rather than object of a class.
* A static method can be invoked without the need for creating an instance of a class.
* static method can access static data member and can change the value of it.

### Example of static method

***//Program of changing the common property of all objects(static field).***

***class*** *Student9{*

***int*** *rollno;*

*String name;*

***static*** *String college = "ITS";*

***static******void*** *change(){*

*college = "BBDIT";*

*}*

*Student9(****int*** *r, String n){*

*rollno = r;*

*name = n;*

*}*

***void*** *display (){*

*System.out.println(rollno+" "+name+" "+college);*

*}*

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

*Student9.change();*

*Student9 s1 =* ***new*** *Student9 (111,"Karan");*

*Student9 s2 =* ***new*** *Student9 (222,"Aryan");*

*Student9 s3 =* ***new*** *Student9 (333,"Sonoo");*

*s1.display();*

*s2.display();*

*s3.display();*

*}*

*}*

**Output**:

111 Karan BBDIT

222 Aryan BBDIT

333 Sonoo BBDIT

### Another example of static method that performs normal calculation

***//Program to get cube of a given number by static method***

***class*** *Calculate{*

***static******int*** *cube(****int*** *x){*

***return*** *x\*x\*x;*

*}*

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

***int*** *result=Calculate.cube(5);*

*System.out.println(result);*

*}*

*}****Output:***

*125*

### *Restrictions for static method*

|  |
| --- |
| ***There are two main restrictions for the static method. They are:*** |

|  |
| --- |
| 1. *The static method cannot use non static data member or call non-static method directly.* 2. *this and super cannot be used in static context.* |

***class*** *A{*

***int*** *a=40;//non static*

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

*System.out.println(a);*

***//CE: Cannot make a static reference to the non-static field a***

*}*

*}***Output:** Compile Time Error

### Q) Why java main method is static?

|  |
| --- |
| **Ans:** Because object is not required to call static method, if it were non-static method, jvm create object first then call main() method that will lead the problem of extra memory allocation. |

**3) Java static block**

* Is used to initialize the static data member.
* It is executed before main method at the time of classloading.

### Example of static block

***class*** *A2{*

***static****{*

*System.out.println("static block is invoked");*

*}*

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

*System.out.println("Hello main");*

*}*

*}*

**Output**:  
 static block is invoked

Hello main

### Q) Can we execute a program without main() method?

A) Yes, one of the way is static block but in previous version of JDK not in JDK 1.7.

***class*** *A3{*

***static****{*

*System.out.println("static block is invoked");*

*System.exit(0);*

*}*

*}*

**Output**:

*static block is invoked (if not JDK7)*

**In JDK7 and above, output will be:**

**Output:**

**Error**: Main method not found in class A3, please define the main method as:

public static void main(String[] args)

**Why main() is static? If the main() method is allowed to call without the need of an object, the JVM can create hundreds of objects once enters the main(). For this reason, main() is declared as static.**

1. Your program, say, a Demo program and JVM are two separate processes in the RAM. We know execution starts from main() method. To call the main(), JVM requires an object because JVM is an outside (separate) process not connected to Demo process. So to call main(), JVM requires an object. How JVM gets an object of Demo unless it enters into the execution space of Demo. The execution space of Demo contains main(). For this reason, main() is declared static to allow the JVM to call main() without the need of an object.

**Static key word can be used with which access specifiers? and why?**

1. Static can be used with all access specifiers. Only thing is static methods cannot be overridden and a constructor, local variables and classes cannot be static.

**Why the local variables and class are not static??**

1. Static, in Java, is known as access modifier. Access modifier modifies the access.

Let us see what it modifies. A static variable and a static method can be called without the help of an object and moreover, a static variable will have only location that can be accessed by all the objects of the class. Local variable scope and life is within the method where it is declared.

How can a local variable will have only one location that is accessed by all objects? To use a class, its object is to be created with new keyword. How can you call a method without the help of an object? Object identifies for which class the method belongs. JVM searches that class and loads it.

**The static variable and static method can be called with the help of an object and without object also then what is the difference between these two callings (with object and without object)**

1. Without object helps you to call in the same class. Suppose you want call in a different class by composition, then how you call? For the method you asked, where the JVM search.

**I’m not use “static”in my main method what will happen ?**

1. Your program will be compiled but not executed.

##### **In Java, static is keyword that can be applied to variables and methods but not to classes. static is used as an** [**access modifier**](http://way2java.com/oops-concepts/specifiers-modifiers/access-specifier-vs-access-modifier-in-java/)**.**

In Java, to call an instance variable or method, an **object is required**. But a static variable and method can be called without the help of an object. This is the basic difference between static and non-static in Java.

###### **Observe the following code on Static Method and Static Variable.**

*public class Demo{*

*int marks = 50;*  ***// non-static variable***

*static double average = 40.6;*  ***// static variable***

*public void display(){* ***// non-static method***

*System.out.println("Hello 1");*

*}*

*public static void show(){* ***// static method***

*System.out.println("Hello 2");*

*}*

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

***// static members - no object is required***

*System.out.println(average);*

*show();*

***// non-static members - object is required***

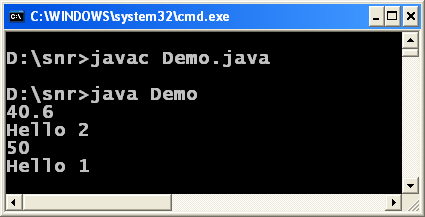
*Demo d1 = new Demo();* ***// create the object***

*System.out.println(d1.marks);*

*d1.display();*

*}*

*}*



Observe, **average** and **show()** are static members of the class **Demo** and are called without the need of an object. **marks** and **display()**are non-static members and are called with object **d1**.

## Does Java support static local variables inside method?

**Or**

[**Can a local variable be static?**](http://way2java.com/oops-concepts/does-java-support-static-local-variables-inside-method/)

**Ans.** Few points about static variables.

1. A static variable does not maintain **encapsulation**.
2. There will be only one location for static variable accessed by all the objects of the whole class. For this reason, static variables are known as "**class variables**"; in the sense, the static variable is only one for all the objects of the class.
3. Only instance variable is declared static so that its scope is for all the methods and for all the objects.

Now let us go into the answer. By rule, when the static variable **scope** (or visible) is for the whole class, by common sense, how a local variable be declared static. A local variable scope is within the method where it is declared. If the local variable is declared static, the**meaning of static is lost**. If the local variable is static, the purpose of static variable is bypassed. For this reason, compiler does not allow static local variables.

Finally, Local variables cannot be static in Java. Only instance variables can be static.

**Following code raises compilation error.**

*public void display(){*

*static int marks = 80;* ***// raises compilation error***

*}*

**Do the local variables declared in a static method are implicitly static or can we assume static?**

If you think implicitly static, it is a **compilation error** as discussed in the previous question that local variables cannot be static. The local variables in a static method are simply **just local variables** (non-static) only equivalent to declared in a non-static method. That is, the variables declared in a static method do not have any special meaning anyhow.

Similarly, passing a static variable to a**method parameter**does not have any special effect. It is simply as if passed a non-static variable.

If a Programmer needs **static variables**, he must declare them as **instance variables** only outside any method.

## Is it possible static class in Java?

1. The answer is both "YES" and "NO". Let us see.

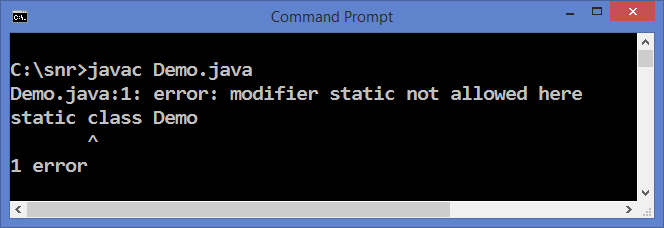
Java comes with two types of classes – **top-level classes** and **inner classes** (also known as nested classes). **A top-level class cannot be static where as a nested class can be static.**

###### Top-level class cannot be static. See the compilation error on static class.

*static class Demo{*

*// some code here*

*}*



It says access modifier static is not allowed here.

###### **Static nested classes**

A class declared in another class is known as nested class.

*class OuterOne* ***// top-level class or known as outer*** *class{*

*class InnerOne{*  ***// inner class or nested class***

*}*

*}*

In the above code, **OuterOne** cannot be static and **InnerOne**can be declared static as follows.

*class OuterOne{* ***// top-level class or known as outer class***

*static class InnerOne{* ***// inner static class***

*}*

*}*

That is, **only nested classes can be declared static**. But in general sense, it is not possible.

Nested classes are given a chance because to permit to call the nested classes without creating object of outer class.

## What is Java static with Example?

##### Java static is access modifier. Access specifier specifies access.

The "static" is one of the keywords ([not reserved and prohibited](http://way2java.com/java-introduction/java-keywords/)) of Java. It is used as an access modifier. An access modifier modifies the access and it [differs](http://way2java.com/oops-concepts/specifiers-modifiers/access-specifier-vs-access-modifier-in-java/) from access specifier in that access specifier specifies the access.

Let us see what static modifies the access. Everyone knows, to call an instance variable and method in Java, an object is required because Java is an OOPs language. Java maintains encapsulation [through calling with object](http://way2java.com/oops-concepts/java-encapsulation-abstraction-data-hiding-and-data-binding/). But sometimes, it may be required to call the instance variable or method without object. Then, declare the variable or method as static. That is, **a static variable or static method does not require an object to call**.

**Following example shows the difference between calling Java static and non-static variable.**

*public class Demo{*

*int age = 65;* ***// non-static and requires an object to call***

*static double salary = 9999.99;****// static and does not require an object to call***

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

***// System.out.println(age); // error, it requires object as non-static***

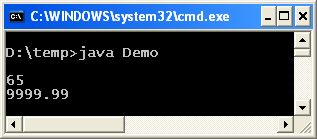
*Demo d1 = new Demo();*  ***// now, create an object of Demo class***

*System.out.println(d1.age);* ***// it works fine because it is called with object***

*System.out.println(salary);****// works fine even not called with object as it is static***

*}*

*}*



## What is Java static keyword, static variable, static method?

##### Java static is keyword used as an [access modifier](http://way2java.com/oops-concepts/access-modifiers-%e2%80%93-meanings/). It modifies the accessing technique of variables and methods in Java coding. "static" keyword can be applied to variables, methods, inner classes and blocks.

We know, to call an instance variable and method, Java requires an object (and thus Java maintains **encapsulation**). But declaring a variable and method as static, an object is not required to call. That is, a static variable and static method can be called without the help of an object. It is the facility static gives but we loose encapsulation. Of course, it is not a minus point as sometimes between objects encapsulation is not required.

**A Java static variable or method can be called in three ways.**

1. without object
2. with object
3. with class name

**1. Using Java static variable**

*public class Employee{*

*String name = "Rao";* ***// non-static and requires an object to call***

*static int salary = 7500;* ***// static and does not require an object***

***// and can be called in three ways***

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

***// USING NON-STATIC VARIABLE***

*System.out.println("Calling non-static variable name:");*

***// System.out.println(name); // error as object is required***

*Employee emp1 = new Employee();* ***// now create an object***

*System.out.println(emp1.name);* ***// its fine and works***

***// USING STATIC VARIABLE IN THREE WAYS***

*System.out.println("\nCalling static variable salary in three ways:");*

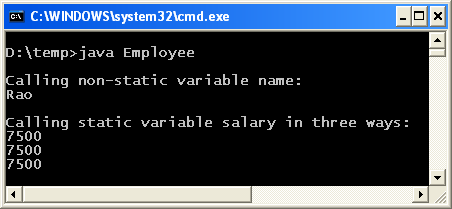
*System.out.println(salary);* ***// calling without object***

*System.out.println(emp1.salary);* ***// calling with object***

*System.out.println(Employee.salary);*  ***// calling with class name***

*}*

*}*



**2. Using Java static methods**

*public class Employee{*

*public void display(){*

***// non-static method and requires an object to call***

*System.out.println("Hello 1");*

*}*

*public static void show(){*

***// static method and requires NO object to call***

***// and can be called in three ways***

*System.out.println("Hello 2");*

*}*

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

***// USING NON-STATIC VARIABLE***

*System.out.println("\nCalling non-static method display() method:");*

***// display(); // error as object is required***

*Employee emp1 = new Employee();* ***// now create an object***

*emp1.display();* ***// its fine and works***

***// USING STATIC VARIABLE IN THREE WAYS***

*System.out.println("\nCalling static method show() in three ways:");*

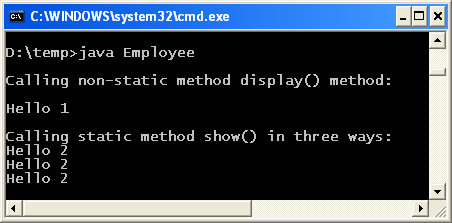
*show();*  ***// calling without object***

*emp1.show();*  ***// calling with object***

*Employee.show();* ***// calling with class name***

*}*

*}*



Why static variable is called a “class variable”?

**3. Does static maintain Encapsulation?**

First let us write an example and then discuss.

*public class Employee{*

*static int salary = 7500;*

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

*System.out.println(salary); //* ***prints 7500***

*Employee emp1 = new Employee();* ***// create two objects***

*Employee emp2 = new Employee();*

*System.out.println(emp1.salary);* ***// prints 7500***

*System.out.println(emp2.salary);* ***// prints 7500***

*emp1.salary = 8500;* ***// change emp1 salary***

*System.out.println(emp2.salary);*

***// emp2 salary also gets effected; prints 8500***

*emp2.salary = 9500;*  ***// change emp2 salary***

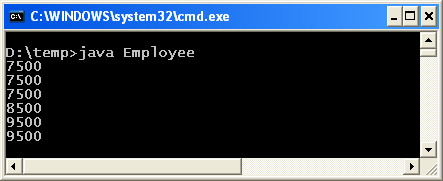
*System.out.println(emp1.salary);*

***// emp1 salary also gets effected; prints 9500***

*System.out.println(salary);* ***// prints 9500***

*}*

*}*



In the above code, observe the screenshot, if **emp1 salary** changes **emp2 salary** is also gets effected and similarly when**emp2 salary** is changed, **emp1 salary** gets effected. This is not encapsulation. With static variable encapsulation is not maintained. Why it is so? It is because the same salary variable (or address) is referred (or pointed) by all the objects of the class. That is, all objects of the **whole class** uses the same location. For this reason, static variable is known as **class variable**. Note that, with non-static variable, each object maintains its own location (or copy) separately.

1. **When to use Java static variables?**
2. When encapsulation is not required.   
   **For example**, three **partners** exist in a business with a **joint account**. If the balance in the account is 1 lakh, if one partner draws one lakh, other partners cannot draw one lakh again. That is, between all the partners there should be a common balance variable and it is achieved with static keyword as static balance.
3. Many methods of many classes in **Java API**are declared static to allow the Programmer to call them without then need of creation of object. Ex: **Character** and **Math** etc.

## Difference between static and final Keywords

###### static vs final java, both are known **access modifiers** in Java doing different functionalities.

###### **static:**static keyword can be applied to instance variables and methods but not to classes. When applied, variables and methods can be called without the help of an object. When a method or variable is called without object, encapsulation is not maintained. That is, with static variables and methods, **encapsulation** does not exist.

**final:**final keyword can be applied to all constructs – variables, methods and classes. When applied, final behaves very differently with each with different functionalities.

**Let us explore static vs final java programmatically.**

**I) Usage of static Keyword**

Generally, an instance variable or method requires an object to call with. But static variables and methods do not require an object. Observe, the following code.

*public class Employee{*

*int age = 45;*  ***// non-static variable***

*static double salary = 8877.66;*  ***// static variable***

*public static void display(){* ***// static method***

*System.out.println("Hello World");*

*}*

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

***// System.out.println(age);***

***// raises error,it is non-static and requires object***

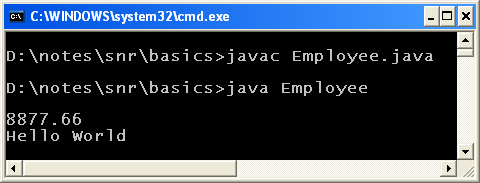
*System.out.println(salary);*

***// works fine as salary is static, object not required***

*display();****// works fine as display() is static, object not required***

*}*

*}*



In the above code, **age is non-static** and requires an object to call with. Observe, the main() method. **salary being static**does not require an object. Similarly with **static display()** method. It is called wihtout the need of an object.

**II) Usage of final Keyword**

As stated earlier, the final keyword can be applied in three places – variables, methods and classes. When applied it behaves differently.

1. A final variable cannot be reassigned.
2. A final method cannot be overridden.
3. A final class cannot be inherited.

But one thing can be found in common among the above three, "final means something cannot be done".

**a) Final with variables**

A **final variable** works like a **constant** of C/C++. That is, a final variable cannot be reassigned. Java doesnot support **const** keyword and its place uses **final**keyword.

*public class Bank{*

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

*int rupees = 500;*  ***// non-final can be reassigned***

*System.out.println("rupees before reassigned: " + rupees);*

*rupees = 600;*  ***// non-final***

*System.out.println("rupees after reassigned: " + rupees);*

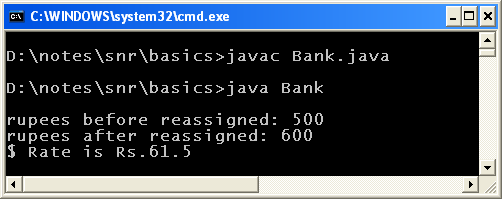
*final double $rate = 61.5;*  ***// final cannot be reassigned***

*System.out.println("$ Rate is Rs." + $rate);*

***// $rate = 62.8; // raises error***

*}*

*}*



The variable **rupees is not final** and thereby reassigned with a new value of 600. **$rate is final** and thereby cannot be reassigned.

**b) final with methods**

If the super class does not permit the subclass to override, it simply declares the method as final. **The final methods of super class cannot be overridden by subclass**.

*class Manager{*

*public final void doSomeThing(){* ***// observe final***

*System.out.println("Do it right now");*

*}*

*}*

*public class Worker extends Manager{*

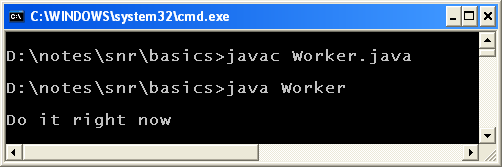
*public static void main(String args[]){*

*Worker w1 = new Worker();*

*w1.doSomeThing();*

*}*

*}*



In the super class**Manager**, **doSomeThing()** is declared **final**. The subclass **Worker**is not permitted to override. Try once, it raises compilation error.

If the super class have non-final methods (not shown in the code), the subclass can feel free to override at its convenience.

**c) Final with classes**

The programmer may not like a class to be inherited by other classes because he would like to allow the accessibility throgh composition not by inheritance.

*final class Fruit{*  ***// observe, final class***

*String nature = "Seedless";*

*}*

*public class Mango{* ***// extends Fruit raises compilation error***

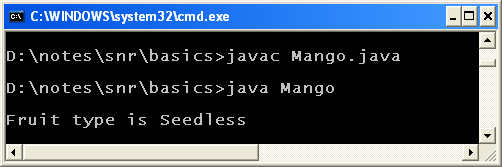
*public static void main(String args[]){*

*Fruit f1 = new Fruit();*

*System.out.println("Fruit type is " + f1.nature);*

*}*

*}*



In the above code, **Fruit is declared as final** and thereby**Mango**cannot extend it. Then how it can make use of Fruit class methods? Simply by creating an object of **Fruit** and calling **Fruit** class methods. This is known as **composition**.  
 **What is Composition in Java?**

Creating an object of other class in our class and calling other class variables and methods is known as [composition](http://way2java.com/oops-concepts/composition-has-a-relationship/) (known as has-a relationship as one class "has a" object of other class). Here, object and method or variable belong to the same class. In contrast, in inheritance object belongs to subclass and variable or method belong to super class.

[**Why main() must be static?**](http://way2java.com/oops-concepts/public-static-void-mainstring-args/)

In Java, the execution starts from **main()** method. But for compilation, main() is not required. Java's main() method syntax is quiet different from C/C++.

Following is the public static void main complete signature

Every word in the public static void main statement has got a meaning to the JVM.

1. [**public**](http://way2java.com/packages/access-specifiers-accessibility-permissions-restrictions/)**:**It is a keyword and denotes that any other class (JVM) can call the main() method without any restrictions.
2. [**static**](http://way2java.com/oops-concepts/static-keyword-%e2%80%93-philosophy/)**:**It is a keyword and denotes that any other class (JVM) can call the main() method without the help of an object. More about static is available at [static Keyword – Philosophy](http://way2java.com/oops-concepts/static-keyword-%e2%80%93-philosophy/).
3. **void:**It is a keyword and denotes that the main() method does not return a value.
4. **main():**It is the name of the method.
5. **String args[]:**The parameter is a [String array](http://way2java.com/arrays/string-array-object-string-array/) by name **args**. The string array is used to access [command-line](http://way2java.com/arrays/command-line-arguments/) arguments.

The Java program under execution takes some context area (execution area) in the RAM. JVM is in another context area. To call the main(), the JVM requires an object. When the JVM has not entered into the program, how it can create an object or get an object. To overcome this, allow the JVM to access the main() without object just by declaring static. Once the JVM enters, it can create hundreds of objects later.

Whether you pass right now **command-line arguments** or not, you must have the string array as parameter as it is the part of syntax.

Any word is missed in the above statement, the program compiles, but does not execute.

A program using main() is discussed in "[Basic Class Structure, Compilation and Execution](http://way2java.com/oops-concepts/your-first-java-program/)".

**1. public static void main(String args[]) throws [IOException](http://way2java.com/exceptions/filenotfoundexception-ioexception/):** This sort of main() method is used in "**[BufferedInputStream and BufferedOutputStream](http://way2java.com/io/bufferedinputstream-and-bufferedoutputstream/)**" and in many places.

**2. public static void main(String args[]) throws**[**Exception**](http://way2java.com/exceptions/hierarchy-of-exceptions-checkedunchecked-exceptions/)**:** This type of main() method is used in "[**Communication with TCP/IP Protocol**](http://way2java.com/networking/communication-with-tcpip-protocol/)" and also in many places.

## Can you make Static Constructor in Java?

Java **does not permit** to declare a **constructor as static**. Following are the reasons.

**1. Static means for the same class. For example,** [static methods](http://way2java.com/oops-concepts/static-keyword-%e2%80%93-philosophy/) **cannot be inherited.**

2. With static, "[this](http://way2java.com/oops-concepts/using-this-keyword/)" reference (keyword) cannot be used. "this" is always linked to an object. A constructor always belongs to some object.

**3. If a** [constructor](http://way2java.com/oops-concepts/constructors-and-constructor-overloading/) **is static, an object of subclass cannot access. If static is allowed with constructor, it is accessible within the class but not by subclass.**  
  
**The following program raises compilation error.**

*public class Demo{*

*public static Demo(){*

*System.out.println("Hello 1");*

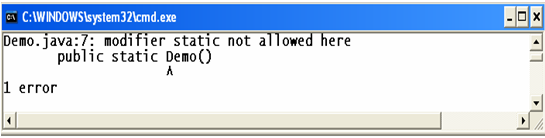
*}*

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

*Demo d1 = new Demo();*

*}*

*}*



Observe, the compiler error message. It says, **static is not allowed**with constructor.

**What other access modifiers a constructor cannot be?**

"A constructor cannot be abstract, static, final, native, strictfp, or synchronized".

**abstract class have constructor**

*abstract class A{  
 A(){  
 System.out.println(“Hello from constructor”);  
 }  
 void f1();  
 }  
class Demo extends A{  
public void f1(){  
 System.out.println(“method f1()”);  
}  
public static void main(String[] args){  
 A ob=new Demo();  
 ob.f1();  
 System.out.println(“Hello World!”);  
 }  
}*

**Interface can be declared as abstract.**

*abstract interface A1{  
 void f1();  
}  
class A implements A1{  
public void f1(){  
 System.out.println(“method f1()”);  
}  
public static void main(String[] args){  
 A ob=new A();  
 ob.f1();  
 System.out.println(“Hello World!”);  
 }  
}*

## Static vs non-static Java

In the following code, **show()** method is non-static and **display()** is static. The question is "Can we call a non-static method from static method?". Everyone says simply "no" because it requires an object to call. Anyhow, let us observe the compiler message.

*public class Demo{*

*public void show() {*

*System.out.println("Hello");*

*}*

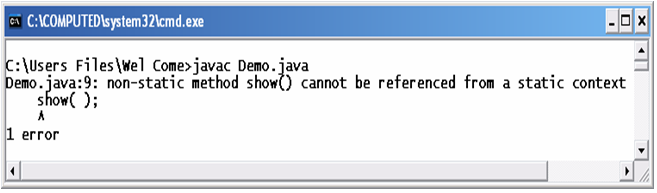
*public static void display()*

*{*

*show( );*

*}*

*}*



This is an usual compiler message every Java professional come acrosses.

**Now let us see the combinations.**

|  |  |  |
| --- | --- | --- |
| **SHOW()** | **DISPLAY()** | **RESULT** |
| non-static | non-static | works |
| static | non-static | works |
| static | static | works |
| non-static | static | does not work |

The above table can easily be remembered.**Only one combination does not work – calling non-static member from static member (it requires object)**.

Anyhow do small programs yourself and check the rules of combinations to have command over coding.

**Can a class be static?**

No. A class cannot be static. But static classes are possible with inner classes.

**What is difference between interface and inheritance?**

Interface is one the types classes with special features Java does support. Inheritance is an OOPs concept.

## Can we override static method in Java with Example?

The **super class static methods cannot be overridden by sub class** because they do not represent the state of the object or with static methods no [encapsulation](http://way2java.com/oops-concepts/three-great-principles-%e2%80%93-data-binding-data-hiding-encapsulation/) exists. For this reason, the super class static methods are not part of subclass. When they are not part of subclass, they cannot be overridden.

###### **Let us write a simple example to prove that we cannot override static method.**

*class Test{*

*public static void display() {*

*System.out.println("From Test");*

*}*

*}*

*public class Demo extends Test{*

*public static void display() {*

*System.out.println("From Demo");*

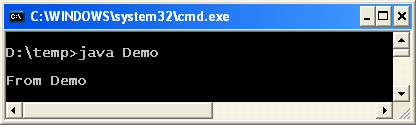
*}*

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

*display();*

*}*

*}*



The above code compiles, executes and prints "From Demo". It looks, the static display() method of Test is overridden by subclass Demo as both are identical. No, both methods are treated **distinct** from each other. That is, subclass **display()** is no way connected with super class **display()** method.

Both are treated separate and does not know each other. We cannot override static method. How to prove it? There are two ways.

**1. Using super.display():**

We know in method overriding, the super class method is hidden by subclass method. To call super class method from subclass method, we use super keyword as follows. Calling super now shows error as subclass display() is not a overridden method.

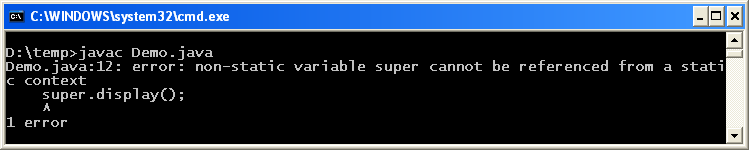
*public static void display() {*

*super.display();*  ***// cause of error***

*System.out.println("From Demo");*

*}*

**Following is the screenshot of error message.**

[](http://way2java.com/wp-content/uploads/2014/07/ima11.png)

**2. Using @Overfide:**   
  
Use @Override annotation introduced with JDK 1.5.

*public class Demo extends Test{*

*@Override*

*public static void display() {*

*System.out.println("From Demo");*

*}*

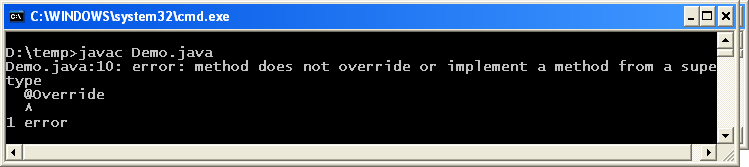
*public static void main(String args[]) {*

*display();*

*}*

*}*

Following is the screenshot of error message.



The annotation @Override means that there is an equivalent method in super class which is overridden. The error message is shown as subclass display() is distinct from super class one.

**Let us think more with the following points on static overriding.**

1. As **static methods cannot be inherited**, no question of overriding.
2. Static methods are treated as **class methods** (like static variables are known as class variables) or class level methods. So, they cannot be inherited and overridden.
3. Static methods are part of class but not part of object. As a part static methods can be**overloaded** but cannot be overridden.
4. Overridden methods support **dynamic binding** and which method is to be dispatched for execution is decided at runtime. As static methods are binded at compile time (known as**static binding**), they cannot support dynamic binding.

## Static String/Static String Array

Like static variables, **static strings**and **static string arrays** are also common in Java coding. The same rules of **static variables** are followed. The following program creates one static string array **names1**and one static string **persons**. The elements of static string array are put into static string and printed.

*public class StringArrayManipulations{*

*static String names1[] = { "Jyostna", "Srinivas", "Pradeep", "Chandu" };*

*public static String persons = "";*

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

*for(String str : names1){*

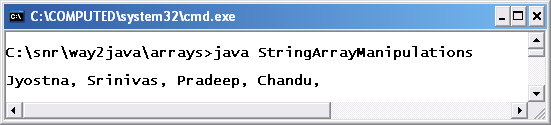
*persons += str+ ", ";*

*}*

*System.out.println(persons);*

*}*

*}*



**1. Can you code like this.**

**System.out.println(pow(2,4)); // Math class is not written**

**instead of traditional**

**System.out.println(Math.pow(2,4)); // Math is written**

Ans: Yes, it is possible from JDK1.5 version with a new feature known as [static imports](http://way2java.com/java-versions-2/jdk-1-5-java-se-5-version/).

**2. Can you use C-lang printf() in Java with all %d and %s etc options.?**

Ans: Yes, it is possible from [JDK1.5](http://way2java.com/java-versions-2/jdk-1-5-java-se-5-version/) version.

**3. Can you use C-lang enums in Java?**

Ans: Yes, it is possible from [JDK1.5](http://way2java.com/java-versions-2/jdk-1-5-java-se-5-version/) version.

**4. What is the garbage collection mechanism in Java?**  
Ans: [Garbage Collection – gc() and exit(0)](http://way2java.com/java-lang/gc-and-exit0-methods/)  
 **5. How to use the C++ destructor functionality where Java does not support destructors?**  
  
Ans: [Java Destructor – finalize()](http://way2java.com/java-lang/finalize-method/)

## Static Final String Java

###### A string can be declared both static and final. The advantages of declaring string with the both access specifiers are narrated hereunder.

**The advantages are inclusive of both the affects of final and static.**

1. A string declared final cannot be reassigned. The string works as a constant.

2. A string declared static can be called without the help of an object or with class name..

3. A static variable **does not maintain encapsulation**. Declaring a static variable as final, no object can change the value but can access it.

*public class Demo{*

*static final String str = "Hello";*

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

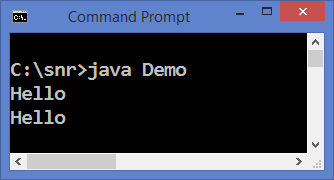
*// str = "world";* ***// gives error***

*System.out.println(str);* ***// called without the help of an object***

*System.out.println(Demo.str);****// called with class name***

*}*

*}*



The first statement in the main() method gives error as string **str** is declared as final.

**Note:** The order of access specifiers is not important. Both the following are correct.

*static final String str = "Hello";*

*final static String str = "Hello";*

*The following statement raises compilation error as string****str****is declared as final.*

*Demo d1 = new Demo();*

*d1.str = "World";*

**Some Realtime examples**

*static final String rate = "$rate is Rs.44.8";  
static final String truth = "Sun rises in the east";  
static final String center = "FlowLayout.LEFT aligns components to left";*

**If string is final why again static final String? Can’t we simply use static String instead of static final string?**

The sting class is designed as final and also strings are immutable. Even though immutable, still you can reassign a new value to string; but at the cost of performance. If you declare sting as final in your code, you cannot reassign a new value for it.

**If string is threadsafe in java why we have stringbuffer class in java?**

String is designed to be immutable. To have mutable string, StringBuffer is introduced. StringBuffer methods are thread-safe as all the methods are synchronized and best suitable in a multithreaded environment. To have unsynchronized methods, StringBuilder was introduced from JDK 1.5.

**The meaning of this. static final String[] numbers?**

Static means it does not require an object to call, final means you cannot reassign another string array to numbers and numbers is a string array.

## Static Blocks Static Initialization

##### Generally, a Java programmer initializes variables in a [constructor](http://way2java.com/oops-concepts/constructors-and-constructor-overloading/) (or init() method in case of [applet](http://way2java.com/applets/life-cycle-of-applet/)). It is the best place chosen, as the constructor is called implicitly when an object is created. Programmer creates objects before anything is done in coding (as object is required to call an instance variable or method). Now read on Static Blocks.

In the place of a constructor, a **static block** can be chosen, if the programmer does not like to have a constructor. One more advantage is static block is executed even before **main()**method is executed. That is, **Java execution starts from static blocks and not from main() method**.

**Three programs are given on static blocks; each differ slightly.**

In the following program, the i**nstance variables are static**. As they are static, they called from main() without the help of an object.

*public class Demo{*

*static double $rate;*

*static int numOfDollars;*

*static{*

*$rate = 44.6;*

*numOfDollars = 12;*

*System.out.println("I am static block, I am called first.");*

*}*

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

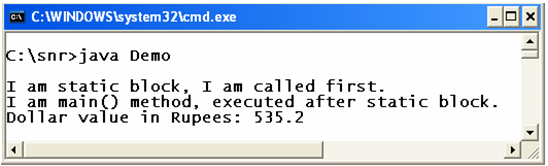
*Demo d1 = new Demo();*

*System.out.println("I am main() method, executed after static block.");*

*System.out.println("Dollar value in Rupees: " + $rate \* numOfDollars);*

*}*

*}*



After executing **static block**, main() is executed. Observe the screenshot.

The previous program can be modified where instance variables are not static. Then you require an object to call them from static main() method.

*public class Demo{*

*double $rate;*

*int numOfDollars;*

*static Demo d1;*

*static{*

*d1 = new Demo();*

*d1.$rate = 44.6;*

*d1.numOfDollars = 12;*

*System.out.println("I am static block, I am called first.");*

*}*

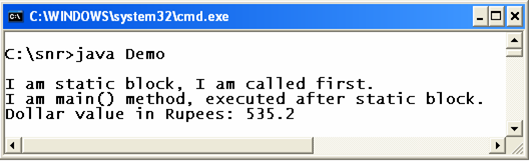
*public static void main(String args[]){*

*System.out.println("I am main() method, executed after static block.");*

*System.out.println("Dollar value in Rupees: " + d1.$rate \* d1.numOfDollars);*

*}*

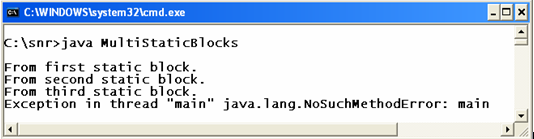
*}*



**Multi Static Blocks**

A program can have any number of static blocks.

|  |  |
| --- | --- |
|  | *public class MultiStaticBlocks{*  *static {*  *System.out.println();*  *System.out.println("From first static block.");*  *}*  *static {*  *System.out.println("From second static block.");*  *}*  *static {*  *System.out.println("From third static block.");*  *}*  *}* |



Observe, purposefully, the [main()](http://way2java.com/oops-concepts/public-static-void-mainstring-args/) method is not included in the program. Even then, the program compiles and runs as static blocks are called before main(). Look at the screenshot, the main() is checked by the JVM after calling all static blocks. As the main() is not available, an exception is thrown.

**JDK 1.7 Modifications**

From **JDK 1.7**, main() method is required to execute static block. Anyhow, static block is called before main() is executed.

**How I found default size of any object?**

Use the getObjectSize(Object obj) of interface java.lang.instrument.Instrumentation.

**Ques.  
*public class StaticClass{***

***String name;  
int rollno;  
static int height = 56;  
static StaticClass s1;***

***static{***

***System.out.println(“Am static block am called first “);***

***s1.name = “naresh”;  
s1.rollno = 16;  
System.out.println(height);***

***}***

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

***System.out.println(“am main() method called after static block”);  
System.out.println(“above defined values are :” +s1.name + “” +s1.rollno);***

***}  
}***

**sir,**

**I have declared everything correctly but it is showing me the error as Exception in initialiserError caused by NullpointerException.Could u please explain why???**

Ans.

*public class StaticClass{  
String name=””;  
int rollno;  
static int height = 56;*

*static StaticClass s1 = new StaticClass();  
static{*

*System.out.println(“Am static block am called first “);*

*s1.name = “naresh”;  
 s1.rollno = 16;  
 System.out.println(height);  
}  
public static void main(String[] args){  
 System.out.println(“am main() method called after static block”);  
 System.out.println(“above defined values are :” +s1.name + ” ” +s1.rollno);  
}  
}  
Do the code as above. You have not instantiated s1.*

**Note:**

Advantage of static block it will execute before main method it means it loads when class is loaded. But from 1.7 VERSION of jdk onward Java people done some changes in JVM it will strictly searching for “public static void main(String[] args)” until and unless you do not provide main method it is not going to execute any code.

**Note:**

Your observation is very correct. From JDK 1.7, main() method is required to execute static block. Anyhow, static block is called before main() is executed.

**Note:**

When the JVM starts executing a program, first it executes static blocks and then main() method. After executing static blocks and did not find main() method, it gives error NoSuchMethodFound

## What is static blank final variable in Java with Example?

We have seen earlier, **blank final variable** and let us see now **static blank final variable**.

**1. What is blank final variable?**

A final variable declared but not initialized (or not given a value) is known as **blank final variable**. It can be given a value through a **constructor** only (but not through a method call).

**2. What is static blank final variable?**

It is simply a blank final variable declared extra as **static**. That is, a final static variable declared but not given a value or not initialized is known as **static blank final variable**.

**3. How to give a value to a static blank final variable?**

It can be given value (or initialized) through a **static block** only.

**4. Give an example on the initialization of static blank final variable?**  
Following example illustrates.

*public class Employee{*

*final static double salary;*

*static*  ***// static block***

*{*

*salary = 9999.99;*

*}*

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

*System.out.println(Employee.salary);*  ***// prints 9999.99***

*}*

*}*

In the above code, "salary = 9999.99;" is commented in the static block, it raises compilation error. It is because a static blank final variable should be initialized somewhere in the program and that too from a static block only.

**5. What is the usage of static blank final variable?**

It allows a blank final variable to get initialized (through static block) at the time of execution**before any method or constructor** called. It works like a constant and increases performance with JVM **optimizations**.

**6. What is the difference between blank final variable and static blank final variable?**  
The difference is just **static** keyword. A simple blank final variable can be initialized through a**constructor** where as static blank final variable through a **static block**only.

**why we can’t assign value to blank final variable through method in Java**

One of the functionalities of a constructor is Developer can use the constructor to assign properties (values to instance variables) to an object at the time of creation itself. If an object is created without assigning values, the object takes default values to variables. Once taken, as it is final variable, a value cannot be given. This is the reason.

###### **Difference Static binding Dynamic binding**

Java, being an**OOPs** language, supports both **static binding** and **dynamic binding**. Coming to our topic, a class may have **overloaded**and **overridden**methods. When to call which method is decided (binded) sometimes by compiler and sometimes by JVM at runtime. Why this disparity? Why Java is designed like this? All the questions will be answered right now.

###### **A) Static binding with Method Overloading**

Instead of writing hundred lines of narration, it is better to explain the concept through a program (like, instead of writing 100 lines about Charminar monument, better show an photo of Charminar; child gets clear idea fast).

*public class Area{*

*public void calculate(){* ***// I***

*System.out.println("Nothing to calculate");*

*}*

*public void calculate(int x){* ***// II***

*System.out.println("Circle Area: " + Math.PI \* Math.pow(x, 2));*

*}*

*public void calculate(double x){* ***// III***

*System.out.println("Circle Perimeter: " + 2 \* Math.PI \* x);*

*}*

*public void calculate(int x, int y){*  ***// IV***

*System.out.println("Rectangle Area: " + x\*y);*

*}*

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

*Area a1 = new Area();*

*a1.calculate();* ***// I***

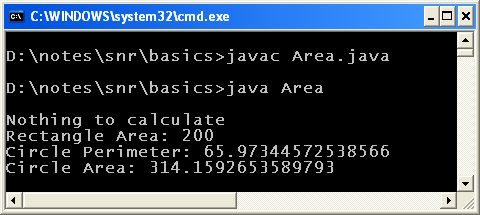
*a1.calculate(10, 20);* ***// IV***

*a1.calculate(10.5);* ***// III***

*a1.calculate(10);*  ***// II***

*}*

*}*



In the above code, **calculate()**method is overloaded (known as [Method Overloading)](http://way2java.com/oops-concepts/using-methods-and-method-overloading/) 4 times. In the main() method all the 4 methods are called. Okay, nice. When **a1.calculate(10.5)** is called, now which overloaded method to be called is decided at**compile time** or **runtime**. Which is better you think, say you are developing a language. Any Designer’s prime importance is **performance**. If decided at compile time, at run time (or execution time), the method is simply called without wasting time in thinking. That is, decided at compile time and called (or executed) at runtime.

Observe, the compiler has got all the information to bind the method at compile time. In**a1.calculate(10.5)**, the parameter is double value. Compiler searches for a double value parameter **calculate()**method and binds it. This increases performance of execution. **Deciding earlier at compile time is known as compile time binidng or static binding.**

**Static binding is achieved through method overloading.**Static binding increases performance.

**Note:** Static binding cannot be done with private, final and static methods.

###### **B) Dynamic binding with Method Overriding**

**Dynamic binding is achieved through method overriding.** Let us write a program with method overriding.

*class Lecturer{*

*public void call(){* ***// I***

*System.out.println("Hello Sir, Good morning");*

*}*

*}*

*class Raju extends Lecturer{*

*public void call(){ //* ***II***

*System.out.println("Hello Raju, improve your programming skills");*

*}*

*}*

*class Prasad extends Lecturer{*

*public void call(){ //* ***III***

*System.out.println("Hello Prasad, improve your English language");*

*}*

*}*

*class Jyostna extends Lecturer{*

*public void call(){ //* ***IV***

*System.out.println("Hello Jyostna, improve your aptitude and reasoning skills");*

*}*

*}*

*public class DynamicBindingExample{*

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

*Lecturer l1 = new Lecturer();*

*l1.call();* ***// I***

*Raju r1 = new Raju();*

*l1 = r1;*

*l1.call();* ***// II***

*Prasad p1 = new Prasad();*

*l1 = p1;*

*l1.call();* ***// III***

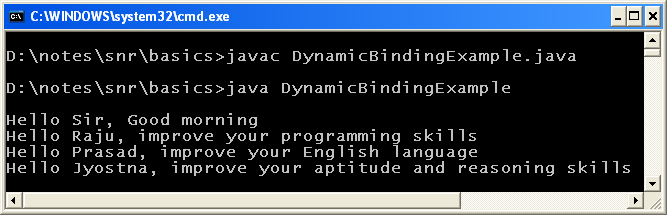
*Jyostna j1 = new Jyostna();*

*l1 = j1;*

*l1.call();* ***// IV***

*}*

*}*



There are three subclasses (**Hierarchical inheritance**), **Raju**, **Prasad** and **Jyostna** for **Lecturer**class. All the three has overridden the **call()**method.

*l1 = r1;  
l1.call();*

In the **main()** method, the super class object **l1** is assigned with subclass object Raju **r1**. Now**l1** will call which call() method, its own or subclass Raju’s; the doubt came because in **l1**, the reference of subclass object Raju exists. Definitely you say**l1** calls **r1**call() method.

In the next statement the reference **r1** is replaced by **p1**, the subclass object of **Prasad**. Now l1 will call **Prasad** call() method. Similarly, in the immediate next statement,**p1** reference is replaced by **j1**, the **Jyostna**object. Now **l1**will call Jyostna’s call() method.

It is okay. The question is, replacement of subclass objects **r1**, **p1** and **j1**in the super class object happens at **compile time** or **runtime**. This replacement will be done at runtime as the execution proceeds forwards statement by statement. When replaced, **l1** calls first **Raju’s call()**method and later that of **Prasad** and**Jyostna**. **Which method is to be called is decided at runtime and is known as runtime binding or dynamic binding.**  
Dynamic binding is achieved through method overriding. If method overriding does not exist, the above program does not work to call subclass methods.

#### Restriction on Static Nested Classes

[static keyword](http://way2java.com/oops-concepts/static-keyword-%e2%80%93-philosophy/) can be applied on classes and interfaces only when they are nested. Top-level classes and interfaces cannot be static.

This is the second variation of the total six discussed in this series.

**Note:**It is advised to read the [basics and types of inner classes](http://way2java.com/java-lang/inner-classes/) before proceeding further.

###### **Let go on this combination – Static Nested Classes**

*public class Outer1{*

*int x = 10;*

*static int y = 20;*

*static class Inner1{*

*int k = 30;*

*static int m = 40;*

*public void display(){*

*// System.out.println(x); // raises compilation error*

*System.out.println(y);*

*System.out.println(k);*

*System.out.println(m);*

*}*

*}*

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

*Outer1.Inner1 i1 = new Outer1.Inner1();*

***// 1st style of creating static inner class object***

*i1.display();*

*Inner1 i2 = new Outer1.Inner1();*

***// 2nd style of creating static inner class object***

*i2.display();*

*}*

*}*

The slight modification in this program than the previous [Java Nested Classes](http://way2java.com/java-lang/inner-classes-java-lang/java-nested-classes/) is here the inner class is declared **static**. The inner class can have static and non-static members like the outer class. But the restriction is "**the static inner class cannot access outer class non-static members**". For this reason, printing of **x** variable in display() method is commented out. It can directly call outer class static member,**y**.

One more difference we can observe is creating object of static inner class. We know in OOPs, a static member call be called with a class name also.

**Ealier:** Non-static inner class object creation: Outer1.Inner1 i2 = new Outer1().new Inner1();

**Now with static inner class:** Outer1.Inner1 i1 = new Outer1.Inner1();

In the above statement,**Outer1** constructor is not necessary. Just class name is enough.

Because inner classes usage is growing slowly, they are discussed more elaborately here. **Android** (Java based Mobile OS developed by Google) uses inner classes very extensively, infact [anonymous inner classes](http://way2java.com/java-lang/inner-classes/) are passed as parameters to methods.

## Class Variables – Static Fields

Class variables also known as static fields share characteristics across all Objects within a Class. When you declare a field to be static, only a single instance of the associated variable is created, which is common to all the Objects of that Class. Hence when one Object changes the value of a Class variable, it affects all the Objects of the Class. We can access a Class variable by using the name of the Class, and not necessarily using a reference to an individual Object within the Class. Static variables can be accessed even when no Objects of that Class exists. Class variables are  declared using the static keyword.

## Class Methods – Static Methods

Class Methods, similar to Class variables can be invoked without having an instance of the Class. Class methods are often used to provide global functions for Java programs. For example, Methods in the java.lang.Math package are Class methods. You cannot call non-static Methods from inside a static Method.

## Static Keyword Rules

* ***Variable or Methods*** marked static belong to the **Class** rather then to any particular Instance.
* **Static Method or variable** can be used without creating or referencing any instance of the Class.
* If there are instances, a static variable of a Class will be shared by all instances of that class, This will result in **only one copy**.
* A static Method can’t access a non static variable nor can directly invoke non static Method (It can invoke or access Method or variable via instances).

## Cheat-sheet

* **Static**is a Non Access Modifier.
* **The Static** modifier can be applied to a variable or Method or block or inner Class.
* **Static members** belong to Class only not an instance.
* A Static Method **can not** access an instance variable.
* Static Methods **can not** be overriden as they are Class specific and don’t belong to an Instance.
* Static Methods can be ***redefined***.
* If a Class contains any static blocks then that block will be executed only when the Class is loaded in JVM. Creating multiple instances does not execute the static block multiple time. Only the constructor will be executed multiple time.
* If Class.forName(“class\_name“) is called then the static block of the Class will get executed.