this keyword in java

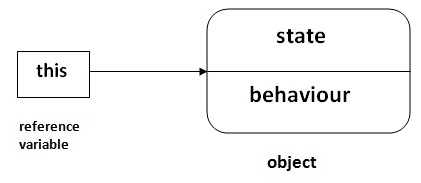
There can be a lot of usage of **java this keyword**. In java, this is a **reference variable** that refers to the current object.

Usage of java this keyword

Here is given the 6 usage of java this keyword.

1. this can be used to refer current class instance variable.
2. this can be used to invoke current class method (implicitly)
3. this() can be used to invoke current class constructor.
4. this can be passed as an argument in the method call.
5. this can be passed as argument in the constructor call.
6. this can be used to return the current class instance from the method.

**Suggestion:** If you are beginner to java, lookup only three usage of this keyword.



1) this: to refer current class instance variable

The this keyword can be used to refer current class instance variable. If there is ambiguity between the instance variables and parameters, this keyword resolves the problem of ambiguity.

Understanding the problem without this keyword

|  |
| --- |
| Let's understand the problem if we don't use this keyword by the example given below: |

1. **class** Student{
2. **int** rollno;
3. String name;
4. **float** fee;
5. Student(**int** rollno,String name,**float** fee){
6. rollno=rollno;
7. name=name;
8. fee=fee;
9. }
10. **void** display(){System.out.println(rollno+" "+name+" "+fee);}
11. }
12. **class** TestThis1{
13. **public** **static** **void** main(String args[]){
14. Student s1=**new** Student(111,"ankit",5000f);
15. Student s2=**new** Student(112,"sumit",6000f);
16. s1.display();
17. s2.display();
18. }
19. }

Output:

0 null 0.0

0 null 0.0

In the above example, parameters (formal arguments) and instance variables are same. So, we are using this keyword to distinguish local variable and instance variable.

Solution of the above problem by this keyword

1. **class** Student{
2. **int** rollno;
3. String name;
4. **float** fee;
5. Student(**int** rollno,String name,**float** fee){
6. **this**.rollno=rollno;
7. **this**.name=name;
8. **this**.fee=fee;
9. }
10. **void** display(){System.out.println(rollno+" "+name+" "+fee);}
11. }
13. **class** TestThis2{
14. **public** **static** **void** main(String args[]){
15. Student s1=**new** Student(111,"ankit",5000f);
16. Student s2=**new** Student(112,"sumit",6000f);
17. s1.display();
18. s2.display();
19. }}

Output:

111 ankit 5000

112 sumit 6000

If local variables(formal arguments) and instance variables are different, there is no need to use this keyword like in the following program:

Program where this keyword is not required

1. **class** Student{
2. **int** rollno;
3. String name;
4. **float** fee;
5. Student(**int** r,String n,**float** f){
6. rollno=r;
7. name=n;
8. fee=f;
9. }
10. **void** display(){System.out.println(rollno+" "+name+" "+fee);}
11. }
13. **class** TestThis3{
14. **public** **static** **void** main(String args[]){
15. Student s1=**new** Student(111,"ankit",5000f);
16. Student s2=**new** Student(112,"sumit",6000f);
17. s1.display();
18. s2.display();
19. }}

Output:

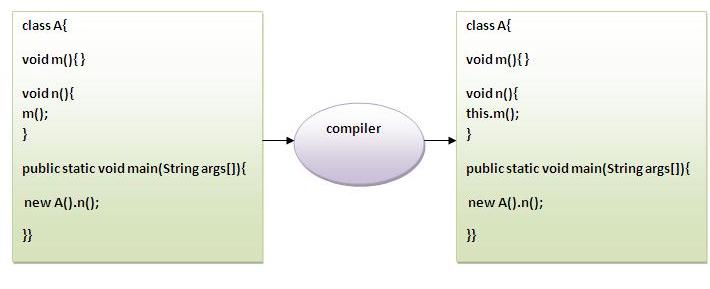
111 ankit 5000

112 sumit 6000

**It is better approach to use meaningful names for variables. So we use same name for instance variables and parameters in real time, and always use this keyword.**

2) this: to invoke current class method

You may invoke the method of the current class by using the this keyword. If you don't use the this keyword, compiler automatically adds this keyword while invoking the method. Let's see the example



1. **class** A{
2. **void** m(){System.out.println("hello m");}
3. **void** n(){
4. System.out.println("hello n");
5. //m();//same as this.m()
6. **this**.m();
7. }
8. }
9. **class** TestThis4{
10. **public** **static** **void** main(String args[]){
11. A a=**new** A();
12. a.n();
13. }}

Output:

hello n

hello m

3) this() : to invoke current class constructor

The this() constructor call can be used to invoke the current class constructor. It is used to reuse the constructor. In other words, it is used for constructor chaining.

**Calling default constructor from parameterized constructor:**

1. **class** A{
2. A(){System.out.println("hello a");}
3. A(**int** x){
4. **this**();
5. System.out.println(x);
6. }
7. }
8. **class** TestThis5{
9. **public** **static** **void** main(String args[]){
10. A a=**new** A(10);
11. }}

Output:

hello a

10

**Calling parameterized constructor from default constructor:**

1. **class** A{
2. A(){
3. **this**(5);
4. System.out.println("hello a");
5. }
6. A(**int** x){
7. System.out.println(x);
8. }
9. }
10. **class** TestThis6{
11. **public** **static** **void** main(String args[]){
12. A a=**new** A();
13. }}

Output:

5

hello a

Real usage of this() constructor call

The this() constructor call should be used to reuse the constructor from the constructor. It maintains the chain between the constructors i.e. it is used for constructor chaining. Let's see the example given below that displays the actual use of this keyword.

1. **class** Student{
2. **int** rollno;
3. String name,course;
4. **float** fee;
5. Student(**int** rollno,String name,String course){
6. **this**.rollno=rollno;
7. **this**.name=name;
8. **this**.course=course;
9. }
10. Student(**int** rollno,String name,String course,**float** fee){
11. **this**(rollno,name,course);//reusing constructor
12. **this**.fee=fee;
13. }
14. **void** display(){System.out.println(rollno+" "+name+" "+course+" "+fee);}
15. }
16. **class** TestThis7{
17. **public** **static** **void** main(String args[]){
18. Student s1=**new** Student(111,"ankit","java");
19. Student s2=**new** Student(112,"sumit","java",6000f);
20. s1.display();
21. s2.display();
22. }}

Output:

111 ankit java null

112 sumit java 6000

**Rule: Call to this() must be the first statement in constructor.**

1. **class** Student{
2. **int** rollno;
3. String name,course;
4. **float** fee;
5. Student(**int** rollno,String name,String course){
6. **this**.rollno=rollno;
7. **this**.name=name;
8. **this**.course=course;
9. }
10. Student(**int** rollno,String name,String course,**float** fee){
11. **this**.fee=fee;
12. **this**(rollno,name,course);//C.T.Error
13. }
14. **void** display(){System.out.println(rollno+" "+name+" "+course+" "+fee);}
15. }
16. **class** TestThis8{
17. **public** **static** **void** main(String args[]){
18. Student s1=**new** Student(111,"ankit","java");
19. Student s2=**new** Student(112,"sumit","java",6000f);
20. s1.display();
21. s2.display();
22. }}

Compile Time Error: Call to this must be first statement in constructor

4) this: to pass as an argument in the method

The this keyword can also be passed as an argument in the method. It is mainly used in the event handling. Let's see the example:

1. **class** S2{
2. **void** m(S2 obj){
3. System.out.println("method is invoked");
4. }
5. **void** p(){
6. m(**this**);
7. }
8. **public** **static** **void** main(String args[]){
9. S2 s1 = **new** S2();
10. s1.p();
11. }
12. }

Output:

method is invoked

Application of this that can be passed as an argument:

In event handling (or) in a situation where we have to provide reference of a class to another one. It is used to reuse one object in many methods.

5) this: to pass as argument in the constructor call

We can pass the this keyword in the constructor also. It is useful if we have to use one object in multiple classes. Let's see the example:

1. **class** B{
2. A4 obj;
3. B(A4 obj){
4. **this**.obj=obj;
5. }
6. **void** display(){
7. System.out.println(obj.data);//using data member of A4 class
8. }
9. }
11. **class** A4{
12. **int** data=10;
13. A4(){
14. B b=**new** B(**this**);
15. b.display();
16. }
17. **public** **static** **void** main(String args[]){
18. A4 a=**new** A4();
19. }
20. }

Output:10

6) this keyword can be used to return current class instance

We can return this keyword as an statement from the method. In such case, return type of the method must be the class type (non-primitive). Let's see the example:

Syntax of this that can be returned as a statement

1. return\_type method\_name(){
2. **return** **this**;
3. }

Example of this keyword that you return as a statement from the method

1. **class** A{
2. A getA(){
3. **return** **this**;
4. }
5. **void** msg(){System.out.println("Hello java");}
6. }
7. **class** Test1{
8. **public** **static** **void** main(String args[]){
9. **new** A().getA().msg();
10. }
11. }

Output:

Hello java

Proving this keyword

|  |
| --- |
| Let's prove that this keyword refers to the current class instance variable. In this program, we are printing the reference variable and this, output of both variables are same. |

1. **class** A5{
2. **void** m(){
3. System.out.println(**this**);//prints same reference ID
4. }
5. **public** **static** **void** main(String args[]){
6. A5 obj=**new** A5();
7. System.out.println(obj);//prints the reference ID
8. obj.m();
9. }
10. }

Output:

A5@22b3ea59

A5@22b3ea59

super keyword in java

The **super** keyword in java is a reference variable which is used to refer immediate parent class object.

Whenever you create the instance of subclass, an instance of parent class is created implicitly which is referred by super reference variable.

Usage of java super Keyword

1. super can be used to refer immediate parent class instance variable.
2. super can be used to invoke immediate parent class method.
3. super() can be used to invoke immediate parent class constructor.

1) super is used to refer immediate parent class instance variable.

We can use super keyword to access the data member or field of parent class. It is used if parent class and child class have same fields.

1. **class** Animal{
2. String color="white";
3. }
4. **class** Dog **extends** Animal{
5. String color="black";
6. **void** printColor(){
7. System.out.println(color);//prints color of Dog class
8. System.out.println(**super**.color);//prints color of Animal class
9. }
10. }
11. **class** TestSuper1{
12. **public** **static** **void** main(String args[]){
13. Dog d=**new** Dog();
14. d.printColor();
15. }}

Output:

black

white

In the above example, Animal and Dog both classes have a common property color. If we print color property, it will print the color of current class by default. To access the parent property, we need to use super keyword.

2) super can be used to invoke parent class method

The super keyword can also be used to invoke parent class method. It should be used if subclass contains the same method as parent class. In other words, it is used if method is overridden.

1. **class** Animal{
2. **void** eat(){System.out.println("eating...");}
3. }
4. **class** Dog **extends** Animal{
5. **void** eat(){System.out.println("eating bread...");}
6. **void** bark(){System.out.println("barking...");}
7. **void** work(){
8. **super**.eat();
9. bark();
10. }
11. }
12. **class** TestSuper2{
13. **public** **static** **void** main(String args[]){
14. Dog d=**new** Dog();
15. d.work();
16. }}

Output:

eating...

barking...

In the above example Animal and Dog both classes have eat() method if we call eat() method from Dog class, it will call the eat() method of Dog class by default because priority is given to local.

To call the parent class method, we need to use super keyword.

3) super is used to invoke parent class constructor.

The super keyword can also be used to invoke the parent class constructor. Let's see a simple example:

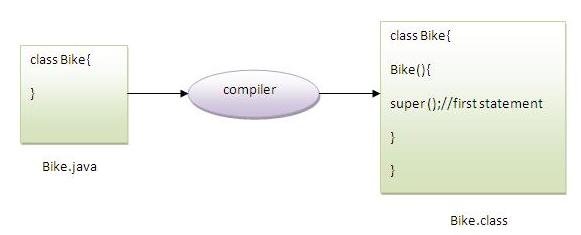
1. **class** Animal{
2. Animal(){System.out.println("animal is created");}
3. }
4. **class** Dog **extends** Animal{
5. Dog(){
6. **super**();
7. System.out.println("dog is created");
8. }
9. }
10. **class** TestSuper3{
11. **public** **static** **void** main(String args[]){
12. Dog d=**new** Dog();
13. }}

Output:

animal is created

dog is created

**Note: super() is added in each class constructor automatically by compiler if there is no super() or this().**



As we know well that default constructor is provided by compiler automatically if there is no constructor. But, it also adds super() as the first statement.

**Another example of super keyword where super() is provided by the compiler implicitly.**

1. **class** Animal{
2. Animal(){System.out.println("animal is created");}
3. }
4. **class** Dog **extends** Animal{
5. Dog(){
6. System.out.println("dog is created");
7. }
8. }
9. **class** TestSuper4{
10. **public** **static** **void** main(String args[]){
11. Dog d=**new** Dog();
12. }}

Output:

animal is created

dog is created

super example: real use

Let's see the real use of super keyword. Here, Emp class inherits Person class so all the properties of Person will be inherited to Emp by default. To initialize all the property, we are using parent class constructor from child class. In such way, we are reusing the parent class constructor.

1. **class** Person{
2. **int** id;
3. String name;
4. Person(**int** id,String name){
5. **this**.id=id;
6. **this**.name=name;
7. }
8. }
9. **class** Emp **extends** Person{
10. **float** salary;
11. Emp(**int** id,String name,**float** salary){
12. **super**(id,name);//reusing parent constructor
13. **this**.salary=salary;
14. }
15. **void** display(){System.out.println(id+" "+name+" "+salary);}
16. }
17. **class** TestSuper5{
18. **public** **static** **void** main(String[] args){
19. Emp e1=**new** Emp(1,"ankit",45000f);
20. e1.display();
21. }}

Output:

1 ankit 45000

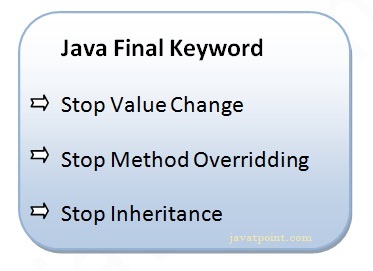
Final Keyword In Java

1. [Final variable](https://www.javatpoint.com/final-keyword#finalv)
2. [Final method](https://www.javatpoint.com/final-keyword#finalm)
3. [Final class](https://www.javatpoint.com/final-keyword#finalc)
4. [Is final method inherited ?](https://www.javatpoint.com/final-keyword#finalisinherited)
5. [Blank final variable](https://www.javatpoint.com/final-keyword#finalblank)
6. [Static blank final variable](https://www.javatpoint.com/final-keyword#finalstaticblank)
7. [Final parameter](https://www.javatpoint.com/final-keyword#finalpara)
8. [Can you declare a final constructor](https://www.javatpoint.com/final-keyword#finalcons)

The **final keyword** in java is used to restrict the user. The java final keyword can be used in many context. Final can be:

1. variable
2. method
3. class

The final keyword can be applied with the variables, a final variable that have no value it is called blank final variable or uninitialized final variable. It can be initialized in the constructor only. The blank final variable can be static also which will be initialized in the static block only. We will have detailed learning of these. Let's first learn the basics of final keyword.



1) Java final variable

If you make any variable as final, you cannot change the value of final variable(It will be constant).

Example of final variable

There is a final variable speedlimit, we are going to change the value of this variable, but It can't be changed because final variable once assigned a value can never be changed.

1. **class** Bike9{
2. **final** **int** speedlimit=90;//final variable
3. **void** run(){
4. speedlimit=400;
5. }
6. **public** **static** **void** main(String args[]){
7. Bike9 obj=**new**  Bike9();
8. obj.run();
9. }
10. }//end of class

Output:Compile Time Error

2) Java final method

If you make any method as final, you cannot override it.

Example of final method

1. **class** Bike{
2. **final** **void** run(){System.out.println("running");}
3. }
5. **class** Honda **extends** Bike{
6. **void** run(){System.out.println("running safely with 100kmph");}
8. **public** **static** **void** main(String args[]){
9. Honda honda= **new** Honda();
10. honda.run();
11. }
12. }

Output:Compile Time Error

3) Java final class

If you make any class as final, you cannot extend it.

Example of final class

1. **final** **class** Bike{}
3. **class** Honda1 **extends** Bike{
4. **void** run(){System.out.println("running safely with 100kmph");}
6. **public** **static** **void** main(String args[]){
7. Honda1 honda= **new** Honda();
8. honda.run();
9. }
10. }

Output:Compile Time Error

Q) Is final method inherited?

Ans) Yes, final method is inherited but you cannot override it. For Example:

1. **class** Bike{
2. **final** **void** run(){System.out.println("running...");}
3. }
4. **class** Honda2 **extends** Bike{
5. **public** **static** **void** main(String args[]){
6. **new** Honda2().run();
7. }
8. }

Output:running...

Q) What is blank or uninitialized final variable?

A final variable that is not initialized at the time of declaration is known as blank final variable.

If you want to create a variable that is initialized at the time of creating object and once initialized may not be changed, it is useful. For example PAN CARD number of an employee.

It can be initialized only in constructor.

Example of blank final variable

1. **class** Student{
2. **int** id;
3. String name;
4. **final** String PAN\_CARD\_NUMBER;
5. ...
6. }

Que) Can we initialize blank final variable?

Yes, but only in constructor. For example:

1. **class** Bike10{
2. **final** **int** speedlimit;//blank final variable
4. Bike10(){
5. speedlimit=70;
6. System.out.println(speedlimit);
7. }
9. **public** **static** **void** main(String args[]){
10. **new** Bike10();
11. }
12. }

Output:70

static blank final variable

A static final variable that is not initialized at the time of declaration is known as static blank final variable. It can be initialized only in static block.

Example of static blank final variable

1. **class** A{
2. **static** **final** **int** data;//static blank final variable
3. **static**{ data=50;}
4. **public** **static** **void** main(String args[]){
5. System.out.println(A.data);
6. }
7. }

Q) What is final parameter?

If you declare any parameter as final, you cannot change the value of it.

1. **class** Bike11{
2. **int** cube(**final** **int** n){
3. n=n+2;//can't be changed as n is final
4. n\*n\*n;
5. }
6. **public** **static** **void** main(String args[]){
7. Bike11 b=**new** Bike11();
8. b.cube(5);
9. }
10. }

Output:Compile Time Error

Q) Can we declare a constructor final?

No, because constructor is never inherited.

Java static keyword

1. [Static variable](https://www.javatpoint.com/static-keyword-in-java#staticv)
2. [Program of counter without static variable](https://www.javatpoint.com/static-keyword-in-java#staticvcounter1)
3. [Program of counter with static variable](https://www.javatpoint.com/static-keyword-in-java#staticvcounter2)
4. [Static method](https://www.javatpoint.com/static-keyword-in-java#staticm)
5. [Restrictions for static method](https://www.javatpoint.com/static-keyword-in-java#staticmr)
6. [Why main method is static ?](https://www.javatpoint.com/static-keyword-in-java#staticwhymain)
7. [Static block](https://www.javatpoint.com/static-keyword-in-java#staticblock)
8. [Can we execute a program without main method ?](https://www.javatpoint.com/static-keyword-in-java#staticwithoutmain)

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

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) e.g. 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).

Understanding problem without static variable

1. **class** Student{
2. **int** rollno;
3. String name;
4. String college="ITS";
5. }

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.

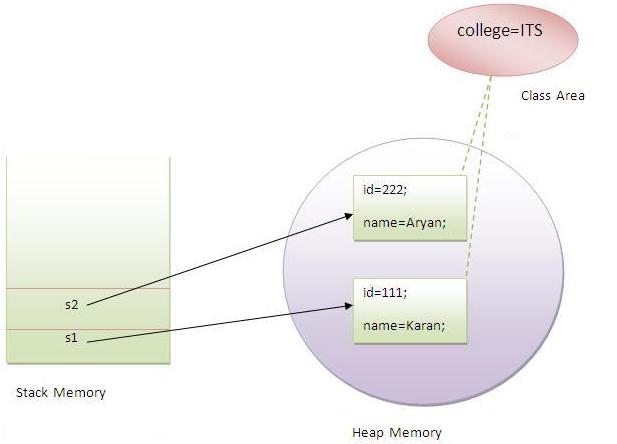
**Java static property is shared to all objects.**

Example of static variable

1. //Program of static variable
3. **class** Student8{
4. **int** rollno;
5. String name;
6. **static** String college ="ITS";
8. Student8(**int** r,String n){
9. rollno = r;
10. name = n;
11. }
12. **void** display (){System.out.println(rollno+" "+name+" "+college);}
14. **public** **static** **void** main(String args[]){
15. Student8 s1 = **new** Student8(111,"Karan");
16. Student8 s2 = **new** Student8(222,"Aryan");
18. s1.display();
19. s2.display();
20. }
21. }

Output:111 Karan ITS

222 Aryan ITS



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.

1. **class** Counter{
2. **int** count=0;//will get memory when instance is created
4. Counter(){
5. count++;
6. System.out.println(count);
7. }
9. **public** **static** **void** main(String args[]){
11. Counter c1=**new** Counter();
12. Counter c2=**new** Counter();
13. Counter c3=**new** Counter();
15. }
16. }

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. |

1. **class** Counter2{
2. **static** **int** count=0;//will get memory only once and retain its value
4. Counter2(){
5. count++;
6. System.out.println(count);
7. }
9. **public** **static** **void** main(String args[]){
11. Counter2 c1=**new** Counter2();
12. Counter2 c2=**new** Counter2();
13. Counter2 c3=**new** Counter2();
15. }
16. }

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

1. //Program of changing the common property of all objects(static field).
3. **class** Student9{
4. **int** rollno;
5. String name;
6. **static** String college = "ITS";
8. **static** **void** change(){
9. college = "BBDIT";
10. }
12. Student9(**int** r, String n){
13. rollno = r;
14. name = n;
15. }
17. **void** display (){System.out.println(rollno+" "+name+" "+college);}
19. **public** **static** **void** main(String args[]){
20. Student9.change();
22. Student9 s1 = **new** Student9 (111,"Karan");
23. Student9 s2 = **new** Student9 (222,"Aryan");
24. Student9 s3 = **new** Student9 (333,"Sonoo");
26. s1.display();
27. s2.display();
28. s3.display();
29. }
30. }

Output:111 Karan BBDIT

222 Aryan BBDIT

333 Sonoo BBDIT

Another example of static method that performs normal calculation

1. //Program to get cube of a given number by static method
3. **class** Calculate{
4. **static** **int** cube(**int** x){
5. **return** x\*x\*x;
6. }
8. **public** **static** **void** main(String args[]){
9. **int** result=Calculate.cube(5);
10. System.out.println(result);
11. }
12. }

Output:125

Restrictions for static method

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

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

1. **class** A{
2. **int** a=40;//non static
4. **public** **static** **void** main(String args[]){
5. System.out.println(a);
6. }
7. }

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

1. **class** A2{
2. **static**{System.out.println("static block is invoked");}
3. **public** **static** **void** main(String args[]){
4. System.out.println("Hello main");
5. }
6. }

Output:static block is invoked

Hello main

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

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

1. **class** A3{
2. **static**{
3. System.out.println("static block is invoked");
4. System.exit(0);
5. }
6. }

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)