Java static keyword

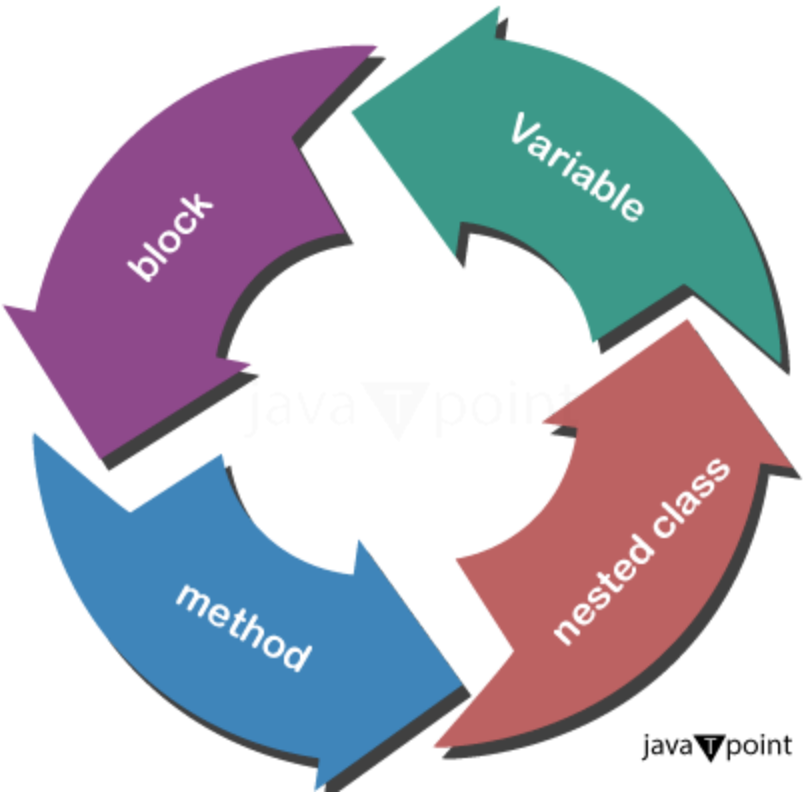
The static keyword in Java is used for memory management mainly. We can apply static keyword with variables, methods, blocks and nested classes. The static keyword belongs to the class than an instance of the class.

The static can be:

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

Beyond memory management, the static keyword in Java has several other uses. When it comes to variables, it means that the variable is shared by all instances of the class and belongs to the class as a whole, not just any one instance. Static methods are available throughout the programme since they can be called without first generating an instance of the class. When the class loads, static blocks are used to initialise static variables or carry out one-time operations. Furthermore, nested static classes can be instantiated individually but are still linked to the outer class. Moreover, class names can be used to access static variables and methods directly, eliminating the need to build an object instance. This is especially helpful for constants or utility methods.

1) Java static variable



If we declare any variable as static, it is known as a static variable.

- The static variable can be used to refer to the common property of all objects (which is not unique for each object), for example, the company name of employees, college name of students, etc.
- The static variable gets memory only once in the class area at the time of class loading.
- Static variables in Java are also initialized to default values if not explicitly initialized by the programmer. They can be accessed directly using the class name without needing to create an instance of the class.
- Static variables are shared among all instances of the class, meaning if the value of a static variable is changed in one instance, it will reflect the change in all other instances as well.

Advantages of Static Variable

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

Understanding The 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 the object is created. All students have their unique rollno and name, so instance data member is good in such case. Here, "college" refers to the common property of all objects. If we make it static, this field will get the memory only once.

Example of Static Variable

Filename: TestStaticVariable1.java

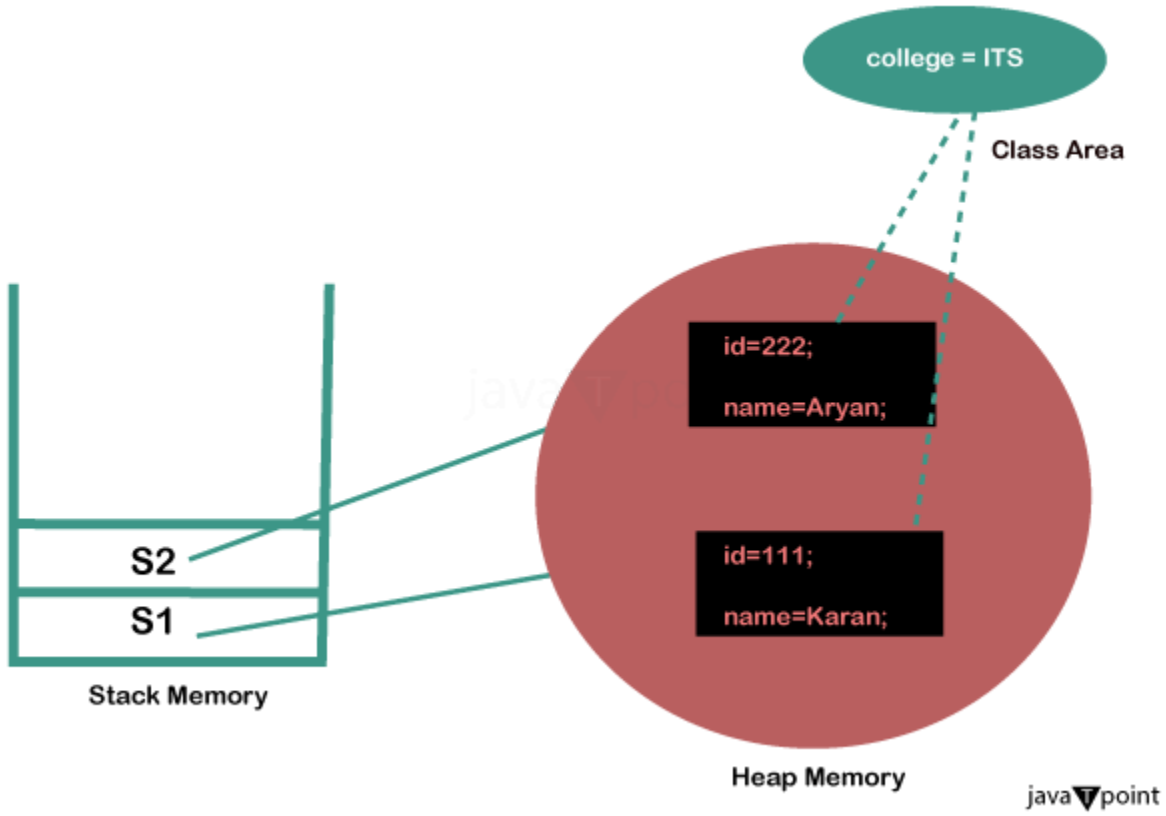
```
//Java Program to demonstrate the use of static variable
class Student{
  int rollno;//instance variable
  String name;
  static String college ="ITS";//static variable
  //constructor
  Student(int r, String n){
  rollno = r;
  name = n;
  }
  //method to display the values
  void display (){System.out.println(rollno+" "+name+" "+college);}
}
//Test class to show the values of objects
public class TestStaticVariable1{
 public static void main(String args[]){
 Student s1 = new Student(111,"Karan");
 Student s2 = new Student(222,"Aryan");
 //we can change the college of all objects by the single line of code
 //Student.college="BBDIT";
 s1.display();
 s2.display();
 }
}
```

Output:

111 Karan ITS
222 Aryan ITS

Explanation

The usage of static variables is demonstrated in this Java programme. In addition to a static variable called college, the Student class defines two instance variables: rollno and name. It has a constructor to set the instance variables' initial values and a display() function to output the rollno, name, and college values. Two Student objects, s1 and s2, with different roll numbers and names, are created via the TestStaticVariable1 class. After that, each object's display() method is called, printing its details. The college name can be modified for all objects at once by uncommenting the line Student.college="BBDIT";, demonstrating how static variables are shared by all instances of a class.



Program of The Counter Without Static Variable

In this example, we have created an instance variable named count which is incremented in the constructor. Since the instance variable gets the memory at the time of object creation, each object will have a copy of the instance variable. If

it is incremented, it won't reflect other objects. So each object will have the value 1 in the count variable.

Filename: Counter.java

```
//Java Program to demonstrate the use of an instance variable
//which get memory each time when we create an object of the class.
class Counter{
int count=0;//will get memory each time when the instance is created

Counter(){
count++;//incrementing value
System.out.println(count);
}

public static void main(String args[]){
//Creating objects
Counter c1=new Counter();
Counter c2=new Counter();
Counter c3=new Counter();
}
}
```

Output:

```
1
1
1
```

Explanation

The idea of instance variables that are specific to each object derived from a class is demonstrated by this Java programme. The count variable in the Counter class is an instance variable that is allocated memory each time an object of the class is instantiated. It is initialized to zero. Every time an object is produced, the constructor prints the current value of the count variable and increases it. Three Counter objects (c1, c2, and c3) with separate instances of the count variable are generated in the main procedure. Because of this, the count is increased independently for each object that is generated, and the output shows the count value for each object that increases successively.

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.

File name: Counter2.java

```
//Java Program to illustrate the use of static variable which
//is shared with all objects.
class Counter2{
static int count=0;//will get memory only once and retain its value

Counter2(){
count++;//incrementing the value of static variable
System.out.println(count);
}

public static void main(String args[]){
//creating objects
Counter2 c1=new Counter2();
Counter2 c2=new Counter2();
Counter2 c3=new Counter2();
}
}
```

Output:

```
1
2
3
```

Explanation

This Java program demonstrates the utilization of a static variable, which is shared among all objects created from the class. In the Counter2 class, the count variable is declared as static, ensuring it is allocated memory only once and retains its value across all instances of the class. Each time an object of the class is created, the constructor increments the value of the static variable count and prints its current value. In the main method, three Counter2 objects (c1, c2, and c3) are instantiated, and as they share the same static variable count, its value increments sequentially across the objects, reflecting the shared nature of static variables among all instances of a class.

2) Java Static Method

If we apply a static keyword with any method, it is known as a static method.

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

Filename: TestStaticMethod.java

```
//Java Program to demonstrate the use of a static method.
class Student{
    int rollno;
    String name;
    static String college = "ITS";
    //static method to change the value of static variable
    static void change(){
    college = "BBDIT";
    }
    //constructor to initialize the variable
    Student(int r, String n){
    rollno = r;
    name = n;
    }
    //method to display values
    void display(){System.out.println(rollno+" "+name+" "+college);}
}
//Test class to create and display the values of object
public class TestStaticMethod{
   public static void main(String args[]){
   Student.change();//calling change method
   //creating objects
   Student s1 = new Student(111,"Karan");
   Student s2 = new Student(222,"Aryan");
   Student s3 = new Student(333,"Sonoo");
   //calling display method
   s1.display();
```

```
    s2.display();
    s3.display();
    }
}
```

Output:

111 Karan BBDIT
222 Aryan BBDIT
333 Sonoo BBDIT

Explanation

This Java programme demonstrates how to use a static method inside of a class. In addition to a static variable called college, the Student class defines two instance variables: rollno and name. To alter the value of the static variable college, it has a static function called change(). In addition, the class has a constructor for initialising instance variables and a display() method for printing the rollno, name, and college values. The main method of the TestStaticMethod class shows how to use the change() method to modify the college value. After then, the display() method is used to create three Student objects (s1, s2, and s3) with distinct roll numbers and names, and their details are shown. This illustrates how the static method change() influences the static variable college for all instances of the class.

Let's see another example of a static method that performs a normal calculation.

Filename: Calucalte.java

```
//Java Program to get the cube of a given number using the 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

Explanation

This Java programme shows how to find the cube of a given number by using a static function. The static method cube(int x) in the Calculate class multiplies an integer parameter x three times to return its cube. The same class's main method demonstrates how to use the class name Calculate to directly access the static method cube() without first having to create an instance of the class. The cube() method is used in this example with argument 5, and the outcome is saved in the variable result before being printed to the console. This programme demonstrates how easy and effective it is to use static methods for routine computations or actions that don't call for object creation.

Restrictions for the Static Method

There are the following two main restrictions for the static method.

1. The static method cannot use non-static data members or call a non-static method directly.
2. this and super keyword cannot be used in static context.

Filename: A.java

```
class A{
 int a=40;//non static

 public static void main(String args[]){
  System.out.println(a);
 }
}
```

Output:

Compile Time Error

Explanation

The Java code provided contains a class called A. A defined instance variable with the value 40 that is designated as non-static is present in class A. Nonetheless, an

attempt is made to use System.out.println(a); to directly access the instance variable an in the class A main method. Because static methods in Java cannot directly access non-static variables, this will lead to a compilation problem. An instance of class A must first be generated in order to access the non-static variable an inside the static main method. The variable a can then be accessed using that instance. It would therefore be OK to use A obj = new A(); System.out.println(obj.a); to access an in the main method.

Q) Why is the Java main() method static?

Ans) In Java, main() method is static because the object is not required to call a static method. If it were a non-static method, the JVM would create an object first and then call the main() method, which would lead to the problem of extra memory allocation.

3) Java Static Block

- It is used to initialize the static data member.
- It is executed before the main() method at the time of class loading.

Example of Static block

Filename: A2.java

```
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

Explanation

In the given Java code, there's a class named A2. Inside this class, there's a static block, which is a block of code enclosed within curly braces and marked with the static keyword. Static blocks are executed only once when the class is loaded into memory, before the execution of the main method or the creation of any object

of the class. In this case, the static block contains the statement System.out.println("static block is invoked");, that prints the message "static block is invoked" to the console when the class A2 is loaded into memory.

Additionally, there's a main method in the class A2 that prints "Hello main" to the console when executed. However, since the main method is the entry point of the program, it needs to be invoked explicitly, typically by the Java Virtual Machine (JVM) when executing the program. Therefore, when you run the program, the static block is executed first, printing the message from the static block, followed by the execution of the main method, printing "Hello main" to the console.

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

Ans) No, one of the ways was the static block, but it was possible till JDK 1.6. Since JDK 1.7, it is not possible to execute a Java class without the main() method.

Filename: A3.java

```
class A3{
  static{
  System.out.println("static block is invoked");
  System.exit(0);
  }
}
```

Output:

static block is invoked

Since JDK 1.7 and above, output would be:

Error: Main method not found in class A3, please define the main method as:
public static void main(String[] args)
or a JavaFX application class must extend javafx.application.Application

Explanation

Class A3 in the provided Java code has a static block that uses System.exit(0); to end the program after printing "static block is invoked" to the console. Because of this, the static block runs when the class loads into memory, printing the message and stopping the program right away to stop more code from running.

4) Java Nested Classes

In Java, a nested class is a class declared within another class. There are four types of nested classes:

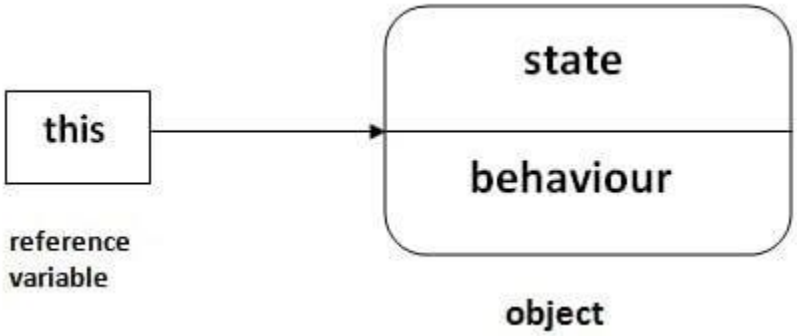
1. Static Nested Class: A static nested class is declared with the static keyword. It behaves like a regular top-level class but is nested for packaging convenience. Static nested classes cannot directly access non-static members of the enclosing class.
2. Non-static Nested Class (Inner Class): It is also known as an inner class; it is declared without the static keyword. Inner classes have access to the members, including private members, of the enclosing class. They are often used to logically group classes that only have meaning in the context of another class.
3. Local Inner Class: A local inner class is declared within a method or a code block. It has access to the variables of the enclosing method and can access members of the enclosing class.
4. Anonymous Inner Class: An anonymous inner class is a local inner class without a name. It is typically used to create small, one-time-use classes, often for implementing interfaces or extending classes.

Nested classes can be sensibly grouped together using classes that are only used once, improving code organization and encapsulation by concealing the outer class's implementation specifics.

Nested classes provide a way to logically group classes that are only used in one place, thereby increasing encapsulation and code organization. They also enhance encapsulation by hiding the implementation details of the outer class.

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.

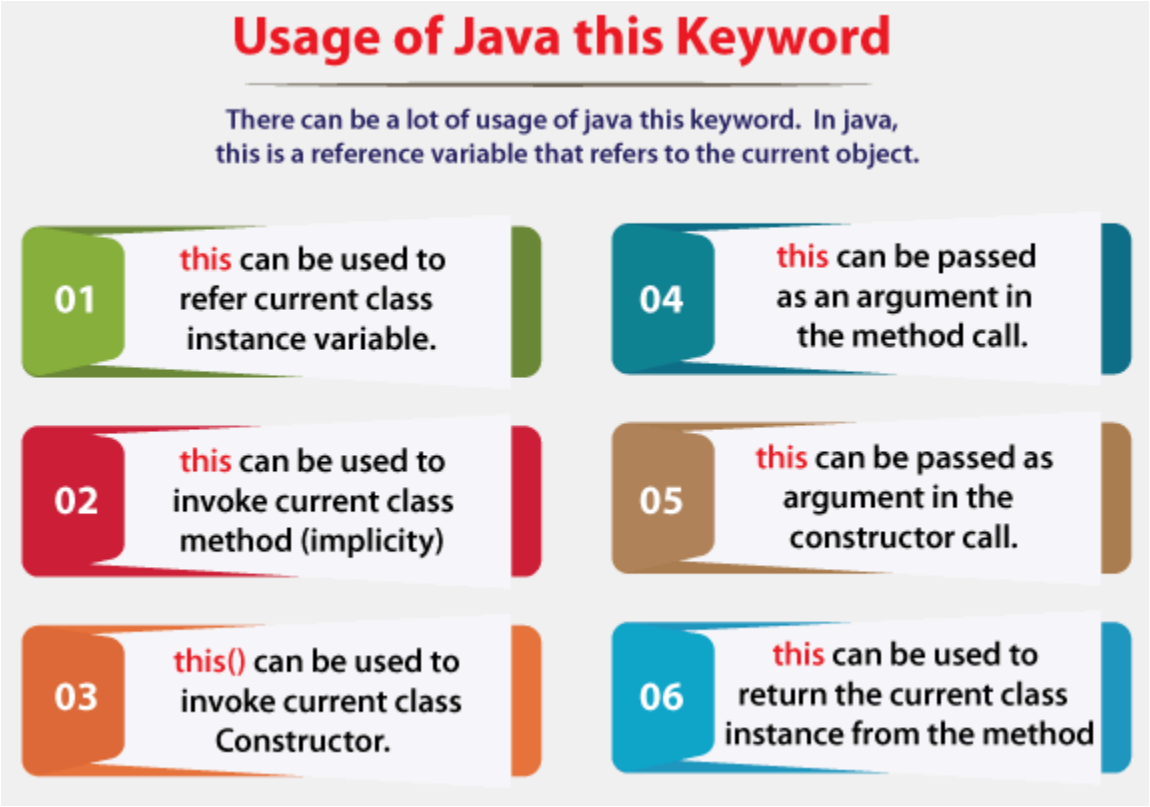


Usage of Java this keyword

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 usages 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:

```
class Student{
int rollno;
String name;
float fee;
Student(int rollno,String name,float fee){
rollno=rollno;
name=name;
fee=fee;
}
void display(){System.out.println(rollno+" "+name+" "+fee);}
}
class TestThis1{
public static void main(String args[]){
Student s1=new Student(111,"ankit",5000f);
Student s2=new Student(112,"sumit",6000f);
s1.display();
s2.display();
}}
```

Test it Now

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

```
class Student{
int rollno;
String name;
float fee;
Student(int rollno,String name,float fee){
```

```
this.rollno=rollno;
this.name=name;
this.fee=fee;
}
void display(){System.out.println(rollno+" "+name+" "+fee);}
}

class TestThis2{
public static void main(String args[]){
Student s1=new Student(111,"ankit",5000f);
Student s2=new Student(112,"sumit",6000f);
s1.display();
s2.display();
}}
```

Test it Now

Output:

111 ankit 5000.0
112 sumit 6000.0

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

```
class Student{
int rollno;
String name;
float fee;
Student(int r,String n,float f){
rollno=r;
name=n;
fee=f;
}
void display(){System.out.println(rollno+" "+name+" "+fee);}
}

class TestThis3{
public static void main(String args[]){
```

```
Student s1=new Student(111,"ankit",5000f);
Student s2=new Student(112,"sumit",6000f);
s1.display();
s2.display();
}}
```

Test it Now

Output:

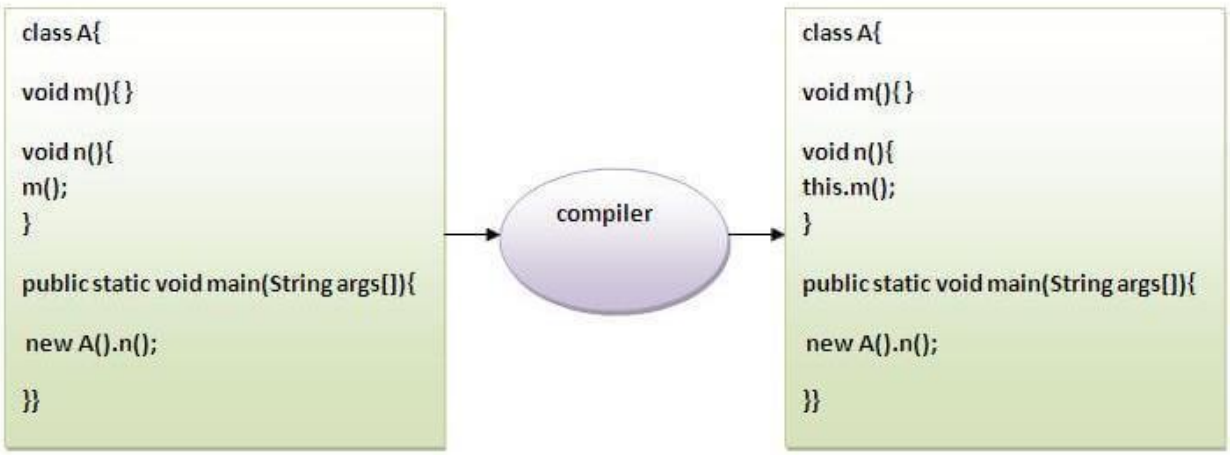
111 ankit 5000.0
112 sumit 6000.0

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




```
class A{
void m(){System.out.println("hello m");}
void n(){
```

```
System.out.println("hello n");
//m();//same as this.m()
this.m();
}
}
class TestThis4{
public static void main(String args[]){
A a=new A();
a.n();
}}
```

Test it Now

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:

```
class A{
A(){System.out.println("hello a");}
A(int x){
this();
System.out.println(x);
}
}
class TestThis5{
public static void main(String args[]){
A a=new A(10);
}}
```

Test it Now

Output:

hello a

10

Calling parameterized constructor from default constructor:

```
class A{
A(){
this(5);
System.out.println("hello a");
}
A(int x){
System.out.println(x);
}
}
class TestThis6{
public static void main(String args[]){
A a=new A();
}}
```

Test it Now

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.

```
class Student{
int rollno;
String name,course;
float fee;
Student(int rollno,String name,String course){
this.rollno=rollno;
this.name=name;
this.course=course;
}
```

```
Student(int rollno,String name,String course,float fee){
this(rollno,name,course);//reusing constructor
this.fee=fee;
}
void display(){System.out.println(rollno+" "+name+" "+course+" "+fee);}
}
class TestThis7{
public static void main(String args[]){
Student s1=new Student(111,"ankit","java");
Student s2=new Student(112,"sumit","java",6000f);
s1.display();
s2.display();
}}
```

Test it Now

Output:

111 ankit java 0.0
112 sumit java 6000.0

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

```
class Student{
int rollno;
String name,course;
float fee;
Student(int rollno,String name,String course){
this.rollno=rollno;
this.name=name;
this.course=course;
}
Student(int rollno,String name,String course,float fee){
this.fee=fee;
this(rollno,name,course);//C.T.Error
}
void display(){System.out.println(rollno+" "+name+" "+course+" "+fee);}
}
```

```
class TestThis8{
public static void main(String args[]){
Student s1=new Student(111,"ankit","java");
Student s2=new Student(112,"sumit","java",6000f);
s1.display();
s2.display();
}}
```

Test it Now

Output:

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:

```
class S2{
  void m(S2 obj){
  System.out.println("method is invoked");
  }
  void p(){
  m(this);
  }
  public static void main(String args[]){
  S2 s1 = new S2();
  s1.p();
  }
}
```

Test it Now

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:

```
class B{
  A4 obj;
  B(A4 obj){
    this.obj=obj;
  }
  void display(){
    System.out.println(obj.data);//using data member of A4 class
  }
}

class A4{
  int data=10;
  A4(){
   B b=new B(this);
   b.display();
  }
  public static void main(String args[]){
   A4 a=new A4();
  }
}
```

Test it Now

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

```
return_type method_name(){
```

```
return this;
}
```

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

```
class A{
A getA(){
return this;
}
void msg(){System.out.println("Hello java");}
}
class Test1{
public static void main(String args[]){
new A().getA().msg();
}
}
```

Test it Now

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.

```
class A5{
void m(){
System.out.println(this);//prints same reference ID
}
public static void main(String args[]){
A5 obj=new A5();
System.out.println(obj);//prints the reference ID
obj.m();
}
}
```

Test it Now

Output:

A5@22b3ea59
A5@22b3ea59
