Programming Language II
CSE-215

Prof. Dr. Mohammad Abu Yousuf

yousuf@juniv.edu

Static and **this** keyword in Java

Java static keyword

- 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 an instance of the class.
- The static can be:
- Variable (also known as a class variable)
- Method (also known as a class method)
- Block
- Nested class

Java static variable

- If you 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.

Java static variable

Advantages of static variable

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

```
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 the object is created. All students have its 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.

```
1.//Java Program to demonstrate the use of static variable
2.class Student{
3.    int rollno;//instance variable
4.    String name;
5.    static String institute ="IIT";//static variable
6.    //constructor
7.    Student(int r, String n){
8.    rollno = r;
9.    name = n;
10.    }
11.    //method to display the values
12.    void display (){System.out.println(rollno+" "+name+" "+institute);}
13.}
14.//Test class to show the values of objects
15.public class TestStaticVariable1{
16. public static void main(String args[]){
17. Student s1 = new Student(101,"Rahim");
18. Student s2 = new Student(102,"Samia");
19. //we can change the institute of all objects by the single line of code
20. //Student.institute="IIT";
21. s1.display();
22. s2.display();
23. }
24.}
```

Output:
101 Rahim IIT
102 Samia IIT

Java 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 other objects. So each object will have the value 1 in the count variable.

Java static variable

```
1.//Java Program to demonstrate the use of an instance variable
2.//which get memory each time when we create an object of the class.
3.class Counter{
4.int count=0;//will get memory each time when the instance is created
5.
6.Counter(){
7.count++;//incrementing value
8.System.out.println(count);
9.}
10.
11.public static void main(String args[]){
12.//Creating objects
13.Counter c1=new Counter();
14.Counter c2=new Counter();
15.Counter c3=new Counter();
16.}
17.}
```

Output:
1
1
1

```
1./Java Program to illustrate the use of static variable which
2.//is shared with all objects.
3.class Counter2{
4.static int count=0;//will get memory only once and retain its value
5.
6.Counter2(){
7.count++;//incrementing the value of static variable
8.System.out.println(count);
9.}
10.
11.public static void main(String args[]){
12.//creating objects
13.Counter2 c1=new Counter2();
14.Counter2 c2=new Counter2();
15.Counter2 c3=new Counter2();
16.}
17.}
```

Output:
1
2
3

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.

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 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 member and can change the value of it.

```
1.//Java Program to get the cube of a given number using the static method
2.
3.class Calculate{
4.  static int cube(int x){
5.  return x*x*x;
6.  }
7.  public static void main(String args[]){
8.  int result=Calculate.cube(5);
9.  System.out.println(result);
10.  }
11.}
```

Output:125

Restrictions for the static method

- There are two main restrictions for the static method. They are:
  - The static method can not use non static data member or call non-static method directly.
  - this and super cannot be used in static context.

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

Output: Compile Time Error

this keyword in java

- In java, this is a **reference variable** that refers to the current object.

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.

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

```
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.}}
```

Output:
0 null 0.0
0 null 0.0

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

```
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.}
12.
13.class TestThis2{
14.public static void main(String args[]){
15.Student s1=new Student(11,"ankit",5000);
16.Student s2=new Student(112,"sumit",6000);
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:

```
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.}
12.
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

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.

```
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.

Output:
hello a
10

```
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.}}
```

Calling default constructor from parameterized constructor

3) this() : to invoke current class 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

Calling parameterized constructor from default constructor:

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

Real usage of this() constructor call

```
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",6000);
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.

Real usage of this() constructor call

```
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.

```
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

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

```
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.}
10.
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.}
```

We can pass the **this** keyword in the constructor also. It is useful if we have to use one object in multiple classes.

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). 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
```

Thank you