Khulna Khan Bahadur Ahsanullah University

Object-oriented programming

CSE 1203

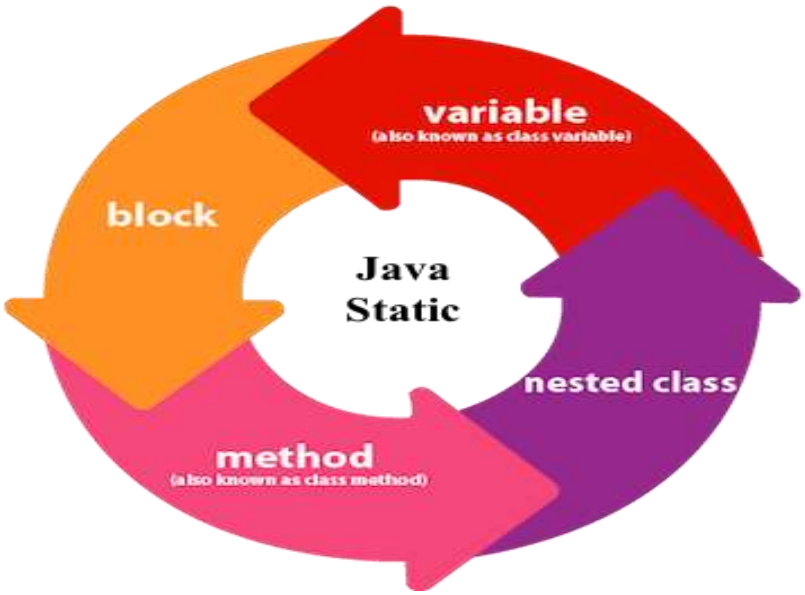
Lecture -8

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



1) 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.

Advantages of static variable

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

Understanding the 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 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.**

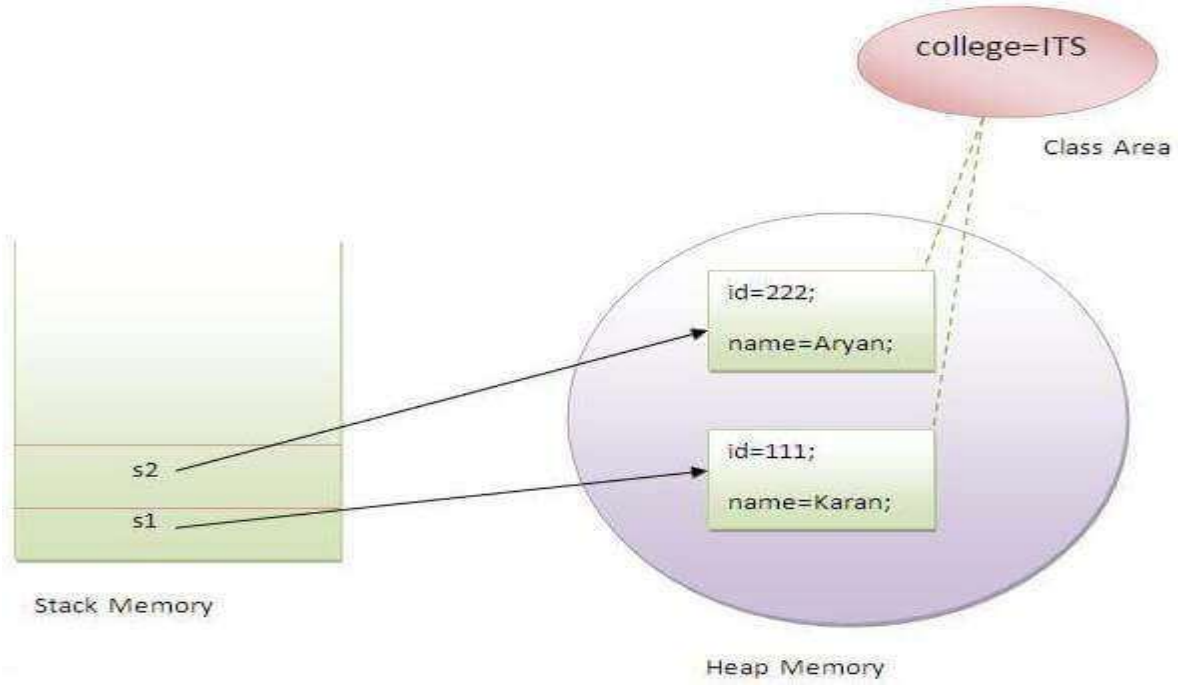
> Java static property is shared to all objects.

Example of static variable

```
1.  //Java Program to demonstrate the use of static variable
2.  class Student{
3.     int rollno;//instance variable
4.     String name;
5.     static String college ="ITS";//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+" "+college);}
13.//Test class to show the values of objects
14.public class TestStaticVariable1{
15. public static void main(String args[]){
16. Student s1 = new Student(111,"Karan");
17. Student s2 = new Student(222,"Aryan");
18. //we can change the college of all objects by the single line of code
19. //Student.college="BBDIT";
20. s1.display();
21. s2.display();
22. }  }
```

Output:

```
111 Karan ITS
222 Aryan ITS
```



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 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.

```
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
```

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

```
8.  }
9.
10.public static void main(String args[]){
11.//creating objects
12.Counter2 c1=new Counter2();
13.Counter2 c2=new Counter2();
14.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 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.

Example of static method

```
1.  //Java Program to demonstrate the use of a static method.
2.  class Student{
3.      int rollno;
4.      String name;
5.      static String college = "ITS";
6.      //static method to change the value of static variable
7.      static void change(){
8.      college = "BBDIT";
9.      }
10.     //constructor to initialize the variable
11.     Student(int r, String n){
12.     rollno = r;
13.     name = n;
14.     }
15.     //method to display values
16.     void display(){System.out.println(rollno+" "+name+" "+college);}
17.}
18.//Test class to create and display the values of object
19.public class TestStaticMethod{
20.   public static void main(String args[]){
21.   Student.change();//calling change method
22.   //creating objects
23.   Student s1 = new Student(111,"Karan");
24.   Student s2 = new Student(222,"Aryan");
```

```
25.    Student s3 = new Student(333,"Sonoo");
26.    //calling display method
27.    s1.display();
28.    s2.display();
29.    s3.display();
30.    }
31.}
```

```
Output:111 Karan BBDIT
       222 Aryan BBDIT
       333 Sonoo BBDIT
```

Another example of a static method that performs a normal calculation

```
1.  //Java Program to get the cube of a given number using the static method
2.  class Calculate{
3.    static int cube(int x){
4.    return x*x*x;
5.    }
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 cannot 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 a=40;//non static
3.
4.  public static void main(String args[]){
5.   System.out.println(a);
6.  }
7. }
```

Output: Compile Time Error

Q) Why is the Java main method static?

Ans) It is because the object is not required to call a static method. If it were a non-static method, JVM creates an 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 the 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?

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.

```
1. class A3{
2.   static {
3.   System.out.println("static block is invoked");
4.   } }
```

Output:

```
static block is invoked
```

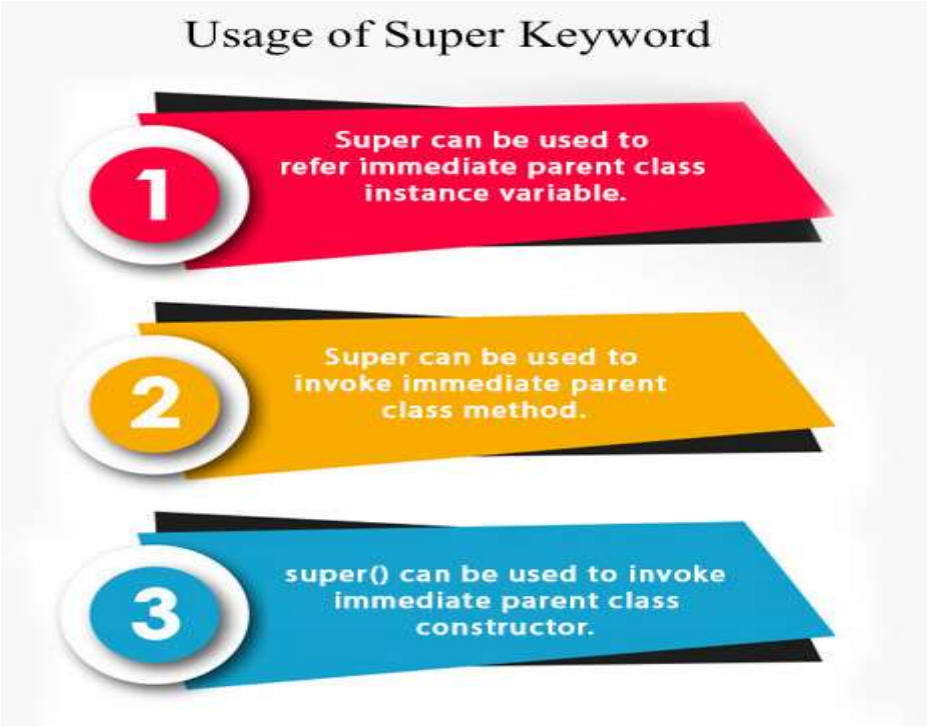
Super Keyword in Java

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

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.

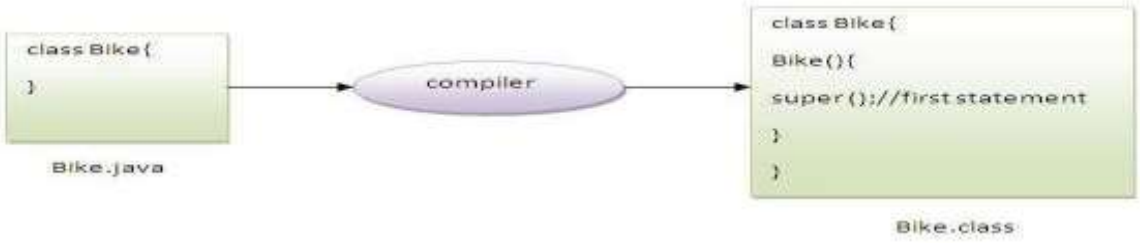
The super keyword can also be used to invoke the parent class constructor.

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

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
```