LEC18 OOP 2018-2019

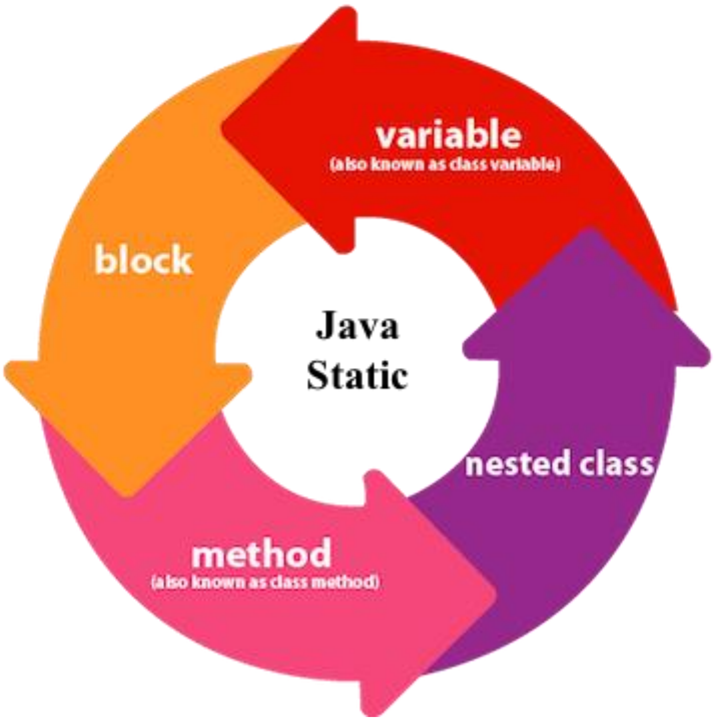
Static keyword in java OOP

The **static keyword in Java** means that the variable or function is shared between all instances of that class as it belongs to the type, not the actual objects themselves. So if you have a variable: private **static** int i = 0; and you increment it (i++) in one instance, the change will be reflected in all instances.

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:

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 class 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. it makes your program **memory efficient** (i.e., it saves memory).
- A class variable is declared using static keyword and may be explicitly initialized.
- As constants declared as final. Typically such constants are made public to be used anywhere within a program. Many java library classes make use of such public static final variables.
- The variable initialization rules state that static variables must be initialized either explicitly (بشكل واضح) or by taking the default value of the variable type (القيمة التي تفرضها قواعد اللغة). In addition, a static variable initialization can't refer to any static variables declared after its own declaration.

Example 1 (Correct or not)

```
Class Example{
…………
public Static int a=b*10;        Error
public Static int b=3;
………….. }
```

```
Class Example{
…………                          Correct
public Static int b=3;
public Static int a=b*10;
…………..
}
```

```
public class A {
public   int b=3;                     Error
 public static  int a=3*b; }
```

```
public class A {
public  static  int b=3;              Correct
 public  int a=3*b;  }
```

Example 2 (Understanding the problem without static variable)

```
public class Student{
private int rollno;
private    String name;
public     String college="ITS";  }
```

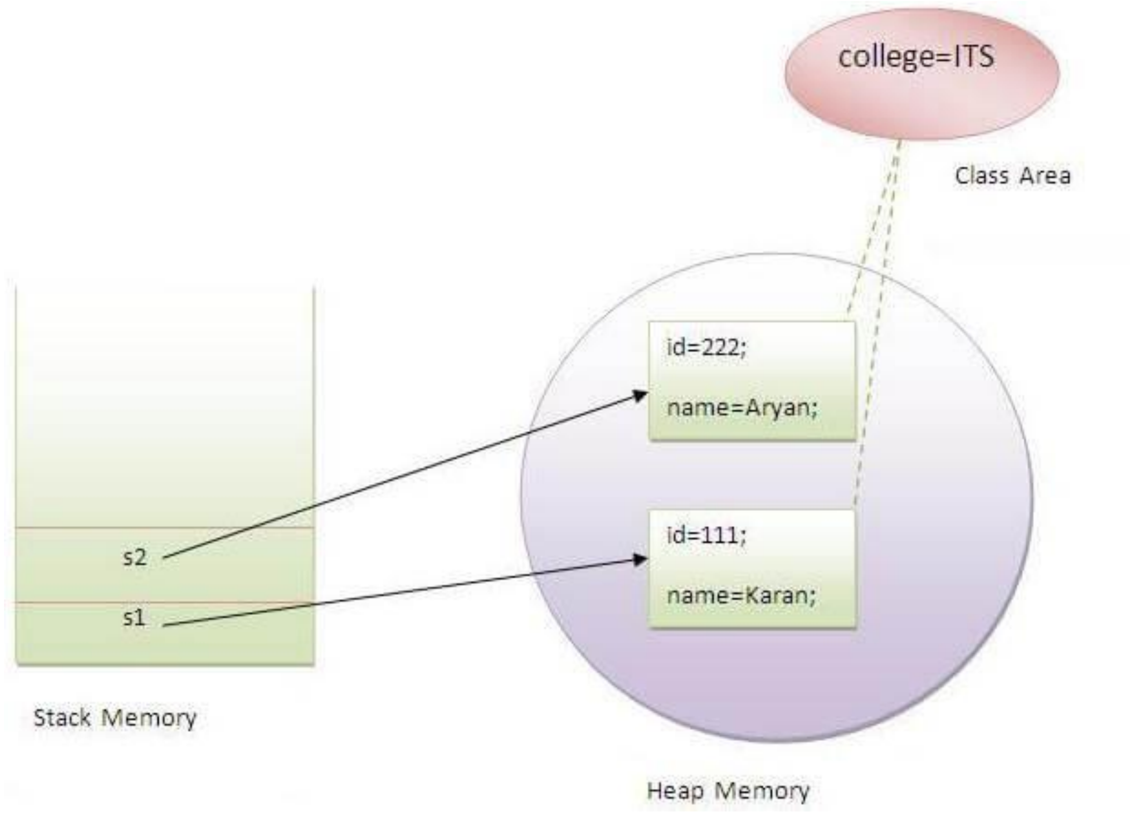
Suppose there are 500 students in my college, now all instance data members will get memory each time when the object is created. All students have its unique rollno and name, so instance data member is good in such case. Here, "college" refers to the common property of all objects. **<u>If we make it static, this field will get the memory only once.</u>**

Example 3 (using static variable for efficient use of memory)

```
public  class Student{

  private int rollno;
  private String name;
  public static String college ="ITS";//static variable

  public  Student(int r, String n){
   rollno = r;
   name = n;     }

  public  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();    }  }
```

The output will be:

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



Example4 (write a program to construct a counter by using a static variable)

```
public class Counter2{
public static int count=0;
Counter2(){  count++; System.out.println(count); }  }

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

Counter2 c1=new Counter2();
Counter2 c2=new Counter2();
Counter2 c3=new Counter2(); }  }
```

The Output will be:

```
1
2
3
```

2) Java static method

There are two types of methods:

- Instance methods
- Static methods

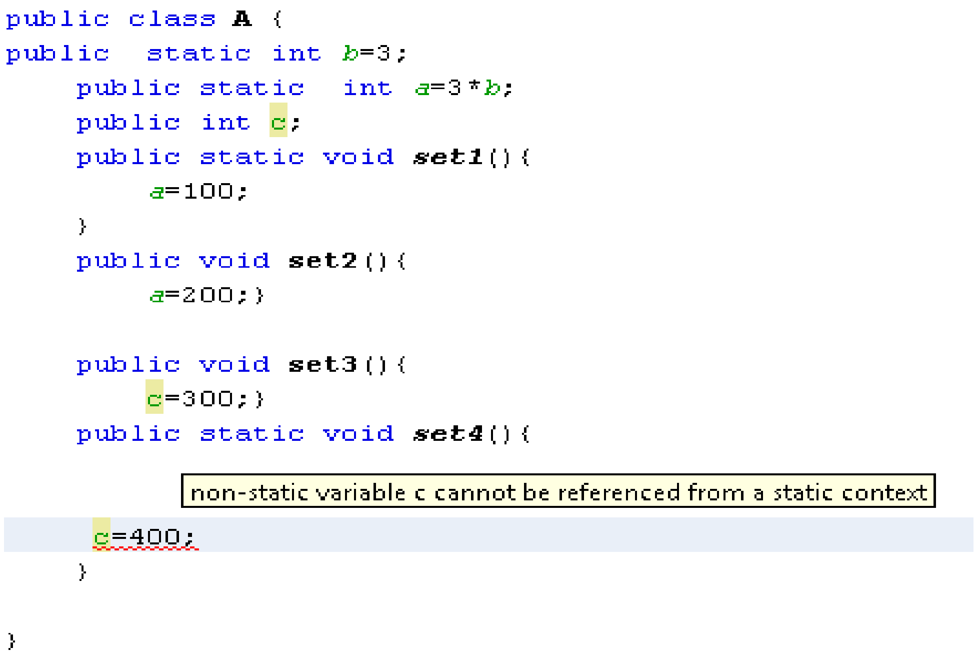
If you apply static keyword with any method, it is known as **static method**.

- 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.
- When a variable is defined at the class level, it is accessible from every member method of the class.

Example 5 (Trace)

```
public class C {
private int n=0;
public static int s=0;
public int getn() {  return (n);  //OK }
public int gets() { return s;  //OK }
public static int get2n() { return n; //ERROR!! } } WHY?!!!
```

Example 6 (Trace) (static method manipulates static variable only)



```
public class A {
public  static int b=3;
    public static  int a=3*b;
    public int c;
    public static void set1(){
        a=100;
    }
    public void set2(){
        a=200;}

    public void set3(){
        c=300;}
    public static void set4(){

            non-static variable c cannot be referenced from a static context
      c=400;
    }

}
```

Example 7 (showing static methods calling)

```
public class A {
public  static int b=3;
    public static  int a=3*b;

     public static void set1(){         a=100;     }
    public void set2(){          a=200;}
    public void set3(){         a=300;}
    public static void set4(){     a=400;   } }

public class Main {
    public static void main(String[] args) {
    A aa=new A();
        aa.set1();
        System.out.println(aa.a+"   "+aa.b);
        System.out.println(A.a+"   "+A.b);

        aa.set3();
       System.out.println(aa.a+"   "+aa.b);
        System.out.println(A.a+"   "+A.b);

       aa.set4();
       System.out.println(aa.a+"   "+aa.b);
       System.out.println(A.a+"   "+A.b);

       aa.set2();
       System.out.println(aa.a+"   "+aa.b);
       System.out.println(A.a+"   "+A.b);

       A.set1();
      System.out.println(aa.a+"   "+aa.b);
      System.out.println(A.a+"   "+A.b);

       // A.set2();  Error              // A.set3();  Error

        A.set4();
        System.out.println(aa.a+"   "+aa.b);
       System.out.println(A.a+"   "+A.b);     }    }
```

The output will be:

```
100   3
100   3

300   3
300   3

400   3
400   3

200   3
200   3

100   3
100   3

400   3
400   3
```

Properties of a static members

1. The static method can be inherited.

Example 8

```
public class Scott {
public static void abc() {
System.out.println("aaa");  }  }
public class Group extends Scott { …………}
public class Main{
public static void main(String[] args){
Group.abc();   }  }
```

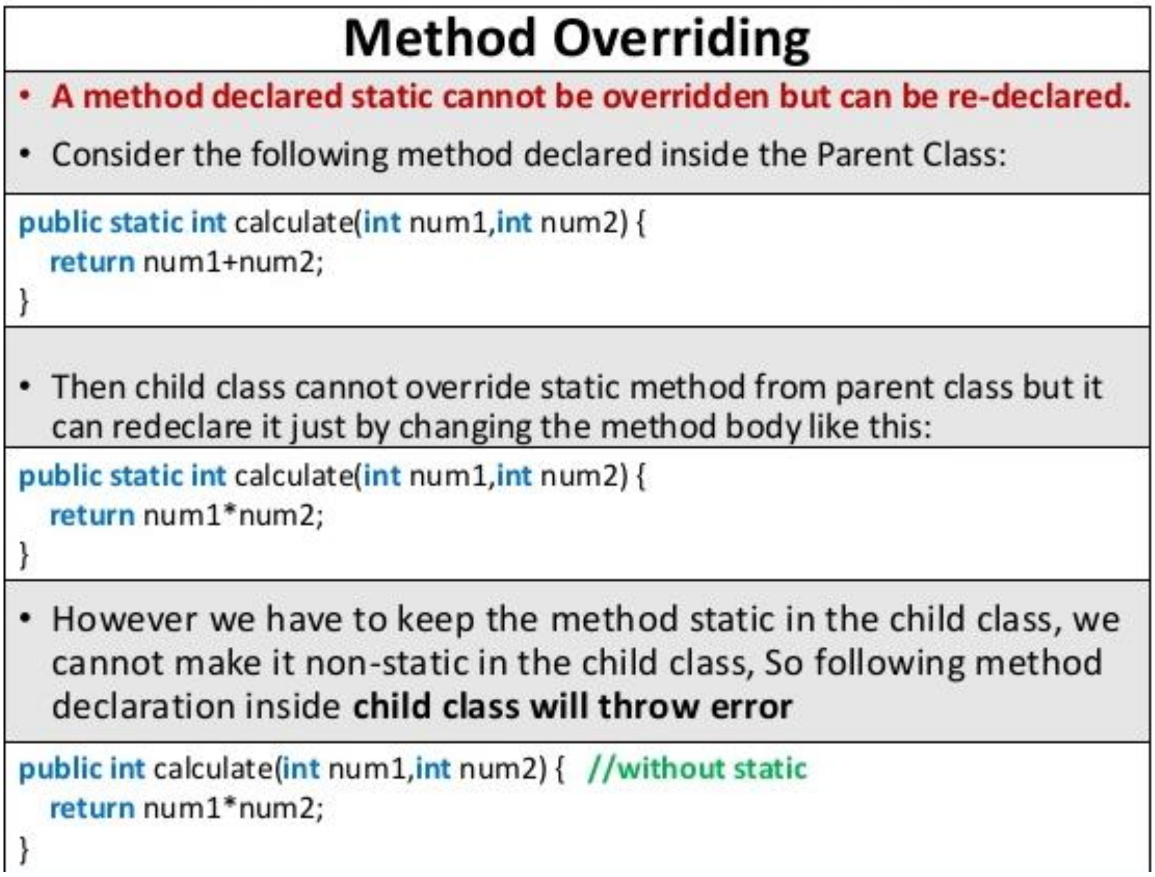
2. The static method can be **redefining**, and the static method cannot be **overriding**

Example 9

```
public class Scott {
public static void abc() { System.out.println("aaa"); } }
public class Group extends Scott {
public static void abc() { System.out.println("zzz");}}
Public class Main{
public static void main(String[] args){
Group.abc();  }  }
```

The output is : zzz

Q1:What is the difference between method overriding and redefining (redeclaring) in java?



Method Overriding

- A method declared static cannot be overridden but can be re-declared.
- Consider the following method declared inside the Parent Class:

```
public static int calculate(int num1,int num2) {
   return num1+num2;
}
```

- Then child class cannot override static method from parent class but it can redeclare it just by changing the method body like this:

```
public static int calculate(int num1,int num2) {
   return num1*num2;
}
```

- However we have to keep the method static in the child class, we cannot make it non-static in the child class, So following method declaration inside **child class will throw error**

```
public int calculate(int num1,int num2) {   //without static
   return num1*num2;
}
```

Q2: An abstract class may have static fields and static methods. You can use these static members with a class reference
for example,
AbstractClass.staticMethod()—as you would with any other class.

Q3: Abstract class cannot be instantiated. If I don't declare the methods in Abstract class static, how can I call them?
Answer : you need to extend the class

Q4: Explain the following and support your answer by programming example?

- **Static method cannot be overridden.**
- **Instance methods cannot be overridden by static method.**

Example 10 (important case1)

```
public class Animal {
public static void hide() {
System.out.println("The hide method in Animal.");}
public void override(){System.out.println("The override method in Animal."); }}

public class Cat extends Animal {
public static void hide(){ System.out.println("The hide method in Cat."); }
public void override() {System.out.println("The override method in Cat.");}}
public class Main{
public static void main(String[] args) {

 Cat myCat = new Cat();
Animal myAnimal = new Cat();

myAnimal.hide();   myAnimal.override();

myCat.hide();  myCat.override();
```

The output will be:

```
The hide method in Animal.
The override method in Cat.
The hide method in Cat.
The override method in Cat.
```

Example 10 (important case2)

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

Cat myCat = new Cat();
Animal myAnimal = (Animal)myCat;

myAnimal.hide(); myAnimal.override();
myCat.hide();  myCat.override();     } }
```

The output will be:

```
The hide method in Animal.
The override method in Cat.
The hide method in Cat.
The override method in Cat.
```

<u>An instance method cannot override a static method, and a static method cannot hide an instance method.</u>

Example 11 (Trace case 1)

```
public abstract class One {
   protected int x,y;
   public static void f1(){  System.out.println("static f1 from class One");}
   abstract public void f2();
   public void f3(){  System.out.println("f3 from One class");     }    }

public class Two extends One {
   private int a;
   public static void f1(){       // super.f1(); error so it is redefining not overriding
      System.out.println("redefining  method f1 from class Two");    }
   public void f2(){         //   super.f2(); Error
      System.out.println(" defining abstract method f2 from class Two");}
   public void f3(){System.out.println("method overriding from class Two");   } }

public class Main {
public static void main(String[] args) {
Two t=new Two();

t.f1();    t.f2();     t.f3()
One.f1();   Two.f1();     } }
```

The output will be:

```
redefining  method f1 from class Two
 defining abstract method f2 from class Two
method overriding from class Two
static f1 from class One
redefining  method f1 from class Two
```

Example 11 (Trace case 2)

if we modify class Two as below:

```
public class Two extends One {
   private int a;
   public static void f1(){ // super.f1(); error so it is redefining not overriding
      System.out.println("redefining  method f1 from class Two");     }
   public void f2(){         //   super.f2(); Error
      System.out.println(" defining abstract method f2 from class Two");}
   public void f3(){System.out.println("method overriding from class Two");
      super.f3();      } }
```

The output will be :

```
redefining  method f1 from class Two
defining abstract method f2 from class Two
method overriding from class Two
f3 from One class
static f1 from class One
redefining  method f1 from class Two
```

Example 11 (Trace case 3)

if we modify class Two as below:

```
public class Two extends One {
   private int a;
   public static void f1(){// super.f1(); error so it is redefining not overriding
      System.out.println("redefining  method f1 from class Two");   }
   public void f2(){  //   super.f2(); Error
  System.out.println(" defining abstract method f2 from class Two");}
   public void f3(){System.out.println("method overriding from class Two");
      super.f3();        this.f1();        this.f2();      } }
```

The output will be:

```
redefining  method f1 from class Two
 defining abstract method f2 from class Two
method overriding from class Two
f3 from One class
redefining  method f1 from class Two
 defining abstract method f2 from class Two
static f1 from class One
redefining  method f1 from class Two
```

H.W. (TRACE)

```
public class A {
public static int x;
public A(){     x=x+1;}
public void print (){     System.out.println("x="+x);} }

public class Main {

public static void main(String[] args) {
        //A.x=1;
        A a1=new A();
        a1.print();
        A a2=new A();
        a2.print();
System.out.println(A.x);
A a_array[]=new A[5];
for(int i=0;i<5;i++){
    a_array[i]=new A();  }
System.out.println(A.x);
a_array[0].print();
}
    }
```