**Declarations and Access Modifiers – Part-09- Static Modifier**

* Static Modifier:

Static is a modifier applicable for methods and variables, but not for classes.

We can declare top level class with static modifier, but we can declare inner class as static (such type of inner classes are called “static nested classes”)

In the case of instance variables for every object a separate copy will be created. But in the case of static variables a single copy will be created at class level and shared by every object of that class.

Example:

class Test{

static int x = 10;

int y = 20;

public static void main(String[] args){

Test t1 = new Test();

t1.x = 888;

t1.y = 999;

Test t2 = new Test();

System.out.println(t2.x+”…”+t2.y);

}

}

Output: 888…20

Note: for static variable only one copy will be there, and the instance variable will be recreated for every instance.

We can’t access instance members directly from static area, but we can access from instance area directly.

We can access static members from both instance and static areas directly.

Consider the following declarations:

1. int x = 10;
2. static int x = 10;
3. public void m1(){

System.out.println(x);

}

1. public static void m1(){

System.out.println(x);

}

Within the same class which of the above declarations we can take simultaneously?

1. 1 & 3 // Valid
2. 1 & 4 // CE: non-static variable x cannot be referenced from a static context
3. 2 & 3 // Valid
4. 2 & 4 // Valid
5. 1 & 2 // Invalid CE: variable x is already defined in Test
6. 2 & 3 // Invalid CE: m1() is already defined in Test.

* Conclusions:
  + Case\_01:

Overloading concept is applicable for static methods including main method. But JVM will always String[] args main method only.

Example:

class Test{

public static void main(String[] args){

System.out.println(“String[]”);

}

public static void main(int[] args){

System.out.println(“int[]”);

}

}

Output: String[]

Other overloaded method we have to call just like a normal method call.

* + Case\_02:

Inheritance concept applicable for static methods including main method. Hence while executing child class if child doesn’t contain main method then parent class main method will be executed.

Example:

class Parent{

public static void main(String[] args){

System.out.println(“Parent main”);

}

}

class Child extends Parent{

}

javac Parent.java

java Parent

Output: Parent main

java Child

Output: Parent main

* + Case\_03:

class Parent{

public static void main(String[] args){

System.out.println(“Parent main”);

}

}

class Child extends Parent{

public static void main(String[] args){

System.out.println(“Child main”);

}

}

Output: java Parent

Parent main

java Child

Child main

It seems overriding concept applicable for static methods but it is not overriding and it is method hiding.

Note:

For static methods overloading and inheritance concepts are applicable but overriding concept is not applicable. But instead of overriding method hiding concept is applicable.

* Instance method or static method:

Inside method implementation if we are using at-least one instance variable, then that method talks about a particular object hence we should declare method as instance method

Inside method implementation if we are not using any instance variable then this method nowhere related to a particular object. Hence we have to declare such type of method as static method irrespective of whether we are using static variables are not.

Example:

class Student{

String name;

int rollNo;

int marks;

static String collegeName;

}

getStudentInfo(){

return name+”…”+marks;

}

// Should be an instance method

getCollegeInfo(){

return collegeName;

}

//Should be a static method

getAverage(int x, int y){

return x+y/2;

}

//Utility method, should be a static method;

getCompleteInfo(){

return name+”…”+rollNo+”…”+marks+”…”+cname;

}

// Should be an instance method

For static methods implementation should be available whereas for abstract methods implementation is not available, hence abstract static combination is illegal for methods.

* Synchronized modifier:

Synchronized is a modifier applicable for methods and blocks but not for classes and variables.

If multiple threads trying to operate simultaneously on the same Java object then there may be a chance of “Data Inconsistency” problem, this is called “Race condition”.

We can overcome this problem by using “synchronized” keyword.

If a method or block declared as synchronized then at a time only one thread is allowed to execute that method or block on the given object. So that “Data Inconsitency” will be resolved.

But the main disadvantage of synchronized keyword is it increases waiting time of threads and creates performance problems. Hence if there is no specific requirement then it is not recommended to use synchronized keyword.

Note: Remember about the briyani and dog fight.

Synchronized method should compulsory contain implementation whereas abstract method doesn’t contain any implementation. Hence abstract synchronized is illegal combination of modifiers for methods.