**Declarations and Access Modifiers – Part-08- Final Variables**

* Final Variables:
  + Final Instance Variables:
    - If the value of a variable is varied from object to object such type of variables is called instance variables.
    - For every object a separate copy of instance variables will be created.
    - For instance variables we are not required to perform initialization explicitly. JVM will always provide default values.
      * Example:

class Test{

int x;

public static void main(String[] args){

Test t = new Test();

System.out.println(t.x);

}

}

Output: 0

* + - If the instance variable declared as final then compulsory we have to perform initialization explicitly whether we are using or not, and JVM won’t provide default values.
      * Example

class Test{

final int x;

}

Output: CE: variable x might not have been initialized.

* + - Rule:
      * For final instance variable compulsory, we should perform initialization before constructor completion. That is the following are various places for initialization.
        + At the time of declaration.
        + Inside instance block.
        + Inside constructor.
      * Declaration

class Test{

final int x = 10;

}

* + - * Instance Block

class Test{

final int x;

{

x = 10;

}

}

* + - * Inside Constructor

class Test{

final int x;

Test(){

x = 10;

}

}

* + - * The above are the only possible places to perform initialization for final instance variables. If we trying to perform initialization anywhere else then we will get compile time error.
        + Example:

class Test{

final int x;

public void m1(){

x = 10;

}

}

CE: cannot assign a value to final variable x

* + Final Static Variable:
    - If the value of the variable not varied from object to object, such type of variables are not recommended declare as instance variables. We have to declare those variables at class level by using “static” modifier.
    - 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.
    - For static variables it is not required to perform initialization explicitly. JVM will always provide default values.

class Test{

static int x;

public static void main(String[] args){

System.out.println(x);

}

}

Output: 0

* + - If the static variable declared as final then compulsory we should perform explicitly otherwise we will get compile time error. And JVM won’t any default values.

class Test{

final static int x;

}

Output: CE: variable x might not have been initialized

* + - Rule:
      * For final static variables compulsory we should perform initialization before class loading completion. That is the following are various places for this.
        + At the time of declaration.
        + Inside static block.
      * Time of Declaration:

class Test{

final static int x = 10;

}

* + - * Inside static block:

class Test{

final static int x;

static{

x = 10;

}

}

* + - The above are the only possible places to perform initialization for final static variables, if we are trying to perform initialization anywhere else then we will get compile time error.

class Test{

final static int x;

public void m1(){

x = 10;

}

}

Output: CE: cannot assign a value to final variable x

* + Final local variables:
    - Sometimes to meet temporary requirement of the programmer we have to declare variables inside a method or block or constructor. Such type of variables are called local variables or temporary variables or stack variables or automatic variables.
    - For local variables JVM won’t provide any default values, compulsory we should perform initialization explicitly before using that local variable. That is, if we are not using then it is not required to perform initialization for local variable.

Example:

class Test{

public static void main(String[] args){

int x;

System.out.println(“Hello”);

}

}

Output: Hello

class Test{

public static void main(String[] args){

int x;

System.out.println(x);

}

}

Output: local variable x might not have been initialized

* + - Even though local variable is final before using only we have to perform initialization, that is if we are not using then it is not required to perform initialization, even though it is final.

Example:

class Test{

final int x;

System.out.println(“Hello”);

}

Output: Hello

* + - The only applicable for local variable is final. By mistake if we are trying to apply any other modifier then we will get compile time error.

class Test{

public static void main(String[] args){

public int x = 10;

private int x = 20;

protected int x = 30;

static int x = 40;

transient int x = 50;

volatile int x = 60;

final int x = 10; // Only this is valid

}

}

Output: illegal start of expression

* + - Note:
      * If we are not declaring any modifier then by default the access modifier is default, but this rule is applicable only for instance and static variables, but not for local variables.
    - Formal Parameters:
      * Formal parameters of a method simply acts as local variables of that method. Hence formal parameter can be declared as final. If formal parameter declared as final, then within the method we can’t perform reassignment.
      * Example:

class Test{

public static void main(String[] args){

m1(10, 20);

}

public static void m1(fina int x, final int y){

x = 100;

y = 200;

System.out.println(x+”…”+y);

}

}

Output: CE: cannot assign a value to final variable x