Types of variables

**Division 1:**

Based on the type of value, variables are divided into two types

1. Primitive variables
2. Reference variables

Primitive variables

Can be used to refer primitive values

Ex: int x=10;

Reference variables

Can be used to refer objects

Ex: Student refVar = new Student ();

**Division 2:**

Based on the behavior and position of declaration, variables are divided into three types.

1. Instance variables
2. Static variables
3. Local variables.

**Instance variables**

If the value of a variable is varied from object to object such type of variables are called instance variables. For every object a separate copy of instance variables will be created.

Instance Variables will be created at the time of object creation and destroyed at the time of object destruction hence the scope of instance variable is exactly same as scope of objects.

Instance Variables will be stored in the heap area as the part of object creation

Instance Variables should be declare with in the class directly but outside of any method or block or constructor.

Instance Variables can access from instance area, cannot be accessed from static area

|  |
| --- |
| **class** InstanceVariables {  **int** i = 10;  **public** **static** **void** main(String[] args) {  // System.out.println(i);  // CE not -static variable i can not be referenced from a static context(invalid  InstanceVariables obj= **new** InstanceVariables();  System.***out***.println(obj.i);  obj.methodOne();  System.***out***.println(obj.i);  // t.methodOne();  }  **public** **void** methodOne() {  System.***out***.println(i);  }  } |

For the instance variables it is not required to perform initialization JVM will always provide default values.

|  |
| --- |
| **class** InstanceVariableDefaultValues {  **int** i;  **float** f;  **boolean** b;  **public** **static** **void** main(String args[]) {  InstanceVariableDefaultValues obj = **new** InstanceVariableDefaultValues();  System.***out***.println(obj.i);  System.***out***.println(obj.f);  System.***out***.println(obj.b);  } |

Instance variables also known as object level variables or objects.

**Static variables**

If the value of static variable is not varied from object to object such type of variables is not recommended to declare as instance variables. We have to declare such type of variables at class level by using static modifier.

In case of Instance variables for every object a separate copy will be created but in the case of static variables for entire class only one copy will be created and shared by every object of that class.

Static variables will be created at the time of class loading and destroyed at the time of class unloading hence the scope of the static variable is exactly same as the scope of the .class file.

Static variables will be stored in method area, static variables should be declared with in the class directly but outside of any method or block or constructor.

Static variables can be accessed from both instance and static areas directly.

We can access static variables by class name or by object reference but usage of class name is recommended.

1. java Test
2. Start JVM
3. Create and start main thread by JVM
4. Locate (find) Test.class by main Thread
5. Load Test.class by main Thread.
6. Execution of main method
7. unload Test.class // static variables destruction
8. Terminate main Thread
9. shutdown JVM

|  |
| --- |
| **public** **class** StaticVariable {  **static** **int** *i* = 10;  **public** **static** **void** main(String[] args) {  StaticVariable obj = **new** StaticVariable();  System.***out***.println(obj.*i*); // 10  System.***out***.println(StaticVariable.*i*);// 10  System.***out***.println(*i*);// 10  }  } |

For the static variables it is not required to perform initialization explicitly.

|  |
| --- |
| **public** **class** StaticVariableDefaultValues {  **static** String *s*;  **public** **static** **void** main(String[] args) {  System.***out***.println(*s*);  }  } |

Example:

|  |
| --- |
| **public** **class** InstanceAndStaticVariables {  **int** x = 10;  **static** **int** *y* = 20;  **public** **static** **void** main(String[] args) {  InstanceAndStaticVariables obj = **new** InstanceAndStaticVariables();  obj.x = 888;  obj.*y* = 999;  InstanceAndStaticVariables t2 = **new** InstanceAndStaticVariables();  System.***out***.println(t2.x + "-----------" + t2.*y*);  }  } |

Local Variables

Some times to meet temporary requirements of the programmer we can declare variables inside a method or block or constructor such type of variables are called local variables or automatic variables or temporary variables or stack variables. Local variables will be stored inside stack.

The local variables will be created as part of the block execution in which it is declared and destroyed once those block execution complete. Hence scope of the local variables is exactly same as scope of the block in which we declared.

Example:

|  |
| --- |
| **public** **class** LocalVariables {  **public** **static** **void** main(String[] args) {  **int** i = 0;  **for** (**int** j = 0; j < 3; j++) {  i = i + j;  System.***out***.println(i + "----" + j);  }  // System.out.println(i+"----"+j); //C.E cannot find symbol ,variable j  }  } |

The Local variables will be stored on the stack.

For the local variables JVM won’t provide any default values compulsory we should perform initialization before using that variable.

Ex:

|  |
| --- |
| **public** **class** LocalVariableDefaultValues {  **public** **static** **void** main(String[] args) {  **int** x;  System.***out***.println("hello");  System.***out***.println(x); // C.E variable x might not have initialized  }  } |

It is never recommended to perform initialization for the local variables inside blocks because there is no guarantee of executing that block at runtime. It is highly recommended to perform initialization for the local variables at the time of declaration at least with default values.

The only applicable modifier for local variables is final. if we are using any other modifier we will get compile time error.

example:

|  |
| --- |
| **public** **class** LocalVariableModifiers {  **public** **static** **void** main(String args[]) {  **public** **int** x1 = 10; // invalid  **private** **int** x2 = 10; // invalid  **protected** **int** x3 = 10; // invalid  **static** **int** x4 = 10; // invalid  **final** **int** x5 = 10; // valid  }  } |

**Conclusion**

1. For the static and instance variables it is not required to perform initialization explicitly JVM will provide default values. But for the local variables JVM won’t provide any default values, compulsory we should perform initialization explicitly before using those variables.

2. For every object a separate copy of instance variable will be created where as for entire class a single copy of static variable will be created . For every thread a separate copy of local variable will be created

3. Instance and static variables can be accessed by multiple threads simultaneously and hence these are not thread safe but local variables can be accessed by only one thread at a time and hence local variables are thread safe.

4. If we are not declaring any modifier explicitly then it means default modifier but this rule is applicable only for static and instance variables but not local variable.