**Declarations and Access Modifiers – Part-10- Native, Transient, Volatile**

* Native Modifier:

Native modifier is applicable only for methods and we can’t apply anywhere else.

The methods which are implemented in non-java (mostly C or C++) are called native methods or foreign methods.

The main objectives of native keyword are

1. To improve performance of the system.
2. To achieve machine level or memory level communication.
3. To use already existing legacy non-java code.

* Pseudo code to use native keyword in Java:

1. Load native libraires.
2. Declare a native method.
3. Invoke a native method.

class Native{

static{ // 1

System.loadLibrary(“Native library path”);

}

public native void m1(); // 2

}

class NativeClient{

public static void main(String[] args){

Native native = new Native();

native.m1(); // 3

}

}

For native method implementation is already available in old-languages like C or C++ and we are not responsible to provide implementation. Hence native method declaration should ends with semi-colon ;

public native void m1();

public native void m1(){} // Invalid

CE: native methods cannot have a body

For native methods implementation is already available in old languages. But for abstract methods implementation should not be available hence we can’t declare native method as abstract. That is native-abstract combination is illegal combination is illegal combination for methods.

We can’t declare native method as strictfp, because there is no guarantee that old languages follow IEEE 754 standard. Hence native-strictfp combination is illegal combination for methods.

* Advantage and Disadvantage of Native:

The main advantage of native keyword is performance will be improved, but the main disadvantage of native keyword is, it breaks platfrom independent nature of Java. As the implementation is depends on the older languages which are platform dependent languages.

* transient Keyword:
  + Transient is the modifier applicable only for variables.
  + We can use transient keyword in Serialization context.
  + At the time of serialization if we don’t want to save the value of a particular variable to meet security constraint, then we should declare that variable as transient.
  + At the time of serialization JVM ignores original value of transient variable and save default value to the file. Hence “transient means not to serialize”.
  + Diagram:

Username: Saravana

Password: null

serialization

User Object:

username: saravana

transient password: rithika

deserialization

User Object: user.ser

Username: saravana

Password: null

* volatile modifier:
  + volatile is a modifier applicable only for variables and we can’t apply anywhere else.
  + If the value of a variable keeps on changing by multiple threads, then there may be a chance of data inconsistency problem, we can solve this problem by using “volatile” modifier.
  + If a variable declared as volatile, then for every thread JVM will create a separate local copy.
  + Every modification performed by the thread will takes place in local copy so that there is no effect on the remaining threads.
  + The main advantage of volatile keyword is we can overcome data inconsistency problem. But the main disadvantage of volatile keyword is creating and maintaining a separate copy for every thread increases complexity of programming and creates performance problems. Hence if there is no specific requirement it is never recommended to use volatile keyword. And it is almost deprecated keyword.
  + Final variable means the value never changes, whereas volatile variable means the value keep on changing. Hence volatile-final is illegal combination for variables.
* Table format of all modifiers:

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Modifier | classes | | Methods | Variables | blocks | interfaces | | | enum | | | constror |
| Outer | Inner | Outer | Inner | | Outer | Inner | |
| public | Y | Y | Y | Y | N | Y | | Y | Y | | Y | Y |
| private | N | Y | Y | Y | N | N | | Y | N | | Y | Y |
| protected | N | Y | Y | Y | N | N | | Y | N | | Y | Y |
| default | Y | Y | Y | Y | N | Y | | Y | Y | | Y | Y |
| final | Y | Y | Y | Y | N | N | | N | N | | N | N |
| abstract | Y | Y | Y | N | N | Y | | Y | N | | N | N |
| static | N | Y | Y | Y | Y | N | | Y | N | | Y | N |
| synchronized | N | N | Y | N | Y | N | | N | N | | N | N |
| native | N | N | Y | N | N | N | | N | N | | N | N |
| strictfp | Y | Y | Y | N | N | Y | | Y | Y | | Y | N |
| transient | N | N | N | Y | N | N | | N | N | | N | N |
| volatile | N | N | N | Y | N | N | | N | N | | N | N |

* The only applicable modifier for local variable is final.
* The only applicable modifiers for constructors are public, private, protected, default.
* The modifiers which are applicable only for methods native.
* The modifiers which are applicable only for variables volatile and transient.
* The modifiers which are applicable for classes but not for interface, final.
* The modifiers which are applicable for classes but not for enum final and abstract.