**Access Modifiers**

Java source file structure

1. We can have any number of classes within the single Java file.
2. We can have almost one public class within the single Java file and the name of the file should be same as public class
3. If there is no public class in java source file, we can give any name as file name.
4. If we declare more than one class as public then compile time error will throw.
5. Whenever we are trying to compiling a java program, for every class present into the program a separate class file will generated.
6. We can compile java source file, but we can run individual java class file.
7. Whenever we are executing a java class the corresponding class main method will be invoked. If the class doesn’t contain main method, then error will throw.

Fully Qualified Name

1. It reduces the readability of the code. We can solve it by import statement.

Java.util.ArrayList al = new Java.util.ArrayList();

Instead of this we can write

Import java.util.ArrayList;

ArrayList al = new ArrayList();

Import statements

Types of import statements

1. Explicit Import (import java.util.ArrayList)
2. Implicit import (import java.util.\*)

Compiler will give priority by below order

1. Explicit class import
2. Class present in the current working directory
3. Implicit class import

Whenever we are importing a java package all the classes and interfaces present in the package by default available. But not sub package level.

To import Pattern class in our program

Import java.\*

Import java.util.\*

Import java.util.regex.\* (Regex is sub package which cannot access when importing util package)

All the classes and interfaces are present in the following packages are available by default for every Java program. Hence are not required to use import statement

1. Java.lang
2. Default package(Current working directory)

If we have more import statements then more time will be taken for compile the program. But it won’t affect the execution of the program.

In the case of Java imports, no class files are loaded in the beginning. Whenever we are asking for particular class then only that class file will be loaded. This is like dynamic include. This is called as local on demand or dynamic include of load on fly.

Static import

1. Usually we can access static member by using class name but whenever we are writing static import we access static member directly.
2. System.out.println()
   1. System = It is class present in the java.lang package
   2. Out = It is static variable present in the system class of type printStream
   3. It is a method present in the printstream class
3. While resolving static members compiler will always consider the precedence in following order
   1. Current class static member
   2. Explicit static member
   3. Implicit static member
4. Two packages contains a class or interface with the same name is very rare. And hence ambiguity problem is also very rare in normal import.
5. But two classes or interfaces can contain a variable or method with same name is very common and hence ambiguity is very common problem with static import.
6. Usage of static import reduces the readability of the program and creates the confusions.

|  |  |
| --- | --- |
| **Normal import** | **Static import** |
| We can use normal import to import the classes and interfaces of a particular package | We can use static import to import the static members of a particular class or interface |
| Whenever we are using normal import it is not required to use fully qualified name and we can use short name | In the case if static import we are not require to use class name to access static members. We can access directly. |

Package statement

An encapsulation mechanism to group related classes and interfaces into a single unit is called package. All classes are required for database operations are grouped into java.sql package

Purpose of Package

1. To resolve naming conflicts.
2. To improve modularity of the application.
3. We can achieve security features.

To compile Java program within the package

Javac -d . Test.java

Here

Javac = Compiler command

-d = destination of class path

. = Represents the current working directory

Test.java = File name

To execute the program

Java packagename.Test

Important conclusions

1. In any java program, atmost one package statement should be write, otherwise we will get compiler error
2. In any java program, first non-comment line should be package statement if it is available. Otherwise compiler error will throw.
3. Valid Source File structure
   1. Package declarations (At most one package statement)
   2. Import statements (Any number of import statements)
   3. Classes | interfaces | enum declarations |

Note – We can compile Java source file without class or interface but to execute we need a class and main method

**Class level Modifiers**

1. Public - Can use for top level classes
2. Private - Can use for inner classes
3. Protected - Can use for inner classes
4. Default - Can use for top level classes
5. Final - Can use for top level classes
6. Abstract - Can use for top level classes
7. Static - Can use for inner classes
8. Synchronized
9. Native
10. StrictFp - Can use for top level classes
11. Transient
12. Volatile

Public

1. Public modifier can use for classes and methods and we call those classes and methods from anywhere.

Default Classes

1. If a class declared as default then we can access that class only within the current package.
2. From outside package we can’t access that class.
3. Default access is known as package level access.

Final Modifier

1. It is a modifier applicable for classes, methods and variables
2. Whatever the method parent has by default available to the child through inheritance.
3. If the child not satisfied parent method implementation, then the child is allowed to redefine that method based on its requirements. This process is called overriding
4. If a class is declared as final, then we can’t extend functionality of that class. We can’t create child class.
5. Inheritance is not possible for final classes.
6. If a class declared as final, then all the methods present inside the class are automatically final. But variables need not to be final.

Abstract Modifier

1. It is applicable for class, methods and not for variables.
2. Even though we don’t know about the implementation we can declare a method with abstract modifier.
3. For abstract methods only declaration is available but not the implementation
4. Child class is responsible to provide implementation of parent class abstract methods.
5. For any java class, if we not allow to create an object because of partial implementation, such type of class we have to declare as abstract class.
6. If a method is abstract, then that class must be the abstract, other compile time error will throw.
7. If a class contains at least one abstract method, the implementation is not complete. Hence it is not recommended to create a object. To restrict class instantiation class must be abstract.
8. A class can be abstract even though there is no abstract method in that class.
9. Abstract class can contain final method, where are final class cannot contain abstract method.

Strictfp Modifier

1. It is applicable for class, methods and not for variables.
2. Usually, the result of floating point of arithmetic is varied from platform to platform. If we want platform independent result then we should go for strictfp modifier.
3. StrictFp Method
   1. If a method declared as strinctfp all the floating point calculations in that method will follow IEEE 754 standard. So that we will get platform independent results.
4. Strictfp Class
   1. If a class declared as Strictfp all floating point calculations in concrete methods has to follow the IEEE754 standard.
   2. We can declare abstract strictfp combination for classes. It is legal for classes but not for methods.

**Member level modifier**

Public Member

1. If a member declared as public then we can access from anywhere. But the corresponding class should be visible.

Default Member

1. If a member declared as default, then we access that member from the current package.
2. Outside of the package we can’t access that member.
3. If we not specifying the modifier "default" modifier will assign.
4. But we cannot include default keyword on method signature except interface

Private member

1. Can access only within the class.

Protected Member

1. The most misunderstood modifier in Java
2. If a method declared as protected, then we can access member anywhere in the current package and child classes of outside packages.
3. We can access protected member from outside package only in child classes, and we should use that child class reference only.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Visibility** | **Private** | **Default** | **Protected** | **Public** |
| With the same class | Checkmark with solid fill | Checkmark with solid fill | Checkmark with solid fill | Checkmark with solid fill |
| From child class of same package |  | Checkmark with solid fill | Checkmark with solid fill | Checkmark with solid fill |
| From non-child class of same package |  | Checkmark with solid fill | Checkmark with solid fill | Checkmark with solid fill |
| From child class of outside package |  |  | Checkmark with solid fill | Checkmark with solid fill |
| From non-child class of outside package |  |  |  | Checkmark with solid fill |

Final variables

1. Final instance variables
   1. For instance variables, we are not require to perform, initialization explicitly. The JVM will always provide default values.
   2. If the instance variables declared as final, then JVM will not provide the default value.
   3. If we not providing any default value to the final variable then compiler will throw error.
   4. We have to perform initialization, before compilation of the constructor.
   5. We can initiate final variables in
      1. At the time of declaration
      2. Inside instance block
      3. Inside constructor
2. Final static variables
   1. If a value of the variable is changed from object to object, we can use static variables.
   2. Static variables are class level variables, the single copy of the variable will be created at class level and shared by every object of that class.
   3. For static variables JVM will provide default values but for final static variables the default initialization will not happen
   4. Following places to declare final static variable
      1. At the time of declaration
      2. Inside static block
      3. Before class loading completion
3. Final local variables
   1. The variables declared inside the methods are known as local variables or stack variables or automatic variables
   2. JVM will not provide any default value to the local variables if it is final or non-final
   3. Even though local variables are final, we can initialize them before the usage of them.
   4. If we are not using then it is not required to perform initialization
   5. The only applicable modifier for local variable is final, any other modifier like public, private , etc.

**Note** – if no modifiers specified before static or instance variables, then it will be default instance or default static variables. But it is not applicable for local variables.

Formal parameters of a method is also local variable of the method. If formal parameters of a method is final then we can’t perform reassignment.

Static Modifier

1. Static is modifier applicable for methods and variables and not for classes.
2. We can’t declare top level class as Static but we can declare inner class with static modifier. Such type of inner class is called static inner class.
3. In the case of instance variable for every object separate copy will be created.
4. In the case of static variable, a single copy will be created by class level and shared by every object.
5. We can’t access instance members directly from static area but we can access from instance area directly.
6. We access static member from both instance and static area directly.

Important Terminologies

1. Overloading concept applicable for static methods including main method. But JVM will call only method with String array param. Other method will act as normal method
2. Inheritance concept applicable for static methods. Hence execution child class if child doesn’t contain main method, then parent main method will be executed.
3. If child class contains main method, then the program will execute but method overriding won’t happen instead of overriding method hiding will happen.
4. Inside method implementation if we are using al least one instance variable then that method talks about particular object hence that method declared as instance method.
5. Inside method implementation if we are not using any instance variable then that method nowhere related to a particular object. Hence we can declare such type of method as static method, irrespective of whether we are using static variables are not.

Synchronized Modifier

1. It is applicable for methods, and blocks and not for classes and variables.
2. If a multiple threads trying to operate simultaneously on the same object, then there may be a chance of data inconsistency problem (RACE condition)
3. We can overcome this problem using Synchronized keyword
4. Synchronized method should contain implementation

Native Modifier

1. It is applicable for only methods and not for classes or variables
2. The methods which are implemented in non-java(Other languages), are called native methods or foreign methods
3. The main objective of Native keyword is
   1. To improve performance of the system
   2. To achieve machine and memory level communication
   3. To use already existing legacy non-Java code
4. Pseudo code of native
   1. Class Native{
   2. Static{
   3. System.loadLibrary(“Native library path”)
   4. }// Local native library
   5. Public native void m1();// Decalre native method
   6. }
   7. Class client{
   8. Public static void main(String[] a){
   9. Native n = new Native(); // invoke a native method
   10. N.m1();
   11. }
   12. }
5. The usage of native will break the platform independent nature of the Java since other language may dependent on the platform.

Transient Modifier

1. It is only applicable for variables
2. We can use this keyword in serialization context.
3. 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.
4. At that time of serialization JVM ignores original value of transient variable and save default variables to the file. Hence transient means no serialize.

Volatile

1. It is a modifier applicable only for variables and we can’t apply anywhere else.
2. If the value of variable keeps on changing by multiple threads, then there maybe chance of data inconsistency problem.
3. We can solve this problem 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 remaining threads.
4. The main advantage of volatile is we can overcome the data inconsistency problem. But the main disadvantages of volatile is creating and maintaining a separate copy for every thread, it increases the complexity of the program and creates performance problem.