**Declarations and Access Modifiers**

1. **Java Source file structure**

A java program can contain n number of classes. But at most one class can be declared as public.

If there is a public class then name of the program and name of the public class must be matched otherwise we will get compile time error

EG 1:

class A{}

class B{}

class C{}

case 1:

If there is no public class then we can use any name and there are no restrictions.

A.java, B.java, C.java, durga.java

case 2:

If class B is public then name of the program should be B.java. Otherwise we will get compile time error as “Class B is public, should be declared in a file named B.java”

case 3:

If class B and C declared as public and the name of the program is B.java then we will get compile time error saying “Class C is public, should be declared in a file named C.java”

EG 2:

**class** A

{

**public** **static** **void** main(String[] args) {

System.***out***.print("A class main");

}

}

**class** B

{

**public** **static** **void** main(String[] args) {

System.***out***.print("B class main");

}

}

**class** C

{

**public** **static** **void** main(String[] args) {

System.***out***.print("C class main");

}

}

**class** D

{

}

Java Durga.java

A.class B.class C.class D.class

Java A:

Output: A class main

Java B:

Output: B class main

Java C:

Output: C Class main

Java D:

Output: RE: NoSuchMethodError:main

Java Durga

Output: RE NoClassDefFoundError :Durga

CONCLUSIONS:

1. Whenever we are compiling a java program for every class present in that program a separate .class file will be generated.
2. We can compile a java program (java source file) but we can run a java .class file.
3. Whenever we are executing a java class the corresponding class main method will be executed.
4. If the class does not contain main method then we will get runtime exception saying

RE : NoSuchMethodError : main

1. If the corresponding .class file is not available then we will get runtime exception saying NoClassDefFoundError : Durga
2. It is not recommended to declare multiple classes in a single source file. It is highly recommended to declare only one class per source file and the name of the program should be same as class name.

The main advantage of this approach is readability and maintainability of the code will be improved.

**Import Statement**

EG:

**public** **static** **void** main(String[] args)

{

ArrayList i = **new** ArrayList();

}

Output: Unresolved compilation problems:

ArrayList cannot be resolved to a type

We can solve this problem by using fully qualified name.

**java.util.ArrayList** l = new **java.util.ArrayList**();

the problem with usage of fully qualified names every time is it increases length of the code and reduces readability.

We can solve this problem by using import statement

Whenever we are writing import statement it is not required to use fully qualified name every time we can use short name directly.

**import** java.util.ArrayList;

**public** **class** Rough

{

**public** **static** **void** main(String[] args)

{

**ArrayList** l = **new** ArrayList();

}

}

Hence import statement acts as typing shortcut.

**CASE 1: Types of import statements**

There are two types of import statements

1. Explicit class import
2. Implicit class import

**Explicit class import**

EG: import java.util.ArrayList

It is highly recommended to use explicit classs import because it improves readability of the code

Best suitable for Hi-tech city where readability is important.

**Implicit class import**

EG : import java.util.\*;

Not recommended to use because it reduces readability of the code.

Best suitable for ameer peth where typing is important

**CASE 2:**

Which of the following import statements are meaningful

1. Import java.util.AL; ..valid(explicit class import)
2. import java.util.AL.\*; ..invalid
3. import java.util.\*; ..valid(implicit class import)
4. import java.util; ..invalid

**CASE 3:**

Consider the following code

Class myObject extends java.rmi.UnicastRemoteObject{}

The code compiles file even though we are not writing import statement because we have used fully qualified name.

NOTE:

Whenever we are using fully qualified names it is not required to write import statement .

Similarly whenever we are writing import statement it is not required to use fully qualified name.

**CASE 4:**

**import** java.util.\*;

**import** java.sql.\*;

**public** **class** Rough

{

**public** **static** **void** main(String[] args)

{

Date i = **new** Date();

}

}

Output: Unresolved compilation problems:

The type Date is ambiguous

The type Date is ambiguous

Even in the case of List we may get same ambiguity problem because it is available in both util and awt package.

**CASE 5:**

While resolving class names compiler will always give some precedence in the following order.

1. Explicit class import
2. Classes present in current working directory(default package)
3. Implicit class import

EG:

**import** java.util.Date;

**import** java.sql.\*;

**public** **class** Rough

{

**public** **static** **void** main(String[] args)

{

Date i = **new** Date();

System.***out***.print(i.getClass().getName());

}

}

Output: java.util.Date

In the above example util package date got considered.

**CASE 6:**

Whenever we are importing a java package all classes and interfaces present in that package are by default available.

But not sub-package classes. If we want to use sub-package class compulsory we should write import statement until sub-package level.

EG:

java

util

regex

pattern

to use pattern class in our program which import statement is required.

1. import java.\*;
2. import java.util.\*;
3. import java.util.regex.\*; ..correct
4. no import required

**CASE 7:**

All classes and interfaces present in the following packages are by default available to every java program. Hence we are not required to write import statement.

1. java.lang package
2. default package (current working directory)

**CASE 8:**

Import statement is totally compile time related concept. If more number of imports then more will be the compile time.

But there is no effect on execution time.

**CASE 9:**

Difference between C language #include and java language import statement.

* In the case of C language #include all input output header files will be loaded at beginning only.(AT translation time).Hence it is static include.
* But in the case of java import statement no .Class file will be loaded at beginning . Whenever we are using a particular class then only corresponding .class file will be loaded. This is like dynamic include or load on demand or load on fly.
  1. **version new features**
* for each loop
* var-args methods
* autoboxing and autounboxing
* Generics
* Co-variant return type
* Queue
* Annotations
* enum
* Static import

**Static Import**

Introduced in 1.5 version

According to SUN usage of static import reduces length of the code and improves readability but according to world wide programming experts (like us) usage of static import creates confusion and reduces readability. Hence if there is no specific requirement then it is not recommended to use static import.

Usually we can access static members using class name but whenever we are writing static import we can access static members directly without class name.

EG: Without static import

**public** **class** Rough

{

**public** **static** **void** main(String[] args)

{

System.***out***.println(Math.*sqrt*(4));

System.***out***.println(Math.*max*(4,40));

System.***out***.print(Math.*random*());

}

}

EG: With static import

**import** **static** java.lang.Math.*sqrt*;

**import** **static** java.lang.Math.\*;

**public** **class** Rough

{

**public** **static** **void** main(String[] args)

{

System.***out***.println(*sqrt*(4));

System.***out***.println(*max*(4,40));

System.***out***.print(*random*());

}

}

**Explain about System.out.println?**

EG 1:

class Test

{

static String s = “java”;

}

Test.s.length()

Test = class name

s = static variable present in Test class of type java.lang.String

length() = it is a method present in String class

EG 2:

class System

{

static PrintStream out;

}

System.out.Println()

System = class name

Out = static variable present in System class of type PrintStream

Println() = it is a method present in printStream class

Out is a static variable present in system class. hence we can access it by using class name system but whenever we are writing static import it is not required to use class name and we can access out directly.

EG:

**import** **static** java.lang.System.***out***;

**class** Rough

{

**public** **static** **void** main(String[] args)

{

***out***.println("hello");

***out***.print("hi");

}

}

Output: Hello

Hi

EG:

**import** **static** java.lang.Integer.\*;

**import** **static** java.lang.Byte.\*;

**class** Rough

{

**public** **static** **void** main(String[] args)

{

System.***out***.println(MAX\_VALUE);

}

}

Output: Unresolved compilation problem:

The field MAX\_VALUE is ambiguous

While resolving static members compiler will always consider the precedence in the following order

1. Current class static members
2. Explicit static import
3. Implicit static import

EG 1:

**import** **static** java.lang.Integer.***MAX\_VALUE***;

**import** **static** java.lang.Byte.\*;

**class** Rough

{

**static** **int** *MAX\_VALUE* = 999;

**public** **static** **void** main(String[] args)

{

System.***out***.println(*MAX\_VALUE*);

}

}

Output: 999

EG 2:

**import** **static** java.lang.Integer.***MAX\_VALUE***;

**import** **static** java.lang.Byte.\*;

**class** Rough

{

//**static** **int** *MAX\_VALUE* = 999;

**public** **static** **void** main(String[] args)

{

System.***out***.println(*MAX\_VALUE*);

}

}

Output: 2147483647

EG 2:

**//import** **static** java.lang.Integer.***MAX\_VALUE***;

**import** **static** java.lang.Byte.\*;

**class** Rough

{

//**static** **int** *MAX\_VALUE* = 999;

**public** **static** **void** main(String[] args)

{

System.***out***.println(*MAX\_VALUE*);

}

}

Output: 127

If we comment line 1 (Static in ma) then explicit static import will be considered and hence integer class max value will be considered. In this case the output is 2147483647.

If we comment both lines 1 and 2 then implicit static import will be considered . in this case output is 127(byte class max value).

**Syntax:**

**Normal import:**

1. Explicit import:

Import packageName.className;

EG:

import java.util.ArrayList;

1. Implicit import:

import packageName.\*;

EG:

Import java.util.\*;

**Static import:**

1. Explicit static import:

import static packageName.className.staticMemberName;

EG:

import static java.lang.Math.sqrt;

import java.lang.System.out;

1. Implicit static import

import static packageName.className.\*;

EG:

import static java.lang.Math.\*;

import static java.lang.System.\*;

13457810

**Which of the following import statements are valid?**

1. import java.lang.Math.\*; ..invalid
2. import static java.lang.Math.\*; ..valid
3. import java.lang.Math.sqrt; ..invalid
4. import static java.lang.Math.sqrt(); ..invalid
5. import java.lang.Math.sqrt.\*; ..invalid
6. import static java.lang.Math.sqrt; ..valid
7. import java.lang; ..invalid
8. import static java.lang; ..invalid
9. import java.lang.\*; ..valid
10. import static java.lang.\*; invalid

Two packages contains a class or interface with same name is very rare and hence ambiguity problem is also very rare in normal import .

But two classes or interfaces contain a variable or method with same name is very common and hence ambiguity problem is also very common problem in static import.

Usage of static import reduces readability and creates confusion and hence if there is no specific requirement then it is not recommended to use static import .

Difference between normal import and static import

* we can use normal import to import classes and interfaces of a particular package. Whenever we are using normal import it is not required to use fully qualified names and we can use short names directly
* we can use static to import static members of a particular class or interface . whenever we are using static import it is not required to use class name to access static members and we can access directly