Eg: Without\_Import

// go through the code

We can resolve this problem by using fully qualified name “java.util.ArrayList”

I = new java.util.ArrayList(); . But problem with this fully qualified name is that, every time it increases the length of the code and reduces the code readability.

We can resolve this problem by using import statements

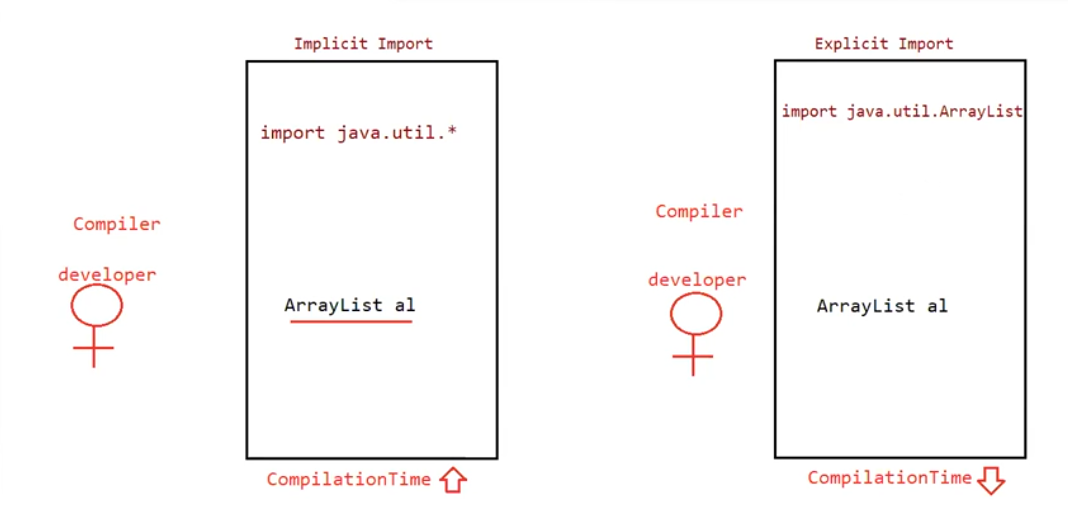
Eg: With\_Import

Hence whenever we are using import statement it is not required to used fully qualified name we can use short names directly. This approach decreases the code length and increases the code readability.

Case1: Types of import statements

There are 2 types of import statements

1. Explicit class import
2. Implicit class import



Explicit class import:

Example: Import java.util.ArrayList;

* These type of imports are highly recommended to use because it improves the readability of the code
* Best suitable for developers where readability is important

Implicit class import:

Example: Import java.util.\*;

* It is never recommended because it reduces the readability of the code.
* Best suitable for students where typing is important.

Case 2:

Which off the following statements are meaningful?

1. Import java.util;
2. Import java.util.ArrayList.\*;
3. Import java.util.\*;
4. Import java.util.ArrayList;

Ans: c and d

Case 3:

Consider the following code?

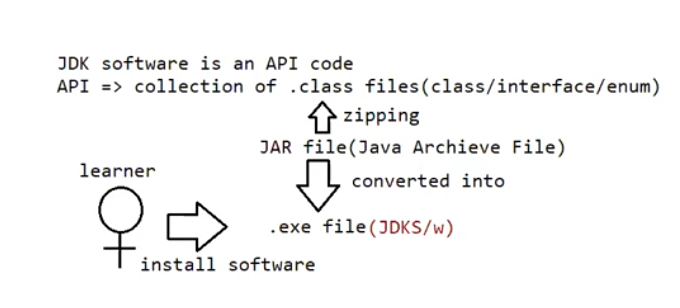
class MyArrayList extends java.util.ArrayList{

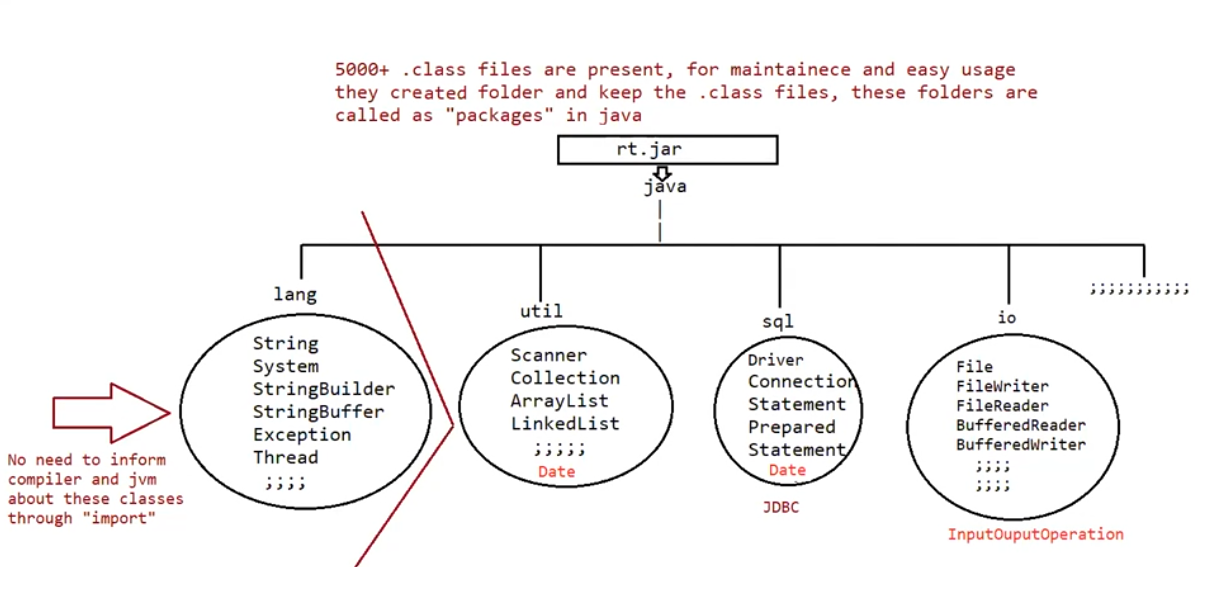
}

The code compiles fine even though we are not using import statements because we used fully qualified name

When we are using fully qualified name it is not required to use import statement

Similarly when we are using import statement it is not required to use fully qualified name





Case 4:

Eg: Ambiguity\_In\_Importing

o/p: compilation error

both class java.sql.Date in java.sql and class java.util.Date in java.util match

Note: Even in the list case also we may get the same ambiguity because it is available in both util and awt packages

Case 5:

While resolving names compiler will always give the importance in the following order

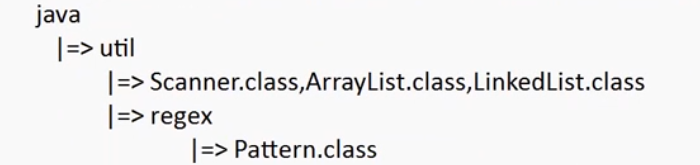
1. Explicit class import
2. Classes present in current working directory
3. Implicit class import

Eg: Import\_Resolving\_Order

The code compiles fine and in this case util package date is considered.

Case 6:

Whenever we are importing a package all the classes and interfaces present in that package are by default available but not sub package classes



To use pattern class in our program directly which import statement is required ?

1. Import java.\*;
2. Import java.util.\*;
3. Import java.util.regex.\*; // valid ( implicit import )
4. Import java.util.regex.Pattern; // valid (explicit import )

Note:

* It refers to only .class files not the subpackages .class files

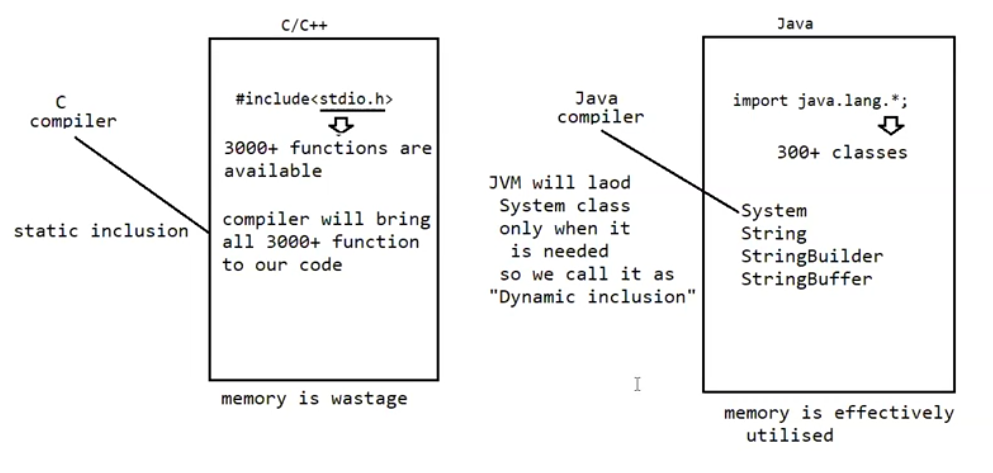
Case 7:

In every java program the following 2 packages are not required to import, because they are available to every java program by default.

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

Case 8:

Import statement is totally compile time concept, if more no of imports are used then more will be the compile time but there is no change in the execution time.



Difference between c language #include and java import?

# include:

1. it can be used in c and c++
2. At compile time only compiler copy the code from standard library and placed in the current program
3. it is static inclusion
4. wastage of memory

import:

1. it can be used in java
2. At runtime jvm will execute the corresponding standard library and use it’s result in our current program
3. It is dynamic inclusion
4. No wastage of memory.

Note:

In case of c language when we use # include all the header files will be loaded at that time. hence it follows static loading

But in java import statement no “.class” file will be loaded after the next lines of import statement, whenever we are using a particular class only that time corresponding “.class” file will be loaded. hence it is dynamic loading (or) load on demand (or) load on fly.

Note: if a class contains all the methods as static then we call that methods as “helper method” (or) “utility methods” and that class is called as “helper class”.

Static import:

This concept is introduced in 1.5 versions. According to sun static improves the readability of the code, but according to world-wide programming experts static import creates the confusion and reduces the readability of the code. Hence if there is no specific requirement never recommended to use a static import.

Usually we can access static members by class name, but whenever we are using static import, it is not required to use class name we can access directly.

Eg: Without\_Static\_Import

// go through the code

Eg: With\_Static\_Import

// go through the code

Eg: Static\_Import\_Eg2

// go through the code

Eg: Static\_Import\_Eg3

// go through the code

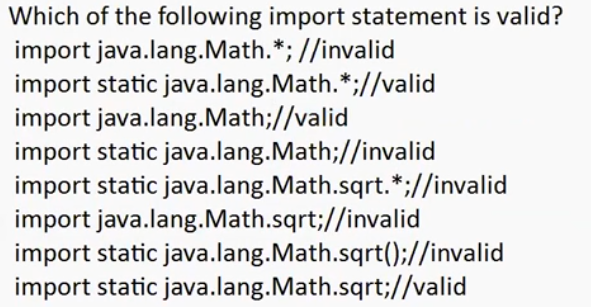
Note:

Two packages containing a class (or) interface with same name is very rare, hence ambiguity problem is very rare in normal import.

But 2 classes (or) interfaces can contain a method (or) a variable with same name is very common, hence ambiguity problem is very common in static import

While resolving static members compiler will give the precedence in the following order

1. Current class static member / import
2. Explicit static import
3. Implicit import



What is the difference between the general import and static import ?

Normal import:

We can use normal imports to import classes and interfaces of a package

Whenever we are using normal import we can access class and interfaces directly by their short names, it is not required to use fully qualified names

Static import:

We can use static import to import particular static members of a class

Whenever we are using static import is not required to use class name, we can access static members directly.