```
Import statement
==============
class Test{
      public static void main(String args[]){
            ArrayList l=new ArrayList();
      }
}
Output:
Compile time error.
D:\Java>javac Test.java
Test.java:3: cannot find symbol
symbol : class ArrayList
location: class Test
ArrayList l=new ArrayList();

=> We can resolve this problem by using fully qualified name "java.util.ArrayList"
      l=new java.util.ArrayList();". But problem with using fully qualified name
every time is it increases length of the code and
      reduces readability.
=> We can resolve this problem by using import statements.

Example:
import java.util.ArrayList;
class Test{
      public static void main(String args[]){
                  ArrayList l=new ArrayList();
      }
}
Output:
D:\Java>javac Test.java
Hence whenever we are using import statement it is not require to use fully
qualified names we can use short names directly.
This approach decreases length of the code and improves readability.


Case 1: 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 ;
      => This type of import is highly recommended to use because it improves
readability of the code.
      => Best suitable for developers where readability is important.

Implicit class import:
Example: import java.util.*;
=> It is never recommended to use because it reduces readability of the code.
=> Best suitable for students where typing is important.


Case 2:
Which of the following import statements are meaningful ?
      a. import java.util;
      b. import java.util.ArrayList.*;
      c. import java.util.*;
      d. import java.util.ArrayList;
```

Answer: c and d

Case3:
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.
=> Whenever we are using fully qualified name it is not required to use import statement.
      Similarly whenever we are using import statements it is not require to use fully qualified name.

Case4:
```
import java.util.*;
import java.sql.*;
class Test{
      public static void main(String args[]) {
            Date d=new Date();
      }
}
```
Output:
Compile time error.
```
D:\Java>javac Test.java
Test.java:7: reference to Date is ambiguous,
both class java.sql.Date in java.sql and class java.util.Date in java.util match

Date d=new Date();
```
Note: Even in the List case also we may get the same ambiguity problem because it is available in both util and awt packages.

Case5:
While resolving class names compiler will always gives the importance in the following order.
1. Explicit class import
2. Classes present in current working directory.
3. Implicit class import.

Example:
```
import java.util.Date;
import java.sql.*;
class Test {
      public static void main(String args[]){
            Date d=new Date();
      }
}
```
The code compiles fine and in this case util package Date will be considered.

Case 6:
Whenever we are importing a package all classes and interfaces present in that package are by default available but not
sub package classes.
```
      java
         |=> util
               |=> Scanner.class,ArrayList.class,LinkedList.class
               |=> regex
                        |=> Pattern.class
```

To use Pattern class in our Program directly which import statement is required ?

```
      a. import java.*;
      b. import java.util.*;
      c. import java.util.regex.*;  //valid(implicit import)
      d. import java.util.regex.Pattern;//valid(explicit import)
```

Note:
* => it refers to only .class files not subpackages .class files

Case7:
In any java Program the following 2 packages are not require to import because these are available by default to every java Program.
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 there then more will be the compile time
 but there is "no change in execution time".

Difference between C language #include and java language import ?
#include
=======
1. It can be used in C & C++
2. At compile time only compiler copy the code from standard library and placed in current program.
3. It is static inclusion
4. wastage of memory
Ex : <jsp:@ file="">

import
======
1. It can be used in Java
2. At runtime JVM will execute the corresponding standard library and use it's result in current program.
3. It is dynamic inclusion
4. No wastage of memory
 Ex : <jsp:include >

Note:
 In the case of C language #include all the header files will be loaded at the time of include statement hence it follows static loading.
 But in java import statement no ".class" will be loaded at the time of import statements in the next lines of the code whenever we are
 using a particular class then only corresponding ".class" file will be loaded. Hence it follows "dynamic loading" or
 "load-on -demand" or "load-on-fly".

JDK 1.5 versions new features :
1.   For-Each
2.   Var-arg

3.   Queue
4.   Generics
5.   Auto boxing and Auto unboxing
6.   Co-varient return types
7.   Annotations
8.   Enum
9.   Static import
10. String builder

Static import:
This concept introduced in 1.5 versions. According to sun static import improves readability of the code but according to
worldwide Programming exports (like us) static imports creates confusion and reduces 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 using class name but whenever we are using static import it is not require to use class name
we can access directly.

Without static import:
```
class Test
{
      public static void main(String args[]){
                  System.out.println(Math.sqrt(4));
                  System.out.println(Math.max(10,20));
                  System.out.println(Math.random());
      }
}
```
Output:
```
D:\Java>javac Test.java
D:\Java>java Test
2.0
20
0.841306154315576
```

With static import:
```
import static java.lang.Math.sqrt;
import static java.lang.Math.*;

class Test{
      public static void main(String args[]){
                  System.out.println(sqrt(4));
                  System.out.println(max(10,20));
                  System.out.println(random());
      }
}
```
Output:
```
D:\Java>javac Test.java
D:\Java>java Test
2.0
20
0.4302853847363891
```

```
class Test{
      static String name = "sachin";
}
```

output

```
======
Test.name.length() ====> 6


import java.io.PrintStream;
class System{
      static PrintStream out;
}

class PrintStream{
      public void println(){
            ;;;;;;
      }
}

System.out.println()
   |        |        |=> It is a method of PrintStream class
   |            |
   |            |=> It is a reference of PrintStream Class
   |=> it is an class
```

Example 3:

```
import static java.lang.System.out;
class Test{
      public static void main(String args[]){
            out.println("hello");
            out.println("hi");
      }
}
```

Output:

```
D:\Java>javac Test.java
D:\Java>java Test
hello
hi
```

Example 4:

```
import static java.lang.Integer.*;
import static java.lang.Byte.*;
class Test{
      public static void main(String args[]){
                  System.out.println(MAX_VALUE);
      }
}
```

Output:
Compile time error.

```
D:\Java>javac Test.java
Test.java:6: reference to MAX_VALUE is ambiguous,
both variable MAX_VALUE in java.lang.Integer and variable MAX_VALUE in
java.lang.Byte match
      System.out.println(MAX_VALUE);
```

Note:
Two packages contain a class or interface with the same is very rare hence
ambiguity problem is very rare in normal import.
But 2 classes or interfaces can contain a method or variable with the same name is
very common hence ambiguity
problem is also very common in static import.

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

order.
1. Current class static member
2. Explicit static import
3. implict static import

eg:
```
import static java.lang.Integer.MAX_VALUE;
import static java.lang.Byte.*;
class Test{
      static int MAX_VALUE  = 999;
      public static void main(String[] args){
            System.out.println(MAX_VALUE);
      }
}
```

Which of the following import statement is valid?
```
 import java.lang.Math.*; //invalid
 import static java.lang.Math.*;//valid
 import java.lang.Math;//valid
 import static java.lang.Math;//invalid
 import static java.lang.Math.sqrt.*;//invalid
 import java.lang.Math.sqrt;//invalid
 import static java.lang.Math.sqrt();//invalid
 import static java.lang.Math.sqrt;//valid
```

Usage of static import reduces readability and creates confusion hence if there is no specific requirement never recommended to use
static import.

What is the difference between 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 name it is not
       require to use fully qualified names.

static import
===========
=> We can use static import to import static members of a particular class.
=> whenever we are using static import it is not require to use class name we can
access static members directly.
```