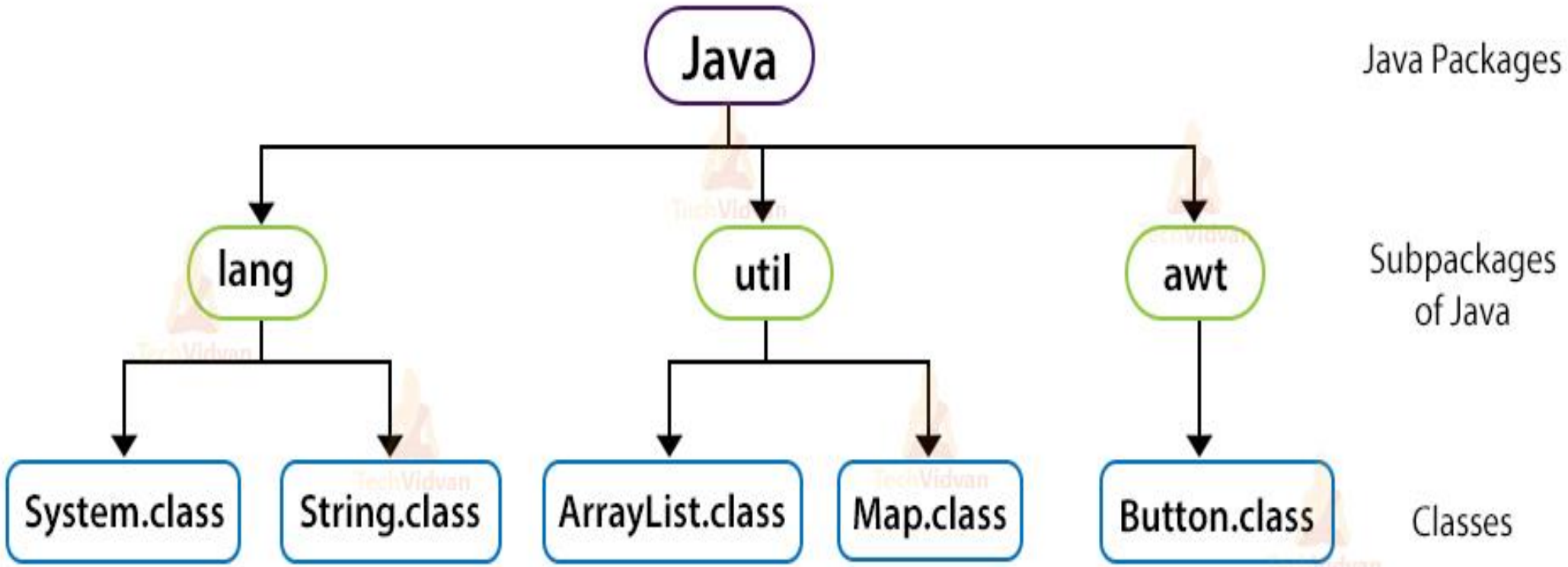
Packages in Java

What is Package in Java?

- **PACKAGE in Java** is a collection of classes, sub-packages, and interfaces.
- It helps organize the classes into a folder structure and make it easy to locate and use them.
- More importantly, it helps improve code reusability.
- Each package in Java has its unique name and organizes its classes and interfaces into a separate namespace, or name group.
- Although interfaces and classes with the same name cannot appear in the same package, they can appear in different packages. This is possible by assigning a separate namespace to each Java package.

Types of Packages in Java

Built-in Packages in Java



Java API packages or built-in packages

- **java.lang:** It contains classes for primitive types, strings, math functions, threads, and exceptions.
- **java.util:** It contains classes such as vectors, hash tables, dates, Calendars,Scanner etc.
- **java.io:** It has stream classes for Input/Output.
- **java.awt:** Classes for implementing Graphical User Interface – windows, buttons, menus, etc.
- **java.net:** Classes for networking
- **java. Applet:** Classes for creating and implementing applets

User-defined packages

- As the name suggests, these packages are defined by the user.
- We create a directory whose name should be the same as the name of the package.
- Then we create a class inside the directory.

Compiling a Java Package

javac -d . Example.java

-d specifies the destination where to locate the generated class file.

You can use any directory name like /home (in case of Linux), C:/folderName (in case of windows), etc.

If you want the package to be present in the same directory, you can use the **dot (.)**

Accessing Packages or Classes from Another Package

- If we want to access all the classes and interfaces of an existing package then we use the **import** statement.
- We can do it in three different ways:

1. import package.*;
2. import package.classname;
3. fully qualified name.

- By using * after the import statement, we can access all the classes of the package but not the sub-packages.
- import packageName.*;

Example MyClass.java

```
package p1; //package
class MyClass
{
	public void printName(String name)
	{
		System.out.println(name);
	}
}
```

MyClass1.java

```
import p1.*; //importing all the classes
public class MyClass1
{
public static void main(String args[])
{
// Initializing the String variable with a value
String name = “CSE";
// Creating an instance of class MyClass from another package.
MyClass obj = new MyClass();
obj.printName(name);
}
}
```

output

```
C:\Users\nhrao\Desktop\packages>javac -d . MyClass.java

C:\Users\nhrao\Desktop\packages>javac MyClass1.java

C:\Users\nhrao\Desktop\packages>java MyClass1
CSE

C:\Users\nhrao\Desktop\packages>
```

Using a Fully qualified name

```
package p1; //package
class MyClass
{
	public void printName(String name)
	{
		System.out.println(name);
	}
}
```

MyClass1.java

```
public class MyClass1
{
public static void main(String args[])
{
// Initializing the String variable with a value
String name = “CSE";
// Creating an instance of class MyClass from another package.
p1.MyClass obj = new p1.MyClass();
obj.printName(name);
}
}
```

output

```
C:\Users\nhrao\Desktop\packages>javac -d . MyClass.java

C:\Users\nhrao\Desktop\packages>javac MyClass1.java

C:\Users\nhrao\Desktop\packages>java MyClass1
CSE

C:\Users\nhrao\Desktop\packages>
```

Calculator Example Using Packages (Calculator.java)

```
package cal;
public class Calculator
{
public int add(int a, int b)
{
return a+b;
}
public int sub(int a, int b)
{
return a-b;
}
public int mul(int a, int b)
{
return a*b;
}
public int div(int a, int b)
{
return a/b;
}
}
```

CalDemo.java

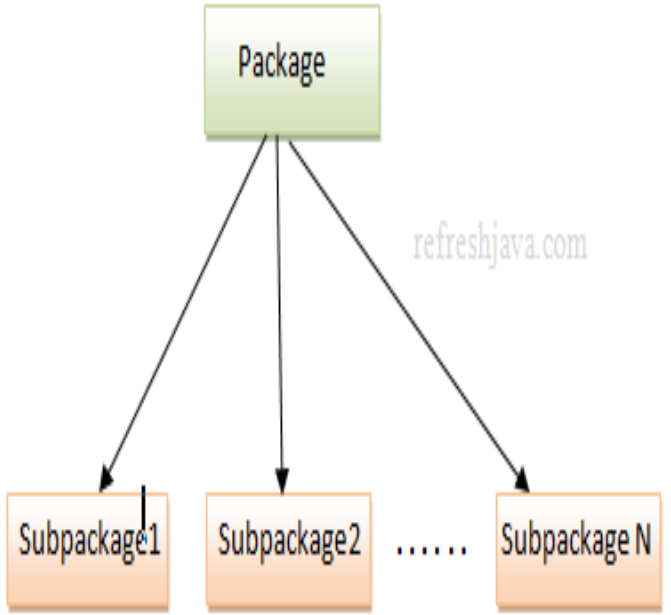
```
import cal.Calculator;
class CalDemo
{
public static void main(String a[])
{
Calculator c=new Calculator();
System.out.println("Sum="+c.add(10,20));
System.out.println("Sub="+c.sub(10,20));
System.out.println("Mul="+c.mul(10,20));
System.out.println("Div="+c.div(10,20));
}
}
```

```
C:\Users\nhrao\Desktop\packages>javac -d . Calculator.java

C:\Users\nhrao\Desktop\packages>javac CalDemo.java

C:\Users\nhrao\Desktop\packages>java CalDemo
Sum=30
Sub=-10
Mul=200
Div=0
```

Subpackage in java

- Package inside the package is called the **subpackage**.
- It should be created **to categorize the package further**.
- **Example:**

```
package LearnJava.corejava;
class Simple{
  public static void main(String args[]){
   System.out.println("Hello from subpackage");
  }
}
```



How to import Sub packages?

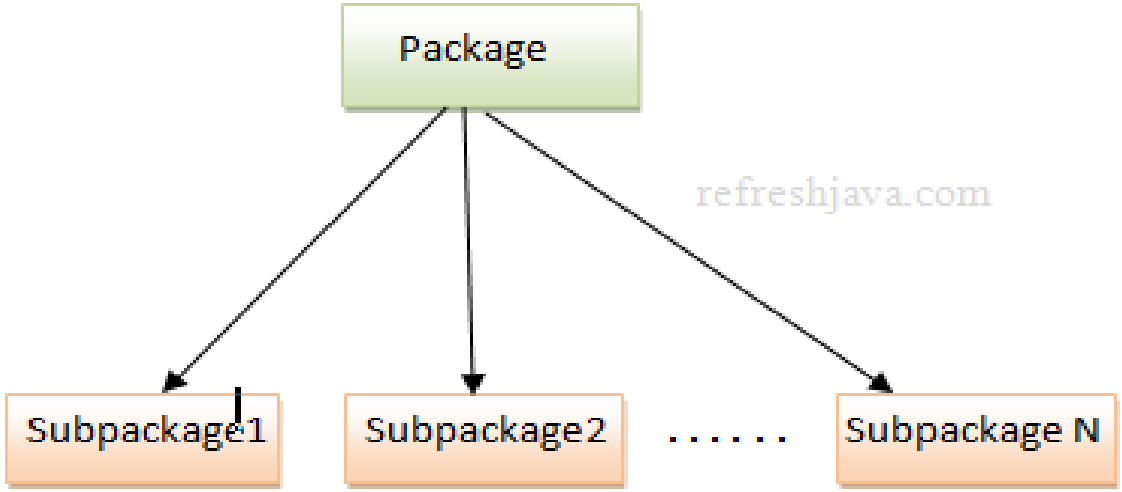
- // To import all classes of a sub package
- import packagename.subpackagename.*;
- // To import specific class of a sub package
- import packagename.subpackagename.classname;

Example

```
package mypack.testpack;


 class MySubPackageProgram {
   public static void main(String args []) {
     System.out.println("My sub package program");
    }
 }
```

Why do we create Sub Packages?

- Subpackages are created to categorize/divide a package further.
- It's similar as creating sub folder inside a folder to categorize it further
- A package may have many sub packages inside it.



How to put two public classes in a package?

- //save as A.java

package rvr;

public class A{}

- //save as B.java

package javatpoint;

public class B{}

Simple Example of static import

```
import static java.lang.System.*;
class StaticImportExample{
  public static void main(String args[]){

    out.println("Hello");//Now no need of System.out
    out.println("Java");

  }
}
```

Output:
Hello
Java

- The static import feature facilitates the java programmer to access any static member of a class directly.
- There is no need to qualify it by the class name.
- **Advantage of static import:**
- Less coding is required if you have access any static member of a class often.
- **Disadvantage of static import**:
- over usage of the static import feature, it makes the program unreadable and unmaintainable.

Access Modifiers in Java

- The access modifiers in Java specifies the accessibility or scope of a field, method, constructor, or class.
- We can change the access level of fields, constructors, methods, and class by applying the access modifier on it.

There are four types of Java access modifiers:

- **Private**: The access level of a private modifier is only within the class. It cannot be accessed from outside the class.
- **Default**: The access level of a default modifier is only within the package. It cannot be accessed from outside the package. If you do not specify any access level, it will be the default.
- **Protected**: The access level of a protected modifier is within the package and outside the package through child class. If you do not make the child class, it cannot be accessed from outside the package.
- **Public**: The access level of a public modifier is everywhere. It can be accessed from within the class, outside the class, within the package and outside the package.

Understanding Java Access Modifiers

Modifier	Class	Package	Subclass	Global
Public	Yes	Yes	Yes	Yes
Protected	Yes	Yes	Yes	No
Default	Yes	Yes	No	No
Private	Yes	No	No	No

The private access modifier is accessible only within the class.

```
class A{
private int data=40;
private void msg(){System.out.println("Hello java");}
}

public class Simple{
 public static void main(String args[]){
   A obj=new A();
   System.out.println(obj.data);//Compile Time Error
   obj.msg();//Compile Time Error
   }
}
```

Private Constructor

```
class A{
private A(){}//private constructor
void msg(){System.out.println("Hello java");}
}
public class Simple{
 public static void main(String args[]){
   A obj=new A();//Compile Time Error
 }
}
```

Default

- If you don't use any modifier, it is treated as **default** by default.
- The default modifier is accessible only within package. It cannot be accessed from outside the package.
- It provides more accessibility than private. But, it is more restrictive than protected, and public.

A.java

```
package pack;
class A{
 void msg()
{
System.out.println("Hello");
}
}
```

B.java

```
package mypack;
import pack.*;
class B{
  public static void main(String args[])
{
   A obj = new A();//Compile Time Error
   obj.msg();//Compile Time Error
  }
}
```

The scope of class A and its method msg() is default so it cannot be accessed from outside the package.

Protected

- The **protected access modifier** is accessible within package and outside the package but through inheritance only.
- The protected access modifier can be applied on the data member, method and constructor. It can't be applied on the class.
- It provides more accessibility than the default modifer.

A.java

```
package pack;
public class A{
protected void msg()
{
System.out.println("Hello");
}
}
```

Output:Hello

B.java

```
package mypack;
import pack.*;
class B extends A{
  public static void main(String args[]){
   B obj = new B();
   obj.msg();
  }
}
```

Hello

Public

- The **public access modifier** is accessible everywhere.
- It has the widest scope among all other modifiers.

A.java

```
package pack;
public class A{
public void msg()
{
System.out.println("Hello");
}
}
```

B.java

```
package mypack;
import pack.*;

class B{
  public static void main(String args[]){
   A obj = new A();
   obj.msg();
  }
}
```

Hello

Java Access Modifiers with Method Overriding

```
class A{
protected void msg()
{
System.out.println("Hello java");}
}
public class Simple extends A
{
void msg(){System.out.println("Hello java");}//C.T.Error
 public static void main(String args[]){
   Simple obj=new Simple();
   obj.msg();
   }
}
```

If you are overriding any method, overridden method (i.e. declared in subclass) must not be more restrictive.