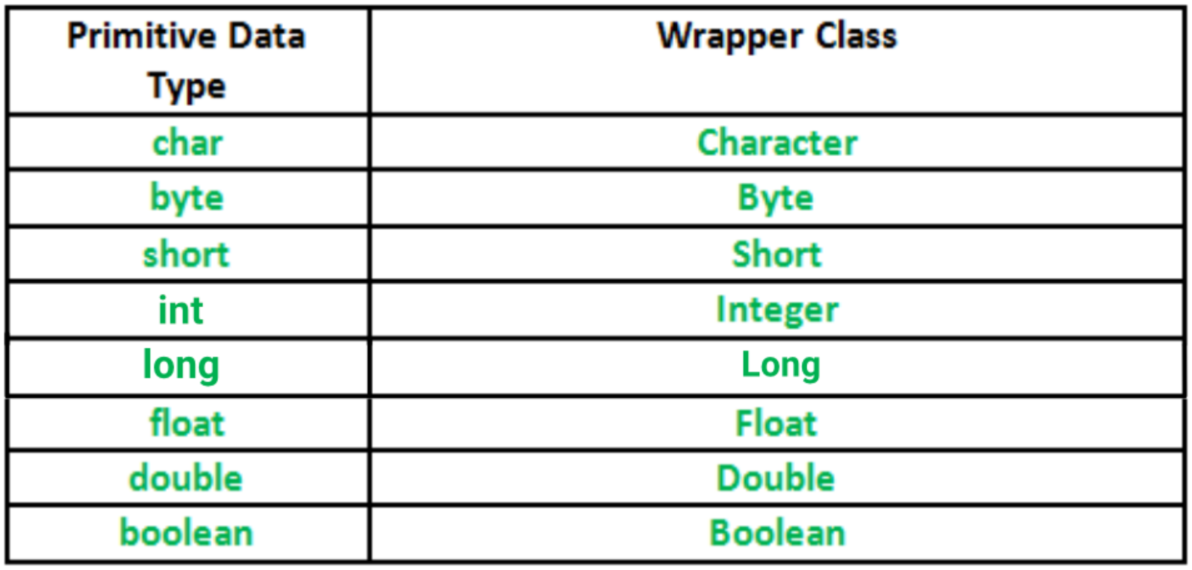
Wrapper Classes in Java

A Wrapper class is a class whose object wraps or contains primitive data types., we can wrap a primitive value into a wrapper class object..



Primitive Data Type	Wrapper Class
char	Character
byte	Byte
short	Short
int	Integer
long	Long
float	Float
double	Double
boolean	Boolean

Autoboxing: Automatic conversion of primitive types to the object of their corresponding wrapper classes is known as autoboxing. For example – conversion of int to Integer, long to Long, double to Double etc.
Example:

```
class Autoboxing
{
    public static void main(String[] args)
    {
       int i=100;
        Integer obj = Integer.valueOf(i); //i is primitive obj is object
        System.out.println(obj);


         Integer obj2 = i;      // Automatically conversion
        System.out.println(obj2);     //Autoboxing


    }
}
```

OUTPUT:

100

100

```
class Autoboxing
{
    public static void main(String[] args)
    {
        int i=100;
       Integer j = Integer.valueOf(i); //I is primitive j is object
        System.out.println(j);

        Integer ab = i;
        System.out.println(ab);     //Autoboxing

    }
}
```

Unboxing: It is just the reverse process of autoboxing. Automatically converting an object of a wrapper class to its corresponding primitive type is known as unboxing. For example – conversion of Integer to int, Long to long, Double to double, etc.

```
class Unboxing
{
    public static void main(String[] args)
    {

       Integer i = new Integer(10); // object

        int ab = i.intValue();
        System.out.println(ab);

         int j=i; //                      unboxing
        System.out.println(j);

    }
}
```

Java Packages

A **java package** is a group of similar types of classes, interfaces and sub-packages.

Package in java can be categorized in two form, built-in package and user-defined package.

There are many built-in packages such as java, lang, awt, javax, swing, net, io, util, sql etc.

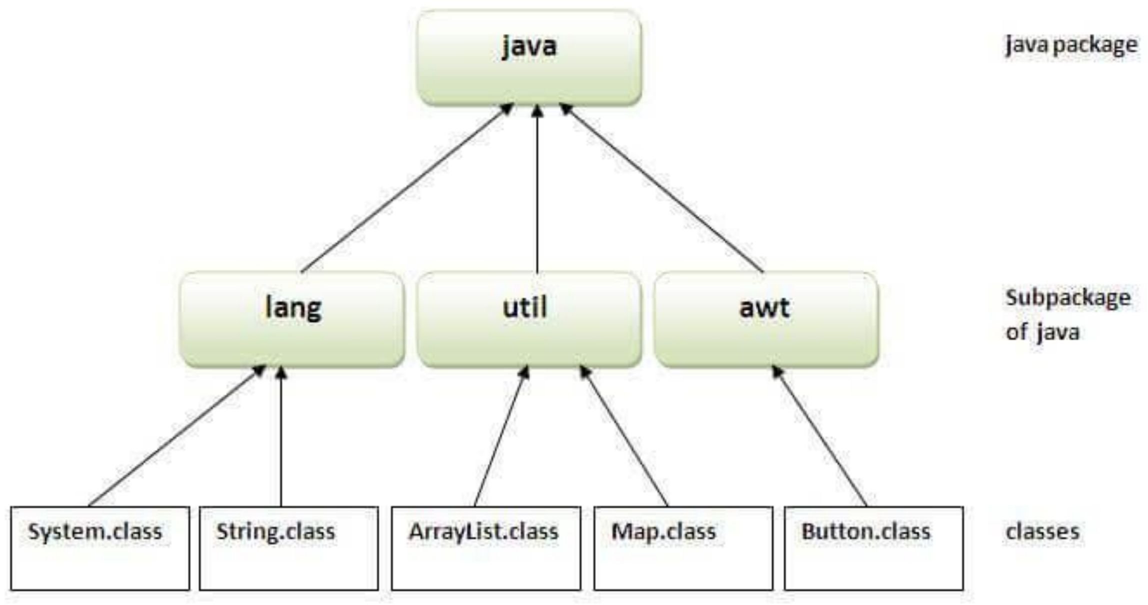
Here, we will have the detailed learning of creating and using user-defined packages.

Advantage of Java Package

1) Java package is used to categorize the classes and interfaces so that they can be easily maintained.

2) Java package provides access protection.

3) Java package removes naming collision.



- Built-in Packages (packages from the Java API)
- User-defined Packages (create your own packages)

How to access package from another package?

There are three ways to access the package from outside the package.

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

*1) Using packagename.**

If you use package.* then all the classes and interfaces of this package will be accessible but not subpackages.

The import keyword is used to make the classes and interface of another package accessible to the current package.

Example of package that import the packagename.*

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

```
package mypack;
import pack.*;

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

```
Output:Hello
```

2) Using packagename.classname

If you import package.classname then only declared class of this package will be accessible.

Example of package by import package.classname

```
//save by A.java

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

```
//save by B.java
package mypack;
import pack.A;

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

```
Output:Hello
```

3) Using fully qualified name

If you use fully qualified name then only declared class of this package will be accessible. Now there is no need to import. But you need to use fully qualified name every time when you are accessing the class or interface.

It is generally used when two packages have same class name e.g. java.util and java.sql packages contain Date class.

Example of package by import fully qualified name

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

```
//save by B.java
package mypack;
class B{
  public static void main(String args[]){
   pack.A obj = new pack.A();//using fully qualified name
   obj.msg();
  }
}
```

```
Output:Hello
```

If you import a package, all the classes and interface of that package will be imported excluding the classes and interfaces of the subpackages. Hence, you need to import the subpackage as well.

Subpackage in java

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

Built-in Packages

The Java API is a library of prewritten classes, that are free to use, included in the Java Development Environment.

The library is divided into **packages** and **classes**. Meaning you can either import a single class (along with its methods and attributes), or a whole package that contain all the classes that belong to the specified package.

To use a class or a package from the library, you need to use the `import` keyword:

Syntax

```
import package.name.Class;   // Import a single class
import package.name.*;   // Import the whole package
```

Import a Class

If you find a class you want to use, for example, the `Scanner` class, **which is used to get user input**, write the following code:

Example

```
import java.util.Scanner;
```

User-defined Packages

To create your own package, you need to understand that Java uses a file system directory to store them. Just like folders on your computer:

To create a package, use the `package` keyword:

```
package mypack;
class MyPackageClass {
  public static void main(String[] args) {
    System.out.println("This is my package!");
  }
}
```