1. Java Package

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.

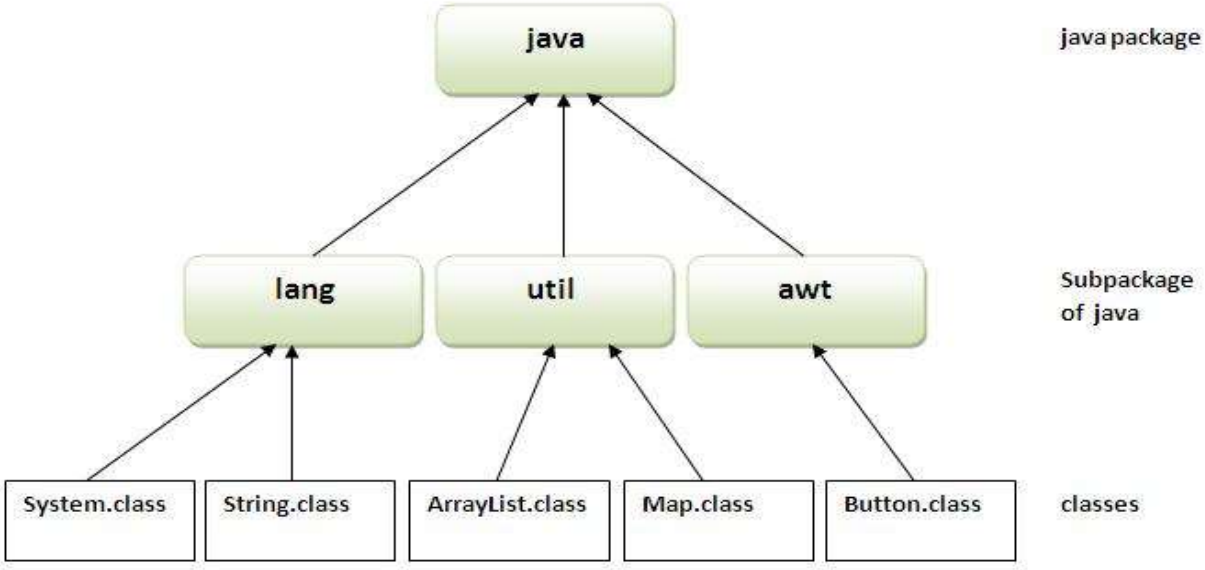
There are many built-in packages such as java, lang, awt, javax, swing, net, io, util, sql etc.

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.



Simple example of java package

The **package keyword** is used to create a package in java.

```
//save as Simple.java
```

```
package mypack;

public class Simple{

 public static void main(String args[]){

    System.out.println("Welcome to package");

   }

}
```

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.*

```
//save by A.java

package pack;

public class A{

  public void msg(){System.out.println("Hello");}

}

//save by B.java

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

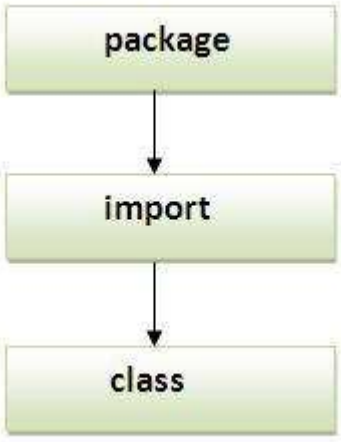
```
//save by A.java


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
```

Note: If you import a package, subpackages will not be imported.

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.

Note: Sequence of the program must be package then import then class.



Subpackage in java

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

The standard of defining package is domain.company.package e.g. com.javapoint.bean or org.sssit.dao.

2. Access Modifiers in java

There are two types of modifiers in java: **access modifiers** and **non-access modifiers**.

The access modifiers in java specifies accessibility (scope) of a data member, method, constructor or class.

There are 4 types of java access modifiers:

1. private
2. default
3. protected
4. public

There are many non-access modifiers such as static, abstract, synchronized, native, volatile, transient etc. Here, we will learn access modifiers.

---

1) private access modifier

The private access modifier is accessible only within class.

Simple example of private access modifier

In this example, we have created two classes A and Simple. A class contains private data member and private method. We are accessing these private members from outside the class, so there is compile time error.

```
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
   }
}
```

Role of Private Constructor

If you make any class constructor private, you cannot create the instance of that class from outside the class. For example:

```
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
 }
}
```

Note: A class cannot be private or protected except nested class.

2) default access modifier

If you don't use any modifier, it is treated as **default** bydefault. The default modifier is accessible only within package.

Example of default access modifier

In this example, we have created two packages pack and mypack. We are accessing the A class from outside its package, since A class is not public, so it cannot be accessed from outside the package.

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

```
  }
}
```

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

---

3) protected access modifier

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.

Example of protected access modifier

In this example, we have created the two packages pack and mypack. The A class of pack package is public, so can be accessed from outside the package. But msg method of this package is declared as protected, so it can be accessed from outside the class only through inheritance.

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

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

```
}
```

```
Output:Hello
```

---

4) public access modifier

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

Example of public access modifier

```
//save by A.java

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

package mypack;
import pack.*;

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

```
Output:Hello
```

Understanding all java access modifiers

Let's understand the access modifiers by a simple table.

Access Modifier	within class	within package	outside package by subclass only	outside package
Private	Y	N	N	N
Default	Y	Y	N	N
Protected	Y	Y	Y	N
Public	Y	Y	Y	Y

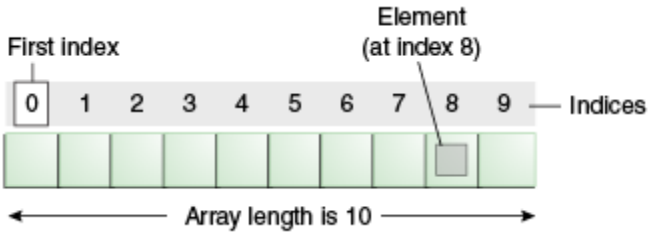
---

3. Java Array

Normally, array is a collection of similar type of elements that have contiguous memory location.

Java array is an object the contains elements of similar data type. It is a data structure where we store similar elements. We can store only fixed set of elements in a java array.

Array in java is index based, first element of the array is stored at 0 index.



Advantage of Java Array

- **Code Optimization:** It makes the code optimized, we can retrieve or sort the data easily.
- **Random access:** We can get any data located at any index position.

Disadvantage of Java Array

- **Size Limit:** We can store only fixed size of elements in the array. It doesn't grow its size at runtime. To solve this problem, collection framework is used in java.

Single Dimensional Array in java

Syntax to Declare an Array in java

1. dataType[] arr; (or)
2. dataType []arr; (or)
3. dataType arr[];

Instantiation of an Array in java

1. arrayRefVar=**new** datatype[size];

Example of single dimensional java array

Let's see the simple example of java array, where we are going to declare, instantiate, initialize and traverse an array.

```
class Testarray{

public static void main(String args[]){


int a[]=new int[5];//declaration and instantiation

a[0]=10;//initialization

a[1]=20;

a[2]=70;

a[3]=40;
```

```
a[4]=50;


//printing array
for(int i=0;i<a.length;i++)//length is the property of array
System.out.println(a[i]);


}}
```

```
Output: 10
        20
        70
        40
        50
```

Declaration, Instantiation and Initialization of Java Array

We can declare, instantiate and initialize the java array together by:

1. `int a[]={33,3,4,5};//declaration, instantiation and initialization`

Let's see the simple example to print this array.

```
class Testarray1{
public static void main(String args[]){


int a[]={33,3,4,5,43,345};//declaration, instantiation and initiali
zation
```

```
//printing array
for(int i=0;i<a.length;i++)//length is the property of array
System.out.println(a[i]);


}}
```

```
Output:33
       3
       4
       5
```

Passing Array to method in java

We can pass the java array to method so that we can reuse the same logic on any array.

Let's see the simple example to get minimum number of an array using method.

```
class Testarray2{
        static void min(int arr[]){
                for(int i=0;i<arr.length;i++)  {
                System.out.println(arr[i]);
         }
        }
        public static void main(String args[]){
        int a[]={33,3,4,5};
        min(a);//passing array to method

}}
```

4. Java String

Java String provides a lot of concepts that can be performed on a string such as compare, concat, equals, split, length, replace, compareTo etc.

In java, string is basically an object that represents sequence of char values.

An array of characters works same as java string. For example:

```
1. char[] ch={'j','a','v','a','p','o','i','n','t'};
2. String s=new String(ch);
```

is same as:

```
1. String s="javapoint";
```

What is String in java

Generally, string is a sequence of characters. But in java, string is an object that represents a sequence of characters. String class is used to create string object.

How to create String object?

There are two ways to create String object:

1. By string literal
2. By new keyword

---

1) String Literal

Java String literal is created by using double quotes. For Example:

```
1. String s="welcome";
```

---

2) By new keyword

```
1. String s=new String("Welcome");
```

Java String Example

```
public class StringExample{

public static void main(String args[]){

String s1="java";//creating string by java string literal


char ch[]={'s','t','r','i','n','g','s'};

String s2=new String(ch);//converting char array to string


String s3=new String("example");//creating java string by new keyword


System.out.println(s1);

System.out.println(s2);

System.out.println(s3);

}}
```

```
java
strings
```

example

Length:

```
public class LengthExample{

public static void main(String args[]){

String s1="javatpoint";

String s2="python";
```

```
System.out.println("string length is: "+s1.length());//10 is the length of java tpoint string

System.out.println("string length is: "+s2.length());//6 is the length of pyth on string

}}
```

```
string length is: 10
```

string length is: 6

Java String equals

The **java string equals()** method compares the two given strings based on the content of the string. If any character is not matched, it returns false. If all characters are matched, it returns true.

The String equals() method overrides the equals() method of Object class.

```
public class EqualsExample{

public static void main(String args[]){

String s1="javatpoint";

String s2="javatpoint";

String s3="jAVATPOINT";

String s4="python";

System.out.println(s1.equals(s2));//true because content and case is same

System.out.println(s1.equals(s3));//false because case is not same

System.out.println(s1.equals(s4));//false because content is not same

}}
```

```
true
```

```
false
```

false

a=97

A=65

Java String isEmpty

The **java string isEmpty()** method checks if this string is empty. It returns *true*, if length of string is 0 otherwise *false*.

```
public class IsEmptyExample{

public static void main(String args[]){

String s1="";

String s2="javatpoint";


System.out.println(s1.isEmpty());

System.out.println(s2.isEmpty());

}}
```

```
true
```

false

Java String concat

The **java string concat()** method combines specified string at the end of this string.

Java String concat() method example

```
public class ConcatExample{

public static void main(String args[]){

String s1="java string";

s1.concat("is immutable");

System.out.println(s1);

s1=s1.concat(" is immutable so assign it explicitly");

System.out.println(s1);

}}
```

```
java string
```

java string is immutable so assign it explicitly

Java String replace

The **java string replace()** method returns a string replacing all the old char or CharSequence to new char or CharSequence.

Signature

There are two type of replace methods in java string.

1. **public** String replace(**char** oldChar, **char** newChar)
2. and
3. **public** String replace(CharSequence target, CharSequence replacement)

Parameters

oldChar : old character

newChar : new character

target : target sequence of characters

replacement : replacement sequence of characters

Returns

replaced string

Java String replace(char old, char new) method example

```
public class ReplaceExample1{

public static void main(String args[]){

String s1="java is a very good";

String replaceString=s1.replace('a','e');//replaces all occurrences of 'a' to 'e'

System.out.println(replaceString);

}}
```

```
jeve is e very good
```

Java String replace(CharSequence target, CharSequence replacement) method example

```
public class ReplaceExample2{

public static void main(String args[]){

String s1="my name is khan my name is java";

String replaceString=s1.replace("is","was");//replaces all occurrences of "is" to "was"

System.out.println(replaceString);

}}
```

```
my name was khan my name was java
```

Java String substring

The **java string substring()** method returns a part of the string.

We pass begin index and end index number position in the java substring method where start index is inclusive and end index is exclusive. In other words, start index starts from 0 whereas end index starts from 1.

There are two types of substring methods in java string.

Signature

```
public String substring(int startIndex)
```

and

```
public String substring(int startIndex, int endIndex)
```

If you don't specify endIndex, java substring() method will return all the characters from startIndex.

Parameters

startIndex : starting index is inclusive

endIndex : ending index is exclusive

Returns

specified string

Java String substring() method example

```
public class SubstringExample{

public static void main(String args[]){

String s1="javatpoint";

System.out.println(s1.substring(2,4));//returns va

System.out.println(s1.substring(2));//returns vatpoint

}}
```

```
va
```

vatpoint

Java String toLowerCase()

The **java string toLowerCase()** method returns the string in lowercase letter. In other words, it converts all characters of the string into lower case letter.

The toLowerCase() method works same as toLowerCase(Locale.getDefault()) method. It internally uses the default locale.

Signature

There are two variant of toLowerCase() method. The signature or syntax of string toLowerCase() method is given below:

copy to clipboard

1. **public** String toLowerCase()
2. **public** String toLowerCase(Locale locale)

The second method variant of toLowerCase(), converts all the characters into lowercase using the rules of given Locale.

Returns

string in lowercase letter.

Java String toLowerCase() method example

```
1. public class StringLowerExample{
2. public static void main(String args[]){
3. String s1="JAVATPOINT HELLO stRIng";
4. String s1lower=s1.toLowerCase();
5. System.out.println(s1lower);
6. }}
```

Output:

```
javatpoint hello string
```

Java String toUpperCase()

The **java string toUpperCase()** method returns the string in uppercase letter. In other words, it converts all characters of the string into upper case letter.

The toUpperCase() method works same as toUpperCase(Locale.getDefault()) method. It internally uses the default locale.

Signature

There are two variant of toUpperCase() method. The signature or syntax of string toUpperCase() method is given below:

```
public String toUpperCase()
public String toUpperCase(Locale locale)
```

The second method variant of toUpperCase(), converts all the characters into uppercase using the rules of given Locale.

Returns

string in uppercase letter.

Java String toUpperCase() method example

```
public class StringUpperExample{

public static void main(String args[]){

String s1="hello string";

String s1upper=s1.toUpperCase();

System.out.println(s1upper);

}}
```

Output:

```
HELLO STRING
```

Java String trim

The **java string trim()** method eliminates leading and trailing spaces.

The string trim() method doesn't omits middle spaces.

Signature

The signature or syntax of string trim method is given below:

1. **public** String trim()

Returns

string with omitted leading and trailing spaces

Java String trim() method example

```
public class StringTrimExample{

public static void main(String args[]){

String s1="  hello string   ";

System.out.println(s1+"javatpoint");//without trim()

System.out.println(s1.trim()+"javatpoint");//with trim()

}}
```

```
  hello string   javatpoint
```

hello stringjavatpoint