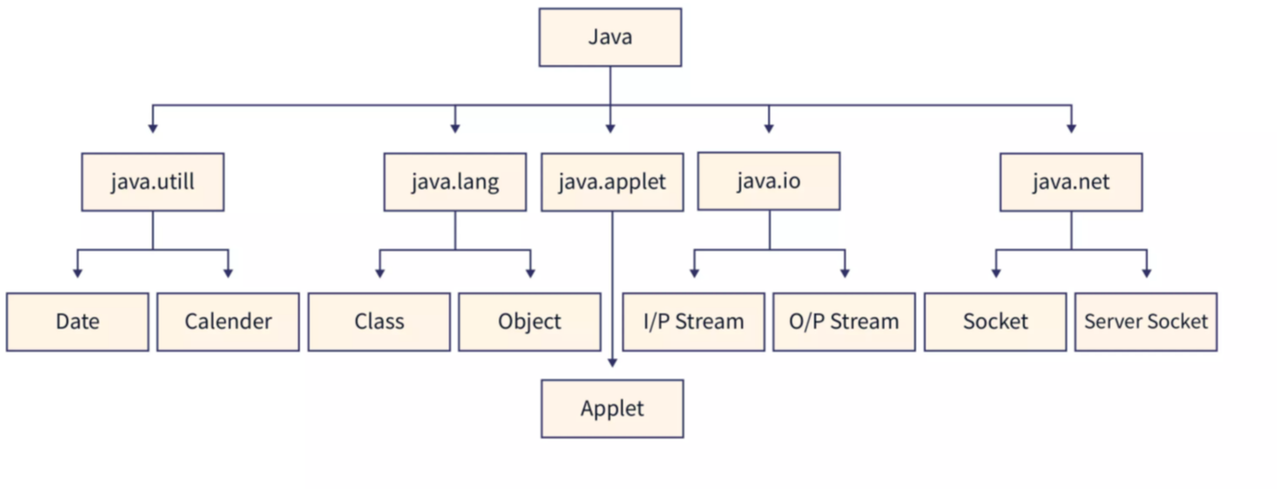
The java.util package in Java is a built-in package that contains various utility classes and interfaces. It provides basic functionality for commonly occurring use cases. It contains Java's collections framework, date and time utilities, string-tokenizer, event-model utilities, etc.

Introduction to java.util Package in Java

To support the needs of a software developer, Java provides various built-in and pre-written functionalities in the form of packages. These built-in packages contain various kinds of classes and interfaces that can be used to develop better and maintainable code. Some of the built-in Java packages are shown in the figure given below:



One of the most important Java packages is the java.util package. The java.util package contains several pre-written classes that can act as a base to solve commonly occurring problems in software development. It provides basic and essential source code snippets to the programmers that can lead to clean and maintainable code. It contains various classes and methods that perform common tasks such as **string tokenization**, **date and time formatting**, Java Collections implementation, etc.

What does import java.util do in Java?

To load the basic utility classes and interfaces provided by the java.util package, we need to use Java's **import** keyword. The **import** keyword is used to access and load the package and its respective classes into the Java program. The syntax to import the package or its classes is given below:

```
import package_name.class_name;
import package_name.*;
```

Here, the first import statement is used to load a certain class from the specified package, whereas the second statement is used to import the whole package into the Java program. For the **java.util** package:

```
import java.util.*;
```

The above statement is used to load all the functionalities provided by the java.util package.

java.util Package Import Statement Example

Let's look at some examples to understand the use of **import** statements while loading the java.util package:

Example 1: To Load the Whole Util Package

```
import java.util.*;
```

As discussed, this statement will load all the pre-written classes and interfaces provided by the java.util package.

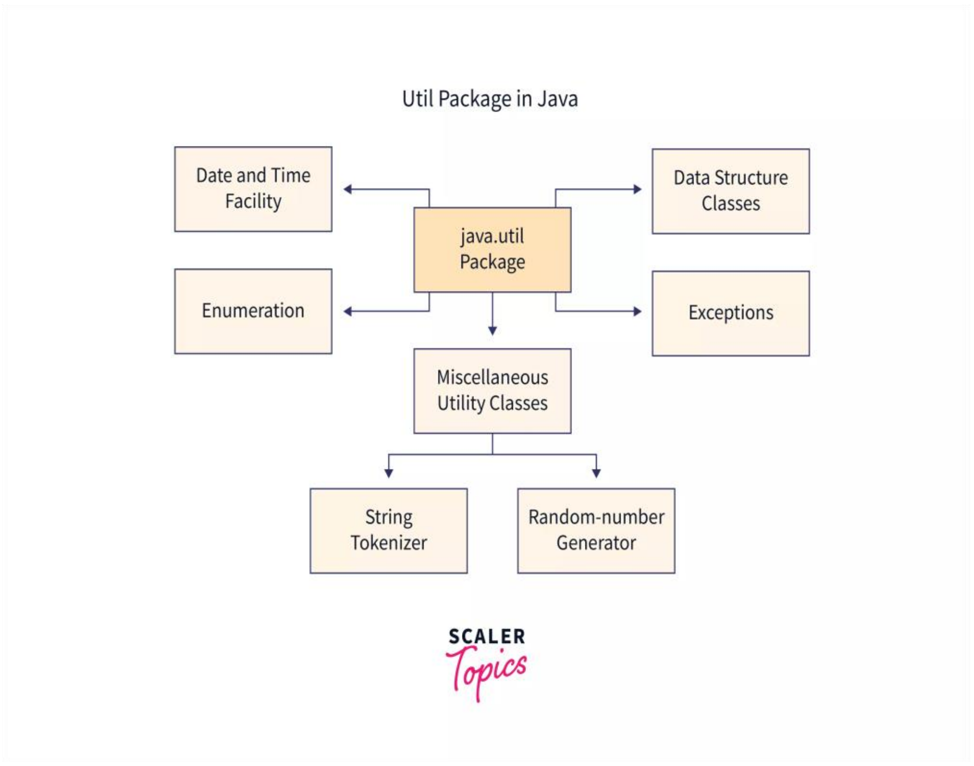
Example 2: To Load a Specific Class from the java.util Package

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

The above statement is importing the **Scanner** class available in the **java.util** package. This class is widely used to take input from the user in a Java program.

What is the Use of java.util Package in Java?

The java.util package in Java consists of several components that provide basic necessities to the programmer. These components include:



Let's explore these components and their usage in Java:

- **Data Structure Classes:** The **java.util** package contains several pre-written data structures like Dictionary, Stack, LinkedList, etc. that can be used directly in the program using the **import** statements.
- **Date and Time Facility:** The **java.util** package provides several date and time-related classes that can be used to create and manipulate date and time values in a program.
- **Enumeration:** It provides a special interface known as enumeration(enum for short) that can be used to iterate through a set of values.
- **Exceptions:** It provides classes to handle commonly occurring exceptions in a Java program.
- **Miscellaneous Utility Classes:** The **java.util** package contains essential utilities such as the string tokenizer and random-number generator.

What is Collections Framework in Java?

Collections Framework in Java is a special part of the **java.util** package that can be used to represent and manipulate collections. It provides easy storage and organization of a group of objects (or collection). It includes pre-written classes, interfaces, and algorithms under a unified architecture.

java.util Package Interfaces

Interface	Description
Collections	Root interface for the Collections API
Comparator	Provides sorting logic
Deque	Implements Deque data structure
Enumeration	Produces enumeration objects
Iterator	Provides iteration logic
List	Implements an ordered collection (Sequence)
Map	Implements Map data structure (key-value pairs)
Queue	Implements Queue data structure
Set	Produces a collection that contains no duplicate elements
SortedSet	Produces a set that remembers the total ordering

java.util Package Classes

Class	Description
Collections	Provides methods to represent and manage collections
Formatter	An interpreter for format strings
Scanner	Text scanner used to take user inputs
Arrays	Provides methods for array manipulation
LinkedList	Implements Doubly-linked list data structure
HashMap	Implements Map data structure using Hash tables
TreeMap	Implements Red-Black tree data structure
Stack	Implements last-in-first-out (LIFO) data structure
PriorityQueue	Implements unbounded priority queue data structure
Date	Represents a specific instance of time
Calendar	Provides methods to manipulate calendar fields
Random	Used to generate a stream of pseudorandom numbers.
StringTokenizer	Used to break a string into tokens
Timer, TimerTask	Used by threads to schedule tasks
UUID	Represents an immutable universally unique identifier

java.util Package Enum

Enum Class	Description
Formatter.BigDecimalLayoutForm	Used for BigDecimal formatting
Locale.Category	Used for locale categories
Locale.FilteringMode	Provides constants for filtering mode selection
Locale.IsoCountryCode	Used to specify the type defined in ISO 3166

java.util Package Exceptions

Exception Class	Description
ConcurrentModificationException	When an object is modified concurrently without permission
EmptyStackException	Indicates that the Stack is empty
InputMismatchException	When the user does not provide valid input
MissingResourceException	Indicates absence of a resource

NoSuchElementException	Indicates absence of requested element
PatternSyntaxException	Indicates syntax error in a regular-expression pattern
IllegalFormatException	Indicates that a format string contains illegal syntax

Some classes example program:-

1.Date Class:-

The class Date represents a specific instant in time, with millisecond precision. The Date class of java.util package implements Serializable, Cloneable and Comparable interface. It provides constructors and methods to deal with date and time with java.

Constructors

- **Date()** : Creates date object representing current date and time.
- **Date(long milliseconds)** : Creates a date object for the given milliseconds since January 1, 1970, 00:00:00 GMT.

  EX:-

```
import java.util.*;

public class Main
{
    public static void main(String[] args)
    {
        Date d1 = new Date();
        System.out.println("Current date is " + d1);
        Date d2 = new Date(2323223232L);
        System.out.println("Date represented is "+ d2 );
    }
}
```

Output:

```
Current date is Fri Feb 09 14:51:48 IST 2024

Date represented is Wed Jan 28 02:50:23 IST 1970
```

2.Random Class:-

- For using this class to generate random numbers, we have to first create an instance of this class and then invoke methods such as nextInt(), nextDouble(), nextLong() etc using that instance.
- We can generate random numbers of types integers, float, double, long, booleans using this class.
- We can pass arguments to the methods for placing an upper bound on the range of the numbers to be generated. For example, nextInt(6) will generate numbers in the range 0 to 5 both inclusive.

  Ex:-

```
import java.util.Random;

 class generateRandom{

    public static void main(String args[])
```

```
        {
            // create instance of Random class
            Random rand = new Random();

            // Generate random integers in range 0 to 999
            int rand_int1 = rand.nextInt(1000);
            int rand_int2 = rand.nextInt(1000);

            // Print random integers
            System.out.println("Random Integers: "+rand_int1);
            System.out.println("Random Integers: "+rand_int2);

            // Generate Random doubles
            double rand_dub1 = rand.nextDouble();
            double rand_dub2 = rand.nextDouble();

            // Print random doubles
            System.out.println("Random Doubles: "+rand_dub1);
            System.out.println("Random Doubles: "+rand_dub2);
        }
    }
```

OutPut1:-

```
Random Integers: 314

Random Integers: 615

Random Doubles: 0.42191614153506474

Random Doubles: 0.5244257318738437
```

OutPut1:-

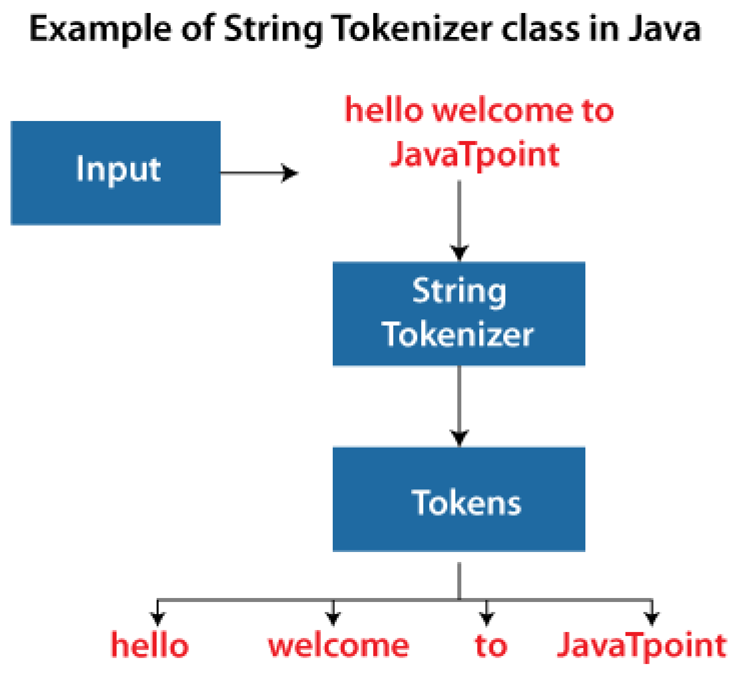
```
Random Integers: 528

Random Integers: 524

Random Doubles: 0.8659438525285449

Random Doubles: 0.7720315452520895
```

3.StringTokenizer

The **java.util.StringTokenizer** class allows you to break a String into tokens. It is simple way to break a String. It is a legacy class of Java.

It doesn't provide the facility to differentiate numbers, quoted strings, identifiers etc. like StreamTokenizer class. We will discuss about the StreamTokenizer class in I/O chapter.

In the StringTokenizer class, the delimiters can be provided at the time of creation or one by one to the tokens.


Example of String Tokenizer class in Java
Input
hello welcome to JavaTpoint
String Tokenizer
Tokens
hello
welcome
to
JavaTpoint

Constructors of the StringTokenizer Class

There are 3 constructors defined in the StringTokenizer class.

Constructor	Description
StringTokenizer(String str)	It creates StringTokenizer with specified string.
StringTokenizer(String str, String delim)	It creates StringTokenizer with specified string and delimiter.
StringTokenizer(String str, String delim, boolean returnValue)	It creates StringTokenizer with specified string, delimiter and returnValue. If return value is true, delimiter characters are considered to be tokens. If it is false, delimiter characters serve to separate tokens.

Methods of the StringTokenizer Class

The six useful methods of the StringTokenizer class are as follows:

Methods	Description
boolean hasMoreTokens()	It checks if there is more tokens available.
String nextToken()	It returns the next token from the StringTokenizer object.

String nextToken(String delim)	It returns the next token based on the delimiter.
boolean hasMoreElements()	It is the same as hasMoreTokens() method.
Object nextElement()	It is the same as nextToken() but its return type is Object.
int countTokens()	It returns the total number of tokens.

Example of StringTokenizer Class

Let's see an example of the StringTokenizer class that tokenizes a string "my name is khan" on the basis of whitespace.

Simple.java

```java
import java.util.StringTokenizer;
public class Simple{
 public static void main(String args[]){
   StringTokenizer st = new StringTokenizer("my name is khan"," ");
     while (st.hasMoreTokens()) {
         System.out.println(st.nextToken());
     }
   }
}
```

Output:

```
my
name
is
khan
```

The above Java code, demonstrates the use of StringTokenizer class and its methods hasMoreTokens() and nextToken()

4.Enums

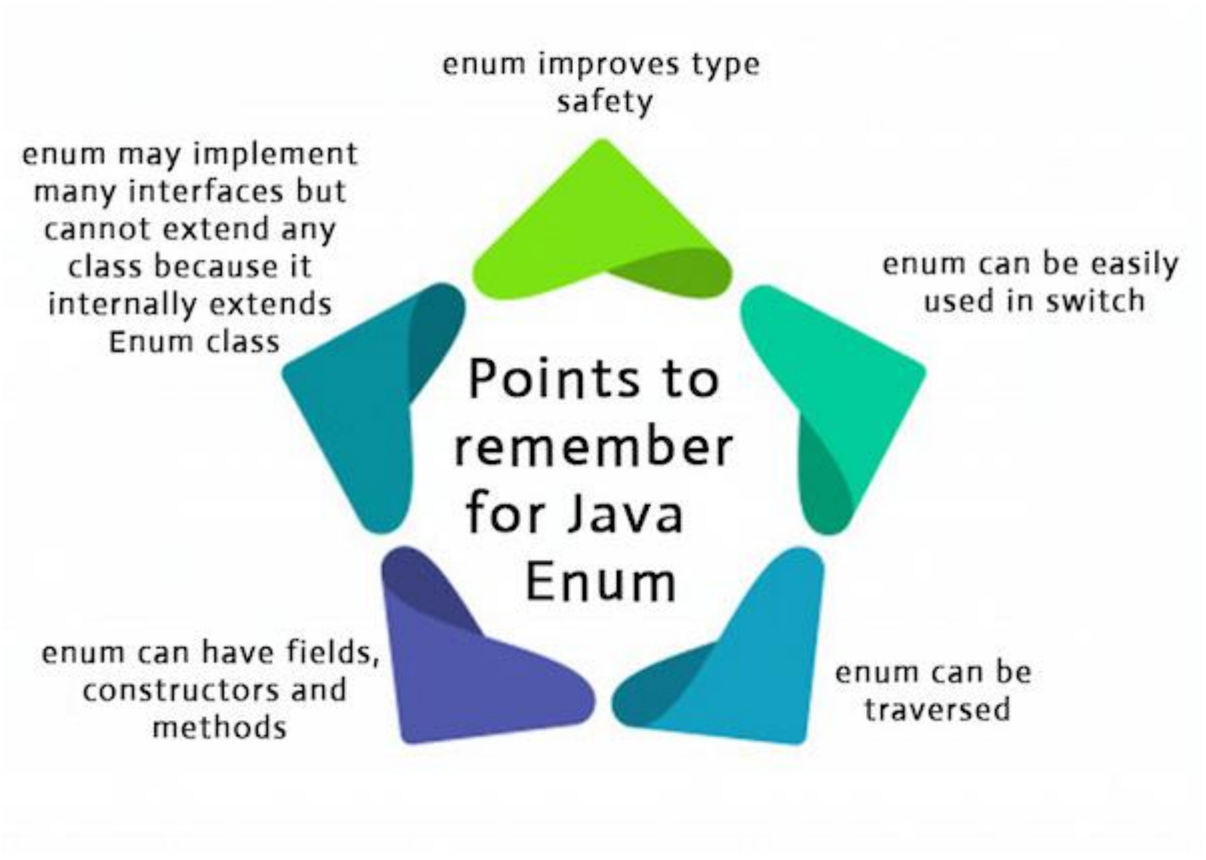
The **Enum in Java** is a data type which contains a fixed set of constants.

It can be used for days of the week (SUNDAY, MONDAY, TUESDAY, WEDNESDAY, THURSDAY, FRIDAY, and SATURDAY) , directions (NORTH, SOUTH, EAST, and WEST), season (SPRING, SUMMER, WINTER, and AUTUMN or FALL), colors (RED, YELLOW, BLUE, GREEN, WHITE, and BLACK) etc. According to the Java naming conventions, we should have all constants in capital letters. So, we have enum constants in capital letters.

Java Enums can be thought of as classes which have a fixed set of constants (a variable that does not change). The Java enum constants are static and final implicitly. It is available since JDK 1.5.

Enums are used to create our own data type like classes. The **enum** data type (also known as Enumerated Data Type) is used to define an enum in Java. Unlike C/C++, enum in Java is more *powerful*. Here, we can define an enum either inside the class or outside the class.

Java Enum internally inherits the *Enum class*, so it cannot inherit any other class, but it can implement many interfaces. We can have fields, constructors, methods, and main methods in Java enum.


enum improves type safety
enum can be easily used in switch
enum can be traversed
enum can have fields, constructors and methods
enum may implement many interfaces but cannot extend any class because it internally extends Enum class
Points to remember for Java Enum

Simple Example of Java Enum

```
class EnumExample1{
//defining the enum inside the class
public enum Season { WINTER, SPRING, SUMMER, FALL }
//main method
public static void main(String[] args) {
//traversing the enum
for (Season s : Season.values())
System.out.println(s);
}}
```

Output:

```
WINTER
SPRING
SUMMER
FALL
```

5.List

List in Java provides the facility to maintain the *ordered collection*. It contains the index-based methods to insert, update, delete and search the elements. It can have the duplicate elements also. We can also store the null elements in the list.

The List interface is found in the java.util package and inherits the Collection interface. It is a factory of ListIterator interface. Through the ListIterator, we can iterate the list in forward and backward directions. The implementation classes of List interface are ArrayList, LinkedList, Stack and Vector. The ArrayList and LinkedList are widely used in Java programming. The Vector class is deprecated since Java 5.

Java List Example

Let's see a simple example of List where we are using the ArrayList class as the implementation.

```java
import java.util.*;
public class ListExample1{
public static void main(String args[]){
 //Creating a List
 List<String> list=new ArrayList<String>();
 //Adding elements in the List
 list.add("Mango");
 list.add("Apple");
 list.add("Banana");
 list.add("Grapes");
 //Iterating the List element using for-each loop
 for(String fruit:list)
  System.out.println(fruit);

}
}
```

Output:

```
Mango
Apple
Banana
Grapes
```