**Unit-9**

**Java.lang packages:**

The **java.la**ng Java package is the default package. It provides classes that are fundamental and basic to the design of the Java Programming Language such as String, Math, class wrappers for primitive types and runtime support for threads. There is **no need to import** this package explicitly into the Java program.

Here are some important classes and concepts from the **java.lang** package:

**Object Class:** The Object class is the root class for all Java classes. It defines methods that are available to all objects, such as **equals**, **hashCode**, **toString**, and **getClass**.

**String Class**: The String class represents a sequence of characters. It's widely used for working with text in Java. The String class provides methods for string manipulation, searching, and comparison.

**Wrapper Classes**: These classes allow **primitive data types** to be treated as **objects**. For example, the **Integer** class is a wrapper for the **int** primitive type, and it provides methods to perform operations on integers.

**Autoboxing** and **Unboxing**: Java allows automatic conversion between **primitive** types and their corresponding **wrapper** classes. This is known as **autoboxing** (conversion of primitives to objects) and **unboxing** (conversion of objects to primitives).

**Enum Class**: **Enum** class allows you to create enumerated types. Enums provide a way to define a set of constants with meaningful names.

**Throwable Class Hierarchy**: The Throwable class is the base class for all exceptions and errors. It has two main subclasses: Exception (for checked exceptions) and Error (for unchecked errors).

**Math Class:** The Math class provides static methods for performing mathematical operations like square root, exponentiation, trigonometry, rounding, etc.

**Thread Class:** Although the Thread class itself is in the java.lang package, it's important to mention. Threads are essential for concurrent programming in Java.

**Java.lang.Math package:**

The **java.lang.Math** class contains methods for performing basic numeric operations such as the elementary exponential, logarithm, square root, and trigonometric functions.

**Math.abs(x):** Returns the absolute value of x.

**Math.sqrt(x):** Returns the square root of x.

**Math.pow(x, y):** Returns x raised to the power of y.

**Math.exp(x):** Returns the exponential value of x.

**Math.log(x):** Returns the natural logarithm (base e) of x.

**Math.sin(x), Math.cos(x), Math.tan(x):** Returns the trigonometric sine, cosine, and tangent of x (in radians).

**Math.ceil(x):** Returns the smallest integer greater than or equal to x.

**Math.floor(x):** Returns the largest integer less than or equal to x.

**Math.round(x):** Returns the closest long or int value to x, rounded to the nearest integer.

**Math.random():** Returns a random double value between 0.0 (inclusive) and 1.0 (exclusive).

public class App {

    public static void main(String[] args) throws Exception {

        double number = 16.0;

        double squareRoot = Math.sqrt(number);

        System.out.println("Square root of " + number + " is " + squareRoot);

        double numToRound = 5.67;

        long roundedNum = Math.round(numToRound);

        System.out.println("Rounded number: " + roundedNum);

        int value = -42;

        int absoluteValue = Math.abs(value);

        System.out.println("Absolute value of " + value + " is " + absoluteValue);

        double angle = 45.0;

        double sinValue = Math.sin(Math.toRadians(angle));

        double cosValue = Math.cos(Math.toRadians(angle));

        double tanValue = Math.tan(Math.toRadians(angle));

        System.out.println("sin(45 degrees): " + sinValue);

        System.out.println("cos(45 degrees): " + cosValue);

        System.out.println("tan(45 degrees): " + tanValue);

        double base = 2.0;

        double exponent = 3.0;

        double result = Math.pow(base, exponent);

        System.out.println(base + " raised to the power of " + exponent + " is " + result);

        double randomNum = Math.random(); // Generates a random double between 0 (inclusive) and 1 (exclusive)

        System.out.println("Random number: " + randomNum);

        int num1 = 20;

        int num2 = 30;

        int max = Math.max(num1, num2);

        int min = Math.min(num1, num2);

        System.out.println("Max: " + max);

        System.out.println("Min: " + min);

    }

}

Output:

Square root of 16.0 is 4.0

Rounded number: 6

Absolute value of -42 is 42

sin(45 degrees): 0.7071067811865475

cos(45 degrees): 0.7071067811865476

tan(45 degrees): 0.9999999999999999

2.0 raised to the power of 3.0 is 8.0

Random number: 0.6660185998948188

**Wrapper Class in Java**

The wrapper class in Java provides the mechanism to *convert primitive into object and object into primitive.*

***autoboxing****and****unboxing****feature convert primitives into objects and objects into primitives automatically. The automatic conversion of primitive into an object is known as* ***autoboxing*** *and vice-versa* ***unboxing****.*

Here are the primitive data types and their corresponding wrapper classes:

* byte: Byte
* short: Short
* int: Integer
* long: Long
* float: Float
* double: Double
* char: Character
* boolean: Boolean

public class App {

    public static void main(String[] args) throws Exception {

         // Using wrapper classes to convert primitive types to objects

         Integer intValue = Integer.valueOf(42);

         Double doubleValue = Double.valueOf(3.14);

         // Using autoboxing to automatically convert primitives to objects

         Character charValue = 'A';

         Boolean boolValue = true;

         // Unboxing to convert objects back to primitive types

         int intValuePrimitive = intValue.intValue();

         double doubleValuePrimitive = doubleValue.doubleValue();

         // Autounboxing

         char charValuePrimitive = charValue;

         boolean boolValuePrimitive = boolValue;

         System.out.println("intValue: " + intValue);

         System.out.println("doubleValue: " + doubleValue);

         System.out.println("charValue: " + charValue);

         System.out.println("boolValue: " + boolValue);

}

}

**Wrapper class Associated Methods:**

**Conversion Method:**

**Integer.valueOf(42):** Converts a primitive type to its wrapper class object.

Converts a Wrapper Object to its Primitive Type.

Integer intValue = Integer.valueOf(42);

int intValuePrimitive = intValue.intValue();

For example: **intValue(), doubleValue(), booleanValue().**

**Parsing Method:**

Integer.parseInt("42");

Parses a string of the primitive type and returns the wrapper class object.

**Comparison Method:**

*compareTo(anotherWrapper):* Compares the values of two wrapper objects. Returns a negative value if less, zero if equal, and positive if greater.

*equals(anotherObject):* Compares the values of two wrapper objects for equality.

**Common Utility Methods:**

*toString():* Returns a string representation of the wrapped value.

*hashCode():* Returns the hash code of the wrapped value.

*getClass():* Returns the class of the wrapper object.

**Using java.util package(Core Class or Legacy Class)**

Java versions earlier to Java2 did not contain any collections framework. They only contained some classes and an interface, which were used to store objects. Legacy class is often used to refer to classes that were part of the original Java programming language or early versions of Java and have since been replaced or deprecated in favor of newer approaches.

The **java.util** package defines the following legacy classes:

* HashTable
* Stack
* Dictionary
* Properties
* Vector

**HashTable:**

A synchronized data structure that stores **key-value pairs**, providing fast access to values using their keys.

import java.util.\*;

public class App {

    public static void main(String[] args) throws Exception {

        Hashtable<String, Integer> scores = new Hashtable<>();

        scores.put("key1", 95);

        scores.put("key2", 87);

        int keyone = scores.get("key1"); // Retrieves the score for Alice

        System.out.println(keyone);

}

}

**Stack:**

A data structure that follows the **Last-In-First-Out (LIFO)** principle, used for managing method calls and undo operations.

import java.util.\*;

public class App {

    public static void main(String[] args) throws Exception {

        Stack<String> browserHistory = new Stack<>();

        browserHistory.push("Homepage");

        browserHistory.push("Search results");

        String currentPage = browserHistory.pop(); // Retrieves the last visited page

        System.out.println(currentPage);

}

}

**Dictionary:**

An abstract class representing a collection of **key-value pairs**. In modern Java, it's replaced by the **Map** interface.

import java.util.\*;

public class App {

    public static void main(String[] args) throws Exception {

        Dictionary<String, String> countryCodes = new Hashtable<>();

        countryCodes.put("NP", "Nepal");

        String countryName = countryCodes.get("NP");

        System.out.println(countryName);

}

}

**Vector:**

A dynamic array-like data structure that can grow or shrink automatically and is synchronized. It's largely replaced by **ArrayList**.

import java.util.\*;

public class App {

    public static void main(String[] args) throws Exception {

        Vector<Integer> numbers = new Vector<>();

        numbers.add(10);

        numbers.add(20);

        int firstNumber = numbers.get(0);

        System.out.println(firstNumber);

}

}

**Enumeration(Legacy Interface)**

Enumeration in Java is an interface used to iterate through a collection of elements

import java.util.\*;

public class App {

    public static void main(String[] args) throws Exception {

        Vector<String> colors = new Vector<>();

        colors.add("Red");

        colors.add("Green");

        colors.add("Blue");

        Enumeration<String> colorEnum = colors.elements();

        while (colorEnum.hasMoreElements()) {

            String color = colorEnum.nextElement();

            System.out.println(color);

        }

}

}

**Random Number Generator:**

Random numbers of type integers, float, double, long, booleans can be created using this class.  
Arguments can be passed to the methods for defining the upper bound till which the number is to generate. For Example, **nextInt(4)** will generate numbers in the range 0 to 3 (both inclusive).

import java.util.\*;

public class App {

    public static void main(String[] args) throws Exception {

     // creating an instance of Random class

        Random rand = new Random();

        // Generating random integers in range 0 to 99

        int int1 = rand.nextInt(100);

         double dub1 = rand.nextDouble();//range o to 1

            System.out.println(int1);

        System.out.println(dub1);

    }

}