1. UTILITY CLASSES

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. 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.

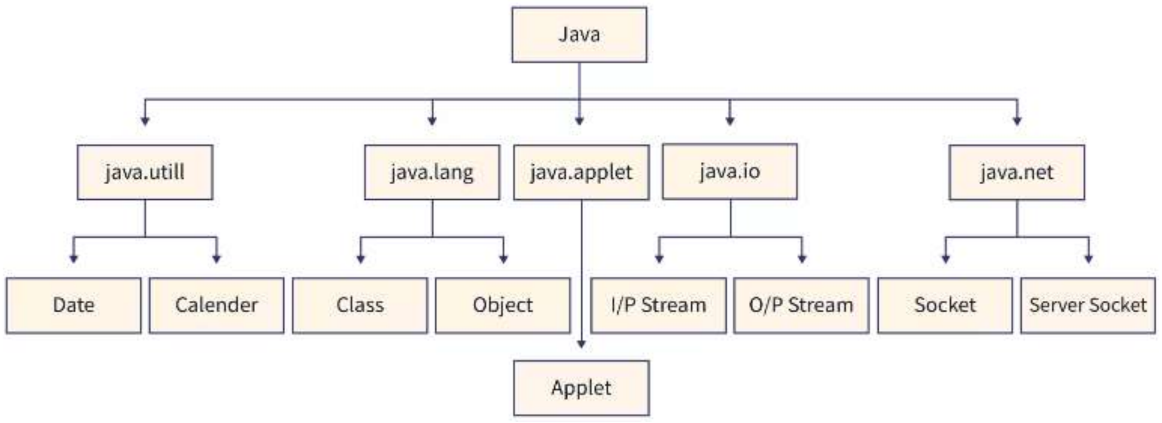


Fig 1: Utility classes in Java

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 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.

- 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

2. STRING TOKENISER

A string can be divided into tokens(Java tokens are the text (words) that are produced when the Java compiler splits a line of code. The Java program's simplest component is this.) using StringTokenizer in Java.

Internally, a StringTokenizer object keeps track of where it is in the string that has to be tokenized. Some procedures move this place ahead of the currently processed characters. By extracting a substring from the string that was used to generate the StringTokenizer object, a token is returned. It offers the initial stage of the parsing procedure, often known as the lexer or scanner.

The StringTokenizer in Java enables applications to tokenize strings. The Enumeration interface is implemented by it. Data parsing is done using this class. We must specify an input string and a string with delimiters in order to use the String Tokenizer class. Tokens are divided by delimiters, which are characters. The delimiter string's characters are all accepted as delimiters. New line, space, tab, and whitespace are the default delimiters.

The java.util package has a class called StringTokenizer that is used to tokenize strings. To put it another way, we can break a sentence up into its words and conduct several operations, such as counting the amount of tokens or splitting a sentence up into tokens.

This StringTokenizer includes constructors and methods that allow us to separate a sentence into tokens. StringTokenizer, tokenize the string on the basis of the delimiters provided to the String tokenizer class object. Whitespaces, tab, newlines, carriage returns, and form feeds are considered common delimiters. These delimiters are used by default, however users can define their own delimiters by including them as arguments in the parameter.

Constructors of the StringTokenizer Class

- StringTokenizer(String str) Tokenization of a certain string that is supplied in the parameter is accomplished by this constructor. All of the standard delimiters specified in the StringTokenizer class description are taken by constructor. WhiteSpaces, newlines, tabs, carriage returns ("r"), line feeds ("n"), and form feeds ("f") are the delimiters.
- StringTokenizer(String str, String delimiter) Based on the delimiter the user specifies in the input, this constructor implements string tokenization.
- .StringTokenizer(String str, String delimiter, boolean flag) This constructor has the ability to display the delimiter in addition to doing string tokenization based on the delimiter.

Methods:

1. int countTokens() -The countToken() method returns the number of tokens in the input string that are separated by any white space. The number of tokens in the String can be determined using this approach, and we can easily process the entire string by using this number of tokens as a loop parameter.

2. nextToken() -The StringTokenizer class's nextToken() method returns the next token as a String. The first time we use it, it returns the following token as the string's first token.

3. String nextToken(String delimiter)- The only difference between the nextToken(String delimiter) and nextToken() methods is that the former returns the next token based on the delimiter we pass as an argument to the latter. Any delimiter, including a symbol, a number, or a character, is acceptable.

4. boolean hasMoreTokens()-This method merely determines if the String contains any additional tokens. If a token is available, it returns true; otherwise, it returns false. It can be used in the while loop to process the entire string up until there are no more tokens left.

5. Object nextElement()-The nextElement() method is comparable to the StringTokenizer class's nextToken() method; The distinction is that, as opposed to the nextToken() method, which returns a String, it returns the value in the form of an Object.

```
import java.util.* ;
import java.io.* ;
public class classA {
  public static void main(String[] args)  {
    String myString = "The java.util package has a class called StringTokenizer,that
is used to tokenize strings.";
    StringTokenizer st = new StringTokenizer(myString,",",true);
    int numberOfTokens;
    numberOfTokens = st.countTokens();
    System.out.println("Number of tokens in the string: " + numberOfTokens);
    while (st.hasMoreTokens()) {
      System.out.println("The next token is " + st.nextToken());
    }
        }
  }
```

Output:

```
Number of tokens in the string: 3
The next token is The java.util package has a class called StringTokenizer
The next token is ,
The next token is that is used to tokenize strings.
```

```
import java.util.* ;
import java.io.* ;
public class classA {
  public static void main(String[] args)  {
    String myString = "The java.util package has a class called StringTokenizer,that
is used to tokenize strings.";
    StringTokenizer st = new StringTokenizer(myString,",",true);
    int numberOfTokens;
    numberOfTokens = st.countTokens();
    System.out.println("Number of tokens in the string: " + numberOfTokens);
    while (st.hasMoreTokens()) {
      System.out.println("The next token is " + st.nextToken());
    }
        }
  }
```

Output:

```
Number of tokens in the string: 3
The next token is The java.util package has a class called StringTokenizer
The next token is ,
The next token is that is used to tokenize strings.
```

3. DATE CLASS

The Date class represents a specific point in time, down to milliseconds. It provides methods to manipulate and format dates, as well as perform arithmetic and comparison operations. Internally the class uses a base or reference date of January 1, 1970, 00:00:00 GMT (also known as the Unix epoch).

Methods:

- **Creating a Date object:**
- Date(): Creates an object representing the current date and time.

- Date(long millis): Creates a Date object with the specified number of milliseconds since January 1, 1970, 00:00:00 GMT (the Unix epoch).
- **Getting and setting date components:**
- .getTime(): Returns the number of milliseconds since the Unix epoch.
- .setTime(long time): Sets the time using the specified number of milliseconds.
- Formatting and parsing dates:
- .toString(): Returns a string representation. The default format is not very readable or localized.
- SimpleDateFormat class (from java.text package): Allows formatting and parsing of dates using patterns.
- **Comparing dates:**
- .before(Date when): Checks if the given date is before the time specified.
- .after(Date when): Checks if the given date is after the time specified.
- .compareTo(Date anotherDate): Returns a value based on the relative order of two dates.

1) Boolean After(Date date): This method is used to return if the current date is after the given date. It returns true if the current date is after the given date.

Syntax: public boolean after(Date date)

2) Boolean Before(Date date)

This method is used to return if the current date is before the given date. It returns true if the current date is before the given date.

Syntax: public boolean before(Date date)

3) int compareTo(Date date)

This method is used to compare the current date with the given date. It returns either 1, 0, or -1 based on the comparison made.

Syntax: public int compareTo(Date date)

The result can be: 0: if the argument date is equal to the given date. -1: if the argument date is greater than the given date. 1: if the argument date is less than the given date.

4) boolean equals(Date date)

This method is used to check whether two dates are equal or not based on their millisecond difference. It returns true if the given dates are equal.

Syntax: public boolean equals(Date date)

5) long getTime()

This method is used to return the time represented by this date object. It returns the number of milliseconds from 1st January 1970 till the current date.

Syntax: public long getTime()

6) void setTime(long time)

This method is used to set the current date and time to the given date and time. It sets the date object to represent time in milliseconds after January 1, 1970, 00:00:00 GMT.

Syntax: public void setTime(long time)

7) String toString()

This method is used to convert this date object to a String-type object.

Syntax: public String toString()

The date object is converted into a string. The resultant string is in the following format- day month dd hh:mm: ss zz yyyy

day: day of the week month: month dd: day of the month hh: hour mm: minute ss: second zz: time zone yyyy: year

8) int hashCode()

This method is used to generate a hash code for the given object.

Syntax: public int hashCode()

9) Object clone()

This method is used to generate a clone for the invoking object. This method overrides the clone method in the Object class.

Syntax: public object clone()

```
import java. util.Date;
public class Main
{
    public static void main(String[] args)
    {
        Date date1 = new Date(2017, 12, 18);
        Date date2 = new Date(1998, 8, 27);

        //check which date is after
        boolean a = date2.after(date1);
        boolean b = date1.after(date2);
        if(a) System. out.println("Date2 is after date1");
        else System. out.println("Date1 is after date2");

        // Use of clone() method
        Object date3 = date1.clone();
        System.out.println("Cloned date3 " + date3.toString());

        // Use of before() to check date2 is after date1
        boolean c = date2.before(date1);
        if(c) System.out.println("Date2 is before date1");
        else System.out.println("Date2 is after date1");
    }
}
```

Output:

```
Date1 is after date2
Cloned date3 Fri Jan 18 00:00:00 GMT 3918
Date2 is before date1
```

4. CALENDER CLASS

Java's calendar class is an abstract class that gives you access to numerous calendar instances and methods for manipulating dates and times. To address several issues with java. util.Date class, the java.util.Calendar class was added to Java with JDK 1.4. It did simplify some tasks; for example, it is now simpler to create arbitrary dates with the new GregorianCalendar(2016, Calendar.JUNE, 11) constructor than with the Date class, where the year starts from 1900 and the month starts from zero.

The Java Calendar class is an abstract class that offers methods for converting dates between a specific point in time and a collection of calendar fields like MONTH, YEAR, HOUR, and so on. It derives from an Object and implements a Comparable interface. Because it is an Abstract class, we cannot create an instance with a constructor. To instantiate and implement a sub-class, we must utilize the static method Calendar.getInstance().

Declaration

```
public abstract class Calendar
extends Object
implements Cloneable,Comparable<Calendar>,Serializable
```

Constructors:

- protected Calendar()-This constructor constructs a Calendar with the default time zone and locale.
- protected Calendar(TimeZone zone, Locale aLocale)- This constructor constructs a calendar with the specified time zone and locale.

Class methods

- abstract void add(int field, int amount): This method adds or subtracts the specified amount of time to the given calendar field, based on the calendar's rules.
- boolean after(Object when): This method returns whether this Calendar represents a time after the time represented by the specified Object.

- boolean before(Object when): This method returns whether this Calendar represents a time before the time represented by the specified Object.
- void clear(): This method sets all the calendar field values and the time value (millisecond offset from the Epoch) of this Calendar undefined.
- Object clone(): This method creates and returns a copy of this object.
- int compareTo(Calendar anotherCalendar): This method compares the time values (millisecond offsets from the Epoch) represented by two Calendar objects.
- boolean equals(Object obj)- This method compares this Calendar to the specified Object.
- int get(int field)- This method returns the value of the given calendar field.
- int getActualMaximum(int field)- This method returns the maximum value that the specified calendar field could have, given the time value of this Calendar.
- int getActualMinimum(int field)- This method returns the minimum value that the specified calendar field could have, given the time value of this Calendar.
- static Set<String> getAvailableCalendarTypes()- This method returns an unmodifiable Set containing all calendar types supported by Calendar in the runtime environment.
- static Locale[] getAvailableLocales()- This method returns an array of all locales for which the getInstance methods of this class can return localized instances.
- String getCalendarType()- This method returns the calendar type of this Calendar.
- String getDisplayName(int field, int style, Locale locale)- This method returns the string representation of the calendar field value in the given style and locale.
- Map<String,Integer> getDisplayNames(int field, int style, Locale locale)- This method returns a Map containing all names of the calendar field in the given style and locale and their corresponding field values.
- int getFirstDayOfWeek()- This method gets what the first day of the week is; e.g., SUNDAY in the U.S., MONDAY in France.
- abstract int getGreatestMinimum(int field)- This method returns the highest minimum value for the given calendar field of this Calendar instance.

- static Calendar getInstance()- This method gets a calendar using the default time zone and locale.
- abstract int getLeastMaximum(int field)- This method returns the lowest maximum value for the given calendar field of this Calendar instance.
- abstract int getMaximum(int field)-This method returns the maximum value for the given calendar field of this Calendar instance.
- int getMinimalDaysInFirstWeek()- This method gets what the minimal days required in the first week of the year are; e.g., if the first week is defined as one that contains the first day of the first month of a year, this method returns 1.
- abstract int getMinimum(int field)-This method returns the minimum value for the given calendar field of this Calendar instance.
- Date getTime()-This method returns a Date object representing this Calendar's time value (millisecond offset from the Epoch").
- long getTimeInMillis()-This method returns this Calendar's time value in milliseconds.
- TimeZone getTimeZone()-This method gets the time zone.
- int hashCode()-This method Returns a hash code for this calendar.
- boolean isLenient()-This method tells whether date/time interpretation is to be lenient.
- boolean isSet(int field)-This method determines if the given calendar field has a value set, including cases that the value has been set by internal fields calculations triggered by a get method call.
- abstract void roll(int field, boolean up)-This method adds or subtracts (up/down) a single unit of time on the given time field without changing larger fields.
- void set(int field, int value)-This method sets the given calendar field to the given value.
- void setFirstDayOfWeek(int value)-This method sets what the first day of the week is; e.g., SUNDAY in the U.S., MONDAY in France.
- void setLenient(boolean lenient)-This method specifies whether or not date/time interpretation is to be lenient.
- void setMinimalDaysInFirstWeek(int value)-This method sets what the minimal days required in the first week of the year are; For Example, if the first week is

defined as one that contains the first day of the first month of a year, call this method with value.

- void setTime(Date date)-This method sets this Calendar's time with the given Date.
- void setTimeInMillis(long millis)-This method sets this Calendar's current time from the given long value.
- void setTimeZone(TimeZone value)-This method sets the time zone with the given time zone value.
- void setWeekDate(int weekYear, int weekOfYear, int dayOfWeek)-This method sets the date of this Calendar with the given date specifiers - week year, week of year, and day of week.
- Instant toInstant-This method converts this object to an Instant.
- String toString()-This method return a string representation of this calendar.

```
import java.util.Calendar;
import java.util.Date;

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

    // create calendar objects.
    Calendar cal = Calendar.getInstance();
    Calendar future = Calendar.getInstance();

    // print the current date
    System.out.println("Current date: " + cal.getTime());

    // change year in future calendar
    future.set(Calendar.YEAR, 2025);
    System.out.println("Year is " + future.get(Calendar.YEAR));

    // check if calendar date is after current date
```

```
     Date time = future.getTime();


     if (future.after(cal)) {
       System.out.println("Date " + time + " is after current date.");
     }
   }
}
```
Output:

```
Current date: Fri Sep 23 14:35:06 IST 2022
Year is 2025
Date Tue Sep 23 14:35:06 IST 2025 is after current date
```

5. GREGORIAN CALENDER CLASS

A concrete subclass of the Calendar class is referred to as GregorianCalendar. The GregorianCalendar class has an implementation of all of its inherited members. The Calendar class implements the mostly used Gregorian calendar. In order to use the Gregorian calendar in Java, we import the Java.util.GregorianCalendar class in our program.

We cannot instantiate the Calendar class because of being an abstract class. So, the initialization of the calendar is done in the following way:

Calendar cal = Calendar.getInstance();

The initialization of the cal object is done with the current date and time in the default locale and timezone.

We can instantiate the GregorianCalendar class because of being a concrete class. So, the initialization of the calendar is done in the following way:

GregorianCalendar gcal = new GregorianCalendar();

The initialization of the gcal object is done with the current date and time in the default locale and timezone. The anno Domini(AD) and the Before Christ(BC) are the two fields defined by the GregorianCalendar class.

Constructors of GregorianCalendar class

- GregorianCalendar() - In order to initialize the object with the current time in the default time zone with the default locale, GregorainCalendar() is used.
- GregorianCalendar(int year, int month, int day) - In order to initialize the object with the date-set defined in the default locale and time zone, GregorianCalendar(int year, int month, int day) is used.
- GregorianCalendar(int year, int month, int day, int hours, int min)- It initializes the object with the date and time-set defined in the default locale and time zone.
- GregorianCalendar(int year, int month, int day int hours, int minutes, int seconds) - It initializes the object with the date and more specific time-set defined in the default locale and time zone.
- GregorianCalendar(Locale locale)- It initializes the object with a defined date, time, and locale.
- GregorianCalendar(TimeZone timeZone)-It initializes the object with the defined date, time-set, and TimeZone.
- GregorianCalendar(TimeZone timeZone, Locale locale)- It initializes the object with the defined Locale and TimeZone.

```
//importing classes for time zone, calendar, locale, and gregorian calendar
import java.util.TimeZone;
import java.util.Locale;
import java.util.Calendar;
import java.util.GregorianCalendar;

//creating GregorianCalendarExample1 to define default constructor
public class GregorianCalendarExample1 {
   //main() method start
   public static void main(String args[])
```

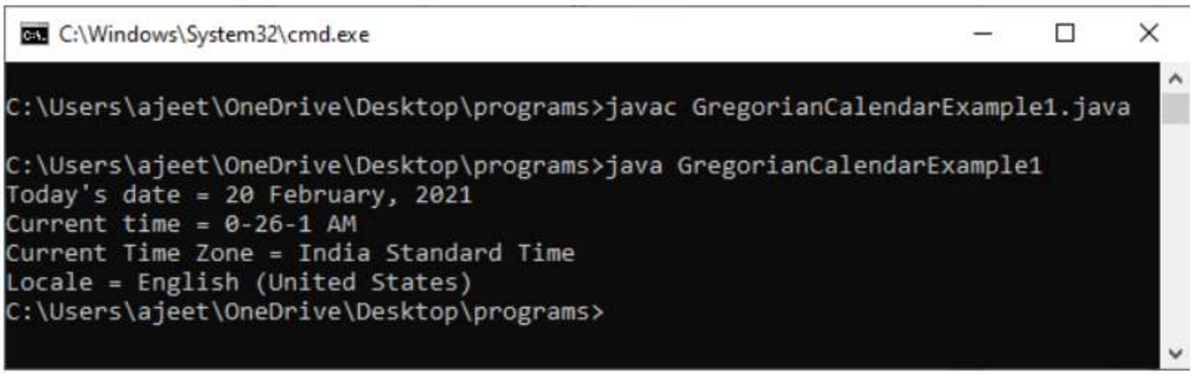
```
    {
        // creating array of months
        String months[] = { "January", "February", "March", "April",
                            "May", "June", "July", "August",
                            "September", "October", "November", "December" };

        // creating an array for storing AM(Morning) and PM(Evening)
        String arr[] = { "AM", "PM" };

        //creating GregorianCalendar class object using its default constructor
        GregorianCalendar obj = new GregorianCalendar();

        // Showing the time, date, locale, and time zone
        System.out.print("Today's date = "
                         + obj.get(Calendar.DATE) + " "
                         + months[obj.get(Calendar.MONTH)] + ", "
                         + obj.get(Calendar.YEAR) + "\n"
                         + "Current time = "
                         + obj.get(Calendar.HOUR) + "-"
                         + obj.get(Calendar.MINUTE) + "-"
                         + obj.get(Calendar.SECOND) + " "
                         + arr[obj.get(Calendar.AM_PM)] + "\n"
                         + "Current Time Zone = " + obj.getTimeZone().getDisplayName()
                         + "\n"
                         + "Locale = "
                         + Locale.getDefault().getDisplayName());
    }
}
```

Output:



```
C:\Users\ajeet\OneDrive\Desktop\programs>javac GregorianCalendarExample1.java

C:\Users\ajeet\OneDrive\Desktop\programs>java GregorianCalendarExample1
Today's date = 20 February, 2021
Current time = 0-26-1 AM
Current Time Zone = India Standard Time
Locale = English (United States)
C:\Users\ajeet\OneDrive\Desktop\programs>
```

6. RANDOM CLASS

Random class in Java which is present in the util package helps in generating random values of different data types like integer, float, double, long, boolean, etc. We can even mention the range in which we want the random number to be generated. Instances of java. util.Random is thread-safe. However concurrent usage of this instance can be misleading, sometimes it results in poor performance of the program. So, the best solution for this can be instead of using java. util.Random instance we can use ThreadLocalRandom, this is especially used in multithreaded programs. Instances of java. util.Random are not cryptographically secure. Therefore instead of that, we can use SecureRandom to get a cryptographically secure pseudo-random number generator for the use of security-sensitive applications. Classes that are implemented by using class Random can generate up to 32 pseudorandomly generated bits. It uses a protected utility method. For a simple application of generating a random number, we can also use a method which is Math. random().

Declaration of Random Class in Java:

After a brief introduction to the Random class in Java, let's learn more about how to declare a random class in Java. This section mainly explains it. The syntax for declaring a Random class in Java is as follows:

```
public class Random
    extends Object
    implements Serializable
```

Constructors of Random Class in Java:

- Random()- This constructor creates a new random generator. This method helps in instantiating the Random class and helps in generating random numbers.
- Random(long seed)- This constructor creates a new random number generator using a single long seed. The seed is considered to be the initial value of the internal state of the pseudorandom number which has to be generated. Generally, new Random(seed) is equivalent to:

```
Random rand = new Random() ;   rand.setSeed(seed);
```

Where seed represents the initial value The implementation of method setSeed() used by class Random uses only 48 bits of the given seed.

Methods:

- doubles()- Returns an unlimited stream of pseudorandom double values.
- ints()-Returns an unlimited stream of pseudorandom int values.
- longs()- Returns an unlimited stream of pseudorandom long values.
- next()-Generates the next pseudorandom number.
- nextBoolean()-Returns the next uniformly distributed pseudorandom boolean value from the random number generator's sequence
- nextByte()-Generates random bytes and puts them into a specified byte array.
- nextDouble()-Returns the next pseudorandom Double value between 0.0 and 1.0 from the random number generator's sequence
- nextFloat()-Returns the next uniformly distributed pseudorandom Float value between 0.0 and 1.0 from this random number generator's sequence
- nextGaussian()-Returns the next pseudorandom Gaussian double value with mean 0.0 and standard deviation 1.0 from this random number generator's sequence.
- nextInt()-Returns a uniformly distributed pseudorandom int value generated from this random number generator's sequence
- nextLong()-Returns the next uniformly distributed pseudorandom long value from the random number generator's sequence.

- setSeed()-Sets the seed of this random number generator using a single long seed.

```
// Java program to picturise examples of methods in java Random class
// import the random class from the util package
import java. util.Random;

// class which highlights the examples of Random package
public class Solution {

    // driver code
    public static void main(String[] args) {

        //instantiate the random object
        Random rand= new Random();

        //returns an unlimited stream of pseudorandom int values
            System. out.println("Integer values generated are: "+rand.ints());

       // returns the next pseudorandom boolean value which is generated
        by boolean Val = rand.nextBoolean();
        System. out.println("The random boolean value generated is : "+val);

        byte[] arr = new byte[10];
        // returns random bytes and put them in an array
        rand.nextBytes(arr);
        // print the generated byte values from the array arr
        System. out.print("The random bytes generated are ");
        for(int i = 0; i< arr.length; i++)  {
            System.out.printf("%d ", arr[i]);
        }
        System.out.println();
        // prints the double value which is generated by the below method
        System. out.println("The random double value generated is: " +
rand.nextDouble());


    }
}
```

Output:

```
   Integer values generated are java.util.stream.IntPipeline$Head@7852e922
   The random boolean value generated is: true
   The random bytes generated are -38 119 -1 75 62 -128 102 56 -36 65
   The random double value generated is: 0.6502191730539953
```

7. SCANNER CLASS

The Scanner class in Java is a part of the java.util package and is used for obtaining input of primitive types like int, double, etc., and strings from the user or a file. It provides various methods to parse and process input, making it versatile for reading different types of data. By creating an instance of the Scanner class and associating it with an input source like System.in, developers can easily interact with users through console input.

Java Scanner Class Declaration

```
 Scanner scanner = new Scanner(System.in);
```

This creates a Scanner object named scanner that is associated with the standard input stream (System.in), allowing you to read user input from the console.

Scanner Class Constructors

- Scanner(InputStream source): Initializes a Scanner to read from an input stream, such as System.in.
- Scanner(File source): Initializes a Scanner to read from a specified file.
- Scanner(String source): Initializes a Scanner to read from a string.

- Scanner(Readable source): Initializes a Scanner to read from any object that implements the Readable interface.
- Scanner(Path source): Initializes a Scanner to read from a specified file path.

These constructors allow flexibility in setting up the Scanner object to read input from various sources, including standard input, files, strings, or custom readable objects.

Methods:

- nextBoolean()- Used for reading Boolean value
- nextByte()- Used for reading Byte value
- nextDouble()- Used for reading Double value
- nextFloat()- Used for reading Float value
- nextInt()-Used for reading Int value
- nextLine()- Used for reading Line value
- nextLong()-Used for reading Long value
- nextShort()- Used for reading Short value

```
import java.util.Scanner;     //importing Scanner class

public class Main

{

        public static void main(String args[])

        {

        Scanner sc=new Scanner(System.in);     //creating scanner object

        System.out.print("Enter your full name: ");

        String name = sc.nextLine();     //String input

        System.out.print("Enter your age: ");

        int age = sc.nextInt();     //integer input

        System.out.print("Enter your salary: ");

        double salary = sc.nextDouble();     //double-type input

        System.out.print("Are you happy today? True/False: ");

        boolean mood = sc.nextBoolean();     //boolean input

        System.out.println();

        System.out.println("Name: "+name+"\nAge: "+age+"\nSalary: "+salary+"
\nHappy? "+mood);

        }

}
```

Output:

```
Enter your full name: Neha Sharma

Enter your age: 25

Enter your salary: 10000

Are you happy today? True/False: true

Name: Neha Sharma

Age: 25
```

```
Salary: 10000.0

Happy? true
```