**Arrays In JAVA**

Arrays, one of Java’s **most useful objects**, enable you to collect objects into an easy-to manage list.

Arrays in Java are actual **objects**that can be passed around and treated just like other objects.

An array is a group of like-typed variables that are referred to by a common name. Arrays of any type can be created and may have one or more dimensions.

A specific element in an array is accessed by its **index**.

Arrays offer a convenient means of **grouping related information**.

To create an array in Java, you use three steps:

1. Declare a variable to hold the array.

2. Create a new array object and assign it to the array variable.

3. Store things in that array

**Declaring Array Variables**

The first step to creating an array is creating a variable that will hold the array, just as you would any other variable.

Array variables indicate the type of object the array will hold (just as they do for any variable) and the name of the array, followed by empty brackets ([]).

The following are all typical array variable declarations:

String difficultWords[];

Float hits[];

int temps[];

An alternate method of defining an array variable is to put the brackets after the type instead of after the variable. They are equivalent, but this latter form is often much more readable.

So, for example, these three declarations could be written like this:

String[] difficultWords;

Float[] hits;

int[] temps;

**Creating Array Objects**

The second step is to create an array object and assign it to that variable. There are two ways to do this:

■ Using new

■ Directly initializing the contents of that array

The first way is to use the new operator to create a new instance of an array: String[] names = new String[10];

That line creates a new array of *Strings* with ten *slots*, or elements. When you create the new array object using *new*, you must indicate how many elements that array will hold.

Array objects can contain primitive types such as integers or booleans, just as they can contain objects:

int[] temps = new int[99];

When you create an array object using *new*, all its elements are initialized for you (0 for numeric arrays, false for boolean, ‘\0’ for character arrays, and null for everything else).

You can also create and initialize an array at the same time. Instead of using new to create the new array object, enclose the elements of the array inside braces, separated by commas:

String[] chiles = { “jalapeno”, “anaheim”, “serrano,” “habanero,” “thai” };

Each of the elements inside the braces must be of the same type and must be the same type as the variable that holds that array. An array the size of the number of elements you’ve included will be automatically created for you. This example creates an array of String objects named chiles that contains five elements.

**Accessing Array Elements**

Once you have an array with initial values, you can test and change the values in each slot of that array. To get at a value stored within an array, use the array subscript expression:

myArray[subscript];

The *subscript* is the slot within the array to access, which can also be an expression.

Note that all array subscripts are checked to make sure that they are inside the boundaries of the array (greater than 0 but less than the array’s length) either when your Java program is compiled or when it is run. It is impossible in Java to access or assign a value to an array

element outside of the boundaries of the array. Note the following two statements, for example

String arr[] = new String[10];

arr[10] = “eggplant”;

A program with that last statement in it produces a compiler error at that line when you try to compile it. The array stored in arr has only ten elements numbered from 0, the element at subscript 10 doesn’t exist, and the Java compiler will check for that.

You can test for the *length* of the array in your programs using the length instance variable— it’s available for all array objects, regardless of type:

int len = arr.length

**Changing Array Elements**

To assign a value to a particular array slot, merely put an assignment statement after the array access expression:

myarray[1] = 15;

sentence[0] = “The”;

sentence[10] = sentence[0];

Example:

Here is a program that creates an array of the number of days in each month. // **Demonstrate a one-dimensional array.**

class My\_Array {

public static void main(String args[])

{

int month\_days[];

month\_days = new int[12];

month\_days[0] = 31;

month\_days[1] = 28;

month\_days[2] = 31;

month\_days[3] = 30;

month\_days[4] = 31;

month\_days[5] = 30;

month\_days[6] = 31;

month\_days[7] = 31;

month\_days[8] = 30;

month\_days[9] = 31;

month\_days[10] = 30;

month\_days[11] = 31;

System.out.println("April has " + month\_days[3] + " days.");

}

}

**Initializing an Array Elements**

An array initializer is a list of comma-separated expressions surrounded by curly braces. The commas separate the values of the array elements.

The array will automatically be created large enough to hold the number of elements you specify in the array initializer. There is no need to use *new*.

// An improved version of the previous program.

class AutoArray {

public static void main(String args[]) {

int month\_days[] = { 31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31 };

System.out.println("April has " + month\_days[3] + " days.");

}

}

Here is one more example that uses a one-dimensional array. It finds the average of a set of numbers. // Average an array of values.

class Average {

public static void main(String args[]) {

double nums[] = {10.1, 11.2, 12.3, 13.4, 14.5};

double result = 0;

int i ;

for(i=0; i<5; i++)

result = result + nums[i];

System.out.println("Average is " + result / 5);

}

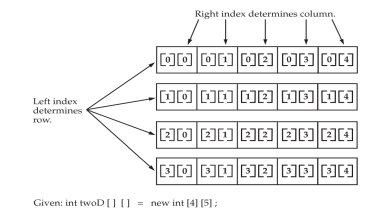
}

**Multidimensional Arrays**

**I**n Java, multidimensional arrays are actually arrays of arrays. These, as you might expect, look and act like regular multidimensional arrays. However, as you will see, there are a couple of subtle differences. To declare a multidimensional array variable, specify each additional index using another set of square brackets.

For example, the following declares a two dimensional array variable called twoD. **int twoD[][] = new int[4][5];**

This allocates a 4 by 5 array and assigns it to twoD. Internally this matrix is implemented as an array of arrays of int.



The following program numbers each element in the array from left to right, top to bottom, and then displays these values:

// Demonstrate a two-dimensional array.

class TwoDArray {

public static void main(String args[]) {

int twoD[][]= new int[4][5];

int i, j, k = 0;

for(i=0; i<4; i++)

for(j=0; j<5; j++) {

twoD[i][j] = k;

k++;

}

for(i=0; i<4; i++) {

for(j=0; j<5; j++)

System.out.print(twoD[i][j] + " "); System.out.println();

}

}

}

**Module-03**

**Functions in java**

**String functions**

In Java, string functions refer to the built-in operations and methods that can be performed on strings, which are objects of the **java.lang.String class.**

These functions provide functionality to manipulate and transform strings, such as concatenating, finding the length, searching for substrings, replacing characters, converting to uppercase or lowercase, etc.

Some common examples of string functions in **Java include length(), substring(), indexOf(), replace(), toUpperCase(), and toLowerCase().**

1. Concatenation

In Java, String concatenation forms a new String that is the combination of multiple strings. There are two ways to concatenate strings in Java:

1. By + (String concatenation) operator

2. By concat() method

1) String Concatenation by + (String concatenation) operator

Java String concatenation operator (+) is used to add strings.

For Example:

1. class TestStringConcatenation1{

2. public static void main(String args[]){

3. String output ="Sachin"+" Tendulkar";

4. System.out.println(output);

5. }

6. }

**2) String Concatenation by concat() method**

The String concat() method concatenates the specified string to the end of current string.

Syntax:

public String concat(String another)

Let's see the example of String concat() method.

class TestStringConcatenation3{

public static void main(String args[]){

String s1="Sachin ";

String s2="Tendulkar";

String s3=s1.concat(s2);

System.out.println(s3);//Sachin Tendulkar

}

}

**Example 2:**

**public class ConcatenationExample {**

**public static void main(String[] args) {**

**String str1 = "Hello";**

**String str2 = " World";**

**// Concatenating two strings using the + operator**

**String concatenatedStr = str1 + str2;**

**System.out.println("Concatenated string: " + concatenatedStr); // Concatenating using the concat() method**

**String concatenatedStr2 = str1.concat(str2);**

**System.out.println("Concatenated string using concat(): " + concatenatedStr2);**

**}**

**}**

**Substring Function**

The substring function is a method of the java.lang.String class that returns a new string that is a portion of the original string. The substring function takes two arguments: the starting index (inclusive) and the ending index (exclusive) of the desired substring**.**

**public class SubstringExample {**

**public static void main(String[] args) {**

**String str = "Hello, World!";**

**// Extracting a substring using the substring() method**

**String subStr = str.substring(7, 12);**

**System.out.println("Substring: " + subStr);**

**}**

**}**

In this example, the substring function is used to extract a part of the string, from index 7 to index 11 (the end index is exclusive). The resulting subStr variable contains the string "World".

**String Editing in Java**

String editing in Java refers to the process of modifying or transforming a string in some way. This can include replacing a character or substring, converting the string to uppercase or lowercase, removing leading or trailing white spaces, or any other operation that changes the contents of a string.

Java provides several methods for string editing, including replace, toUpperCase, toLowerCase, trim, and others, as part of the java.lang.String class. These methods can be used to perform various string transformations and manipulations.

For example, you can use the replace method to replace all occurrences of a specific character or substring in a string, or the toUpperCase and toLowerCase methods to convert a string to uppercase or lowercase respectively.

**public class StringEditingExample {**

**public static void main(String[] args) {**

**String str = "Hello, World!";**

**// Replacing a character or substring**

**String replacedStr = str.replace("o", "\*");**

**System.out.println("String after replacement: " + replacedStr); // Converting to uppercase or lowercase**

**String upperCase = str.toUpperCase();**

**System.out.println("String in uppercase: " + upperCase);**

**String lowerCase = str.toLowerCase();**

**System.out.println("String in lowercase: " + lowerCase);**

**}**

**}**

In this example, the replace function is used to replace all occurrences of the character "o" with the character "\*", and the toUpperCase and toLowerCase functions are used to convert the string to uppercase and lowercase respectively.

**Testing for Equality in java**

Testing for equality between strings involves comparing the contents of two string objects to determine if they are equal. There are two main ways to perform this comparison:

Using the equals method: The equals method is a member of the java.lang.String class, and it compares the contents of two strings to see if they are equal. It returns true if the strings are equal, and false otherwise.

For example:

String str1 = "Hello";

String str2 = "Hello";

boolean areEqual = str1.equals(str2);

System.out.println("Are the strings equal? " + areEqual);

Using the == operator:

The == operator can also be used to compare two strings, but it compares the references to the objects, not the contents of the objects. This means that if two strings have the same contents, they will be considered equal only if they are actually the same object.

For example:

String str1 = "Hello";

String str2 = "Hello";

boolean areEqual = (str1 == str2);

System.out.println("Are the strings equal? " + areEqual);

In general, it is recommended to use the equals method when testing for equality between strings, as it provides a more accurate comparison of the contents of the strings.

**Character extraction function**

In Java, the java.lang.String class provides several methods for extracting characters from a string:

a) **charAt**: The charAt method returns the character at a specified index in the string.

For example:

String str = "Hello";

char c = str.charAt(2);

System.out.println("Character at index 2: " + c);

b) **toCharArray**: The toCharArray method returns an array of characters that represents the contents of the string.

For example:

String str = "Hello";

char[] charArray = str.toCharArray();

System.out.print("Characters in the string: ");

for (char c : charArray) {

System.out.print(c + " ");

}

System.out.println();

c) **getChars**: The getChars method copies characters from the string into a specified character array.

For example:

String str = "Hello";

char[] charArray = new char[5];

str.getChars(0, 5, charArray, 0);

System.out.print("Characters in the string: ");

for (char c : charArray) {

System.out.print(c + " ");

}

System.out.println();

**d) getBytes**

The getBytes method in Java is a method of the java.lang.String class. It returns a byte array that represents the contents of the string encoded in a specified character encoding. The method is often used to convert a string into a sequence of bytes that can be transmitted over a network, written to a file, or used as input to other operations.

The getBytes method takes one optional argument, which is the name of the character encoding to be used. If the argument is not specified, the default character encoding of the platform will be used.

**For example:**

**String str = "Hello";**

**byte[] bytes = str.getBytes();**

**System.out.println("Bytes in the string: " + Arrays.toString(bytes));**

This code outputs the bytes of the string in the default character encoding of the platform.

It's important to note that the specific bytes generated by the getBytes method can depend on the character encoding used, so it's important to specify the correct encoding if your string contains characters that are not part of the default encoding.

**Formatting functions in JAVA**

formatting functions are used to manipulate and format the representation of data in a specific way. These functions are used to convert values of various data types into strings that can be displayed or used in other ways.

Some of the most commonly used formatting functions in Java are:

**a) String.format:** The String.format method allows you to format a string with placeholders, and replace the placeholders with values.

**For example:**

**int number = 42;**

**String formattedString = String.format("The number is %d", number);**

**System.out.println(formattedString);**

**b)DecimalFormat:** The DecimalFormat class provides a way to format numbers in a specific way, such as specifying the number of decimal places, the grouping of digits, and the use of currency symbols.

**For example:**

**double pi = 3.14159265;**

**DecimalFormat decimalFormat = new DecimalFormat("0.00"); String formattedPi = decimalFormat.format(pi);**

**System.out.println("Formatted value of Pi: " + formattedPi);**

**OR**

**double number = 42.123456;**

**DecimalFormat decimalFormat = new DecimalFormat("#.##"); String formattedString = decimalFormat.format(number); System.out.println(formattedString);**

**c)DateFormat:** The DateFormat class provides a way to format dates and times in a specific way, such as specifying the date and time pattern, the time zone, and the locale.

**For example:**

**Date date = new Date();**

**DateFormat dateFormat = new SimpleDateFormat("yyyy-MM-dd HH:mm:ss");**

**String formattedString = dateFormat.format(date);**

**System.out.println(formattedString);**

**OR**

**Date date = new Date();**

**DateFormat dateFormat =**

**DateFormat.getDateInstance(DateFormat.SHORT);**

**String formattedDate = dateFormat.format(date);**

**System.out.println("Formatted date: " + formattedDate);**

All String Functions

| **Method Description Return Type** charAt() Returns the character at the  char  specified index (position)  codePointAt() Returns the Unicode of the  int  character at the specified index  codePointBefore() Returns the Unicode of the  int  character before the specified  index  codePointCount() Returns the number of Unicode  int  values found in a string.  compareTo() Compares two strings  int  lexicographically  compareToIgnoreCase() Compares two strings  int  lexicographically, ignoring case  differences  concat() Appends a string to the end of  String  another string  contains() Checks whether a string  boolean  contains a sequence of  characters  contentEquals() Checks whether a string  boolean  contains the exact same  sequence of characters of the  specified CharSequence or  StringBuffer  copyValueOf() Returns a String that  String  represents the characters of  the character array  endsWith() Checks whether a string ends  boolean  with the specified character(s)  equals() Compares two strings. Returns  boolean  true if the strings are equal,  and false if not |
| --- |

equalsIgnoreCase() Compares two strings, ignoring case considerations

format() Returns a formatted string using the specified locale,

format string, and arguments

getBytes() Encodes this String into a sequence of bytes using the

named charset, storing the

result into a new byte array

getChars() Copies characters from a string to an array of chars

hashCode() Returns the hash code of a string

indexOf() Returns the position of the first found occurrence of specified

characters in a string

intern() Returns the canonical representation for the string

object

isEmpty() Checks whether a string is empty or not

lastIndexOf() Returns the position of the last found occurrence of specified

characters in a string

length() Returns the length of a specified string

matches() Searches a string for a match against a regular expression,

and returns the matches

offsetByCodePoints() Returns the index within this String that is offset from the

given index by codePointOffset

code points

regionMatches() Tests if two string regions are equal

replace() Searches a string for a specified value, and returns a

new string where the specified

values are replaced

boolean String

byte[]

void

int

int

String

boolean int

int

boolean int

boolean String

| replaceFirst() Replaces the first occurrence of  String  a substring that matches the  given regular expression with  the given replacement  replaceAll() Replaces each substring of this  String  string that matches the given  regular expression with the  given replacement  split() Splits a string into an array of  String[]  substrings  startsWith() Checks whether a string starts  boolean  with specified characters  subSequence() Returns a new character  CharSequence  sequence that is a subsequence  of this sequence  substring() Returns a new string which is  String  the substring of a specified  string  toCharArray() Converts this string to a new  char[]  character array  toLowerCase() Converts a string to lower case  String  letters  toString() Returns the value of a String  String  object  toUpperCase() Converts a string to upper case  String  letters  trim() Removes whitespace from both  String  ends of a string  valueOf() Returns the string  String  representation of the specified  value |
| --- |

**public class StringFunctionsExample {**

**public static void main(String[] args) {**

**String str = "Hello, World!";**

**// Finding the length of a string**

**int length = str.length();**

**System.out.println("Length of the string: " + length); // Extracting a substring**

**String subStr = str.substring(7, 12);**

**System.out.println("Substring: " + subStr);**

**// Finding the index of a character or substring int index = str.indexOf("W");**

**System.out.println("Index of character 'W': " + index); // Replacing a character or substring**

**String replacedStr = str.replace("o", "\*");**

**System.out.println("String after replacement: " + replacedStr); // Converting to uppercase or lowercase**

**String upperCase = str.toUpperCase();**

**System.out.println("String in uppercase: " + upperCase); String lowerCase = str.toLowerCase();**

**System.out.println("String in lowercase: " + lowerCase); }**

**}**

**Date and Time functions in java**

Java has several classes for working with dates and times, including:

**java.util.Date**: This class represents a specific instant in time, with millisecond precision.

**java.util.Calendar**: This class provides methods for converting between a Date object and a set of integer fields such as YEAR, MONTH, DAY\_OF\_MONTH, HOUR, and so on.

**java.time.LocalDate**: This class represents a date without time and time zone information.

**java.time.LocalTime**: This class represents time without date and time zone information.

**java.time.LocalDateTime**: This class represents a date-time combination without time zone information.

**java.time.ZonedDateTime**: This class represents a date-time with full information of the time-zone.

**Examples:**

**1. java.util.Date**

**import java.util.Date;**

**public class Main {**

**public static void main(String[] args) {**

**// Create a Date object for the current date and time**

**Date now = new Date();**

**// Print the date and time information**

**System.out.println("Current date and time: " + now);**

**// Get the time in milliseconds since January 1, 1970, 00:00:00 GMT long time = now.getTime();**

**System.out.println("Time in milliseconds: " + time);**

**}**

**}**

This program creates a Date object and uses it to get the current date and time information. The Date class has a getTime() method that returns the number of milliseconds since January 1, 1970, 00:00:00 GMT, which is a standard reference point for representing dates and times in computer systems.

**2. java.util.Calendar**

import java.util.Calendar;

import java.util.Date;

public class Main {

public static void main(String[] args) {

// Get a Calendar instance for the current date and time Calendar calendar = Calendar.getInstance();

// Get the year, month, and day information

int year = calendar.get(Calendar.YEAR);

int month = calendar.get(Calendar.MONTH);

int day = calendar.get(Calendar.DAY\_OF\_MONTH);

System.out.println("Year: " + year + ", Month: " + month + ", Day: " + day);

// Get the hour, minute, and second information

int hour = calendar.get(Calendar.HOUR\_OF\_DAY);

int minute = calendar.get(Calendar.MINUTE);

int second = calendar.get(Calendar.SECOND);

System.out.println("Hour: " + hour + ", Minute: " + minute + ", Second: " + second);

// Get the Date object from the Calendar instance

Date date = calendar.getTime();

System.out.println("Date: " + date);

}

}

This program creates a Calendar instance and uses it to get the current date and time information, including the year, month, day, hour, minute, and second. The Calendar class has a get() method that returns the value of a specified field, such as YEAR, MONTH, DAY\_OF\_MONTH, HOUR\_OF\_DAY, MINUTE, or SECOND. The program also demonstrates how to convert the Calendar instance to a Date object.

**3. java.time.LocalDate**

import java.time.LocalDate;

public class Main {

public static void main(String[] args) {

// Get the current date

LocalDate now = LocalDate.now();

// Print the current date

System.out.println("Current date: " + now);

// Get a specific date

LocalDate birthday = LocalDate.of(1997, 9, 15);

// Print the specific date

System.out.println("My birthday: " + birthday);

// Get the difference between two dates in days

long days = now.toEpochDay() - birthday.toEpochDay(); System.out.println("Days between two dates: " + days); }

}

This program demonstrates how to create LocalDate objects for the current date and a specific date, and how to calculate the difference between two dates in days. The LocalDate class has a now() static method that returns the current date, and an of() static method that creates a date from the year, month, and day information. The LocalDate class also has a toEpochDay() method that returns the number of days from January 1, 1970, to the date.

4. **java.time.LocalTime**

**import java.time.LocalTime;**

public class Main {

public static void main(String[] args) {

// Get the current time

LocalTime now = LocalTime.now();

// Print the current time

System.out.println("Current time: " + now);

// Get a specific time

LocalTime appointment = LocalTime.of(14, 30);

// Print the specific time

System.out.println("Appointment time: " + appointment);

// Get the difference between two times in minutes

long minutes = now.toSecondOfDay() / 60 - appointment.toSecondOfDay() / 60;

System.out.println("Minutes between two times: " + minutes); }

}

This program demonstrates how to create LocalTime objects for the current time and a specific time, and how to calculate the difference between two times in minutes. The LocalTime class has a now() static method that returns the current time, and an of() static method that creates a time from the hour and minute information. The LocalTime class also has a toSecondOfDay() method that returns the number of seconds from midnight to the time.

**5. java.time.LocalDateTime**

**import java.time.LocalDateTime;**

**public class Main {**

**public static void main(String[] args) {**

**// Get the current date and time**

**LocalDateTime now = LocalDateTime.now();**

**// Print the current date and time**

**System.out.println("Current date and time: " + now);**

**// Get a specific date and time**

**LocalDateTime meeting = LocalDateTime.of(2023, 2, 7, 14, 30);**

**// Print the specific date and time**

**System.out.println("Meeting date and time: " + meeting);**

**// Get the difference between two dates and times in minutes long minutes = now.toEpochSecond(now.getOffset()) / 60 - meeting.toEpochSecond(meeting.getOffset()) / 60;**

**System.out.println("Minutes between two date and time: " + minutes);**

**}**

**}**

This program demonstrates how to create LocalDateTime objects for the current date and time and a specific date and time, and how to calculate the difference between two dates and times in minutes. The LocalDateTime class has a now() static method that returns the current date and time, and an of() static method that creates a date and time from the year, month, day, hour, and minute information. The LocalDateTime class also has a toEpochSecond() method that returns the number of seconds from January 1, 1970, to the date and time. The getOffset() method returns the time zone offset.

6. **java.time.ZonedDateTime**

import java.time.ZoneId;

import java.time.ZonedDateTime;

public class Main {

public static void main(String[] args) {

// Get the current date and time in the default time zone ZonedDateTime now = ZonedDateTime.now();

// Print the current date and time

System.out.println("Current date and time in default time zone: " + now);

// Get the current date and time in a specific time zone

ZonedDateTime nowInLondon =

ZonedDateTime.now(ZoneId.of("Europe/London"));

// Print the current date and time in a specific time zone System.out.println("Current date and time in London: " + nowInLondon);

// Get a specific date and time in a specific time zone

ZonedDateTime meetingInNewYork = ZonedDateTime.of(2023, 2, 7, 14, 30, 0, 0, ZoneId.of("America/New\_York"));

// Print the specific date and time in a specific time zone System.out.println("Meeting date and time in New York: " + meetingInNewYork);

}

}

This program demonstrates how to create ZonedDateTime objects for the current date and time in the default time zone and a specific time zone, and for a specific date and time in a specific time zone.

The ZonedDateTime class has a now() static method that returns the current date and time in the default time zone, and another now() static method that takes a ZoneId argument and returns the current date and time in a specific time zone. The of() static method creates a date and time from the year, month, day, hour, minute, second, nanosecond, and ZoneId information.

**Formatting functions in JAVA**

Formatting functions are used to format output, to display the result in a specific format, and to parse string input into specific data types.

These formatting functions provide a convenient way to display data in a specific format, making it easier to read and understand.

Some of the most commonly used formatting functions in Java include:

**Printf**: The printf method is used to format output and print it to the console. It uses a format string and a variable number of arguments to specify the output format.

Example:

**public class Main {**

**public static void main(String[] args) {**

**int number = 42;**

**String name = "Alice";**

**double pi = 3.141592653589793;**

**// Use printf to format and print the output**

**System.out.printf("Hello, %s! The number is %d and the value of pi is %.2f.\n", name, number, pi);**

**}**

**}**

In this example, we have three variables: number, name, and pi. We use the printf method to format and print the values of these variables to the console. The format string "Hello, %s! The number is %d and the value of pi is %.2f.\n" specifies the output format. The %s placeholder is used to display the value of the name variable as a string, the %d placeholder is used to display the value of

the number variable as an integer, and the %.2f placeholder is used to display the value of the pi variable as a floating-point number with two decimal places. The \n at the end of the format string is a line break, which causes the output to be printed on a new line.

**String.format**: This method is similar to the printf method, but it returns a formatted string instead of printing it to the console. The format string and arguments are the same as those used by printf.

Example:

**public class Main {**

**public static void main(String[] args) {**

**int number = 42;**

**String name = "Alice";**

**double pi = 3.141592653589793;**

**// Use String.format to format a string**

**String result = String.format("Hello, %s! The number is %d and the value of pi is %.2f.", name, number, pi);**

**// Print the formatted string to the console**

**System.out.println(result);**

**}**

**}**

In this example, we have three variables: number, name, and pi. We use the String.format method to format the values of these variables into a string. The format string "Hello, %s! The number is %d and the value of pi is %.2f." is the same as the one used in the previous example. The %s placeholder is used to display the value of the name variable as a string, the %d placeholder is used to display the value of the number variable as an integer, and the %.2f placeholder is used to display the value of the pi variable as a floating-point number with two decimal places. The resulting

string is stored in the result variable, which is then printed to the console using the println method.

**DecimalFormat** : This class is used to format decimal numbers into strings. It allows you to specify the number of decimal places, the use of grouping separators, and other aspects of the decimal format.

Example:

**import java.text.DecimalFormat;**

**public class Main {**

**public static void main(String[] args) {**

**double number = 123456.78;**

**// Create a DecimalFormat object**

**DecimalFormat formatter = new DecimalFormat("#,##0.00"); // Use the format method to format the number**

**String result = formatter.format(number);**

**// Print the formatted string to the console**

**System.out.println(result);**

**}**

**}**

In this example, we have a variable number with the value 123456.789. We use the DecimalFormat class to format this number as a string. The format string "#,##0.00" specifies the desired format of the decimal number. The # symbol is used to represent optional digits, and the 0 symbol is used to represent mandatory digits. The , symbol is used to insert a grouping separator. In this case, the format string specifies that the number should be formatted with two decimal places and with a grouping separator every three digits to the left of

the decimal point. The formatter.format(number) method is used to format the number variable according to the specified format, and the resulting string is stored in the result variable, which is then printed to the console using the println method.

**SimpleDateFormat** : This class is used to format dates and times. You can specify the format of the date and time using pattern characters, such as "MM/dd/yyyy" for a date or "HH:mm:ss" for a time.

Example:

**import java.text.SimpleDateFormat;**

**import java.util.Date;**

**public class Main {**

**public static void main(String[] args) {**

**// Create a Date object for the current date and time**

**Date now = new Date();**

**// Create a SimpleDateFormat object**

**SimpleDateFormat formatter = new SimpleDateFormat("yyyy-MM-dd HH:mm:ss "); // Use the format method to format the date**

**String result = formatter.format(now);**

**// Print the formatted string to the console**

**System.out.println(result);**

**}**

**}**

In this example, we use the Date class to create a Date object for the current date and time. We then use the SimpleDateFormat class to format this date as a string. The format string "yyyy-MM-dd HH:mm:ss" specifies the desired format of the date and time. The yyyy symbol represents the year with century as a decimal number, the MM symbol represents the

month as a decimal number (with leading zeros), the dd symbol represents the day of the month as a decimal number (with leading zeros), the HH symbol represents the hour (00-23), the mm symbol represents the minute (00-59), and the ss symbol represents the second (00-59). The formatter.format(now) method is used to format the now variable according to the specified format, and the resulting string is stored in the result variable, which is then printed to the console using the println method.

**NumberFormat**: This class is used to **format numbers and currency values**. You can use it to format numbers with the appropriate thousands separators and decimal points for a specific locale.

**Example**:

**import java.text.NumberFormat;**

**import java.util.Locale;**

**public class Main {**

**public static void main(String[] args) {**

**double number = 123456.789;**

**// Create a NumberFormat object for the US locale**

**NumberFormat formatter = NumberFormat.getNumberInstance(Locale.US); // Use the format method to format the number**

**String result = formatter.format(number);**

**// Print the formatted string to the console**

**System.out.println(result);**

**}**

**}**

In this example, we have a variable number with the value 123456.789. We use the NumberFormat class to format this number as a string. The

NumberFormat.getNumberInstance(Locale.US) method is used to get a NumberFormat object for the US locale, which specifies the desired format of the number. The formatter.format(number) method is used to format the number variable according to the specified format, and the resulting string is stored in the result variable, which is then printed to the console using the println method. The output of the program will be **123,456.789** because the **US locale** uses a grouping separator (comma) every three digits to the left of the decimal point.