Programming in Java

User Input

Java **Scanner class** allows the user to take input from the console. It belongs to **java.util** package. It is used to read the input of primitive types like int, double, long, short, float, byte as well as String. It is the easiest way to read input in Java program.

To use the Scanner class, you must first import the java.util package, then create an object of the class and use any of the available methods found in the Scanner class documentation.

Example:

```
import java.util.Scanner;  // Import the Scanner class
class Main {
  public static void main(String[] args) {
    Scanner myObj = new Scanner(System.in); // Create a Scanner object
    System.out.println("Enter username");
    String userName = myObj.nextLine();  // Read user input
    System.out.println("Username is: " + userName);  // Output user
input
  }
}
```

Input Types

In the example above, we used the nextLine() method, which is used to read Strings. To read other types, look at the table below:

Method	Description
nextBoolean()	Reads a boolean value from the user
nextByte()	Reads a byte value from the user
nextDouble()	Reads a double value from the user
nextFloat()	Reads a float value from the user
nextInt()	Reads a int value from the user
nextLine()	Reads a String value from the user

nextLong()	Reads a long value from the user
nextShort()	Reads a short value from the user

Strings in Java

In Java, String is basically an object that represents sequence of char values.

There are two ways to create String object:

1. By string literal

   ```
   String greeting = "Hello";
   ```

2. By new keyword

   ```
   String s= new String("Hello");
   ```

Example:

```
public class StringExample {
      public static void main(String args[]){
            String s1="java";//creating string by Java string literal
            char ch[]={'s','t','r','i','n','g','s'};
            String s2=new String(ch);//converting char array to string
            String s3=new String("example");//creating Java string by
            new keyword
            System.out.println(s1);
            System.out.println(s2);
            System.out.println(s3);
}}
```

The String class has a set of built-in methods that you can use on strings.

Method	Description	Return Type
charAt()	Returns the character at the specified index (position)	char
compareTo()	Compares two strings lexicographically	int

compareToIgnoreCase()	Compares two strings lexicographically, ignoring case differences	int
concat()	Appends a string to the end of another string	String
contains()	Checks whether a string contains a sequence of characters	boolean
contentEquals()	Checks whether a string contains the exact same sequence of characters of the specified CharSequence or StringBuffer	boolean
endsWith()	Checks whether a string ends with the specified character(s)	boolean
equals()	Compares two strings. Returns true if the strings are equal, and false if not	boolean
equalsIgnoreCase()	Compares two strings, ignoring case considerations	boolean
format()	Returns a formatted string using the specified locale, format string, and arguments	String
getBytes()	Encodes this String into a sequence of bytes using the named charset, storing the result into a new byte array	byte[]
getChars()	Copies characters from a string to an array of chars	void
indexOf()	Returns the position of the first found occurrence of specified characters in a string	int
isEmpty()	Checks whether a string is empty or not	boolean
lastIndexOf()	Returns the position of the last found occurrence of specified characters in a string	int

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

String Examples

The most commonly used String methods are:

- **length():** return the length of the string.
- **charAt(int index):** return the char at the index position (index begins at 0 to length()-1).
- **equals():** for comparing the contents of two strings.

Take note that you should not use "==" to compare two strings.

Examples:

```
public static void main(String[] args){
String str = "Java is cool!";
System.out.println(str.length()); //return int 13
System.out.println(str.charAt(2)); //return char 'v'
System.out.println(str.charAt(5)); //return char 'i'
// Comparing two Strings
String Str2 = "Java is COOL!";
System.out.println(str.equals(Str2)); //return boolean false
System.out.println(str.equalsIgnoreCase(Str2));//return boolean true
System.out.println(Str2.equals(str));//return boolean false
System.out.println(Str2.equalsIgnoreCase(str));//return boolean true
}
```

***Note:**

(str == str2) compares references not values, so it only works if the two strings are the same instance. Meaning they are both declared as literal (not using new) and have the same value (refer to the same place in the String pool)

Additionally, you could use the JDK built-in methods **Integer.parseInt(*anIntStr*)** to convert a String containing a valid integer literal (e.g., "1234") into an int (e.g., 1234).

Example

```
String inStr = "5566";
int number = Integer.parseInt(inStr); // number <- 5566 2
```

You could use **Double.parseDouble(*aDoubleStr*)** or **Float.parseFloat(*aFloatStr*)** to convert a String (containing a floating-point literal) into a double or float, e.g.

```
String inStr = "55.66";
float aFloat = Float.parseFloat(inStr); // aFloat <- 55.66f
double aDouble = Double.parseDouble("1.2345"); // aDouble <- 1.2345
aDouble = Double.parseDouble("abc");// Runtime Error:
                                    NumberFormatException
```

To convert a primitive to a String, you can use the '+' operator to concatenate the primitive with an *empty*String ("") or use the JDK built-in methods:

• String.valueOf(*aPrimitve*)

• Integer.toString(anInt)

• Double.toString(aDouble)

• Character.toString(*aChar*)

• Boolean.toString(*aBoolean*), etc.

Example : concatenation

```
// String concatenation operator '+' (applicable to ALL primitive
types)
String s1 = 123 + ""; // int 123 -> "123"
String s2 = 12.34 + ""; // double 12.34 -> "12.34"
String s3 = 'c' + ""; // char 'c' -> "c"
String s4 = true + ""; // boolean true -> "true"
```

Example : String.valueOf

```
// String.valueOf(a Primitive) is applicable to ALL primitive types
String s1 = String.valueOf(12345); // int 12345 -> "12345"
String s2 = String.valueof(true); // boolean true -> "true"
double d = 55.66; String s3 = String.valueOf(d); // double 55.66 ->
"55.66" // toString() for each primitive type
String s4 = Integer.toString(1234); // int 1234 -> "1234"
String s5 = Double.toString(1.23); // double 1.23 -> "1.23"
String c1 = Character.toString('z'); // char 'z' -> "z"
```

Java Arrays

Arrays are used to store multiple values in a single variable, instead of declaring separate variables for each value. To declare an array, define the variable type with **square brackets**:

```
String[] cars;
```

We have now declared a variable that holds an array of strings. To insert values to it, you can place the values in a comma-separated list, inside curly brackets:

```
String[] cars = {"Volvo", "BMW", "Ford", "Mazda"};
```

To create an array of integers, you could write:

```
int[] myNum = {10, 20, 30, 40};
```

You can access an array element by referring to the index number.

```
String[] cars = {"Volvo", "BMW", "Ford", "Mazda"};
System.out.println(cars[0]);
```

You can loop through the array elements with the for loop and use the length property to specify how many times the loop should run.

Example

```
String[] cars = {"Volvo", "BMW", "Ford", "Mazda"};
for (int i = 0; i < cars.length; i++) {
   System.out.println(cars[i]);
}
```

There is also a "**for-each**" loop, which is used exclusively to loop through elements in arrays:

Syntax

for (*type variable* : *arrayname*) {

...

}

Example

```
String[] cars = {"Volvo", "BMW", "Ford", "Mazda"};
for (String i : cars) {
    System.out.println(i);
}
```

Multidimensional Arrays

A multidimensional array is an array of arrays. To create a two-dimensional array, add each array within its own set of **curly brackets**:

```
int[][] myNumbers = { {1, 2, 3, 4}, {5, 6, 7} };
```

myNumbers is now an array with two arrays as its elements.

To access the elements of the **myNumbers** array, specify two indexes: one for the array, and one for the element inside that array. This example accesses the third element (2) in the second array (1) of myNumbers:

Example

```
int[][] myNumbers = { {1, 2, 3, 4}, {5, 6, 7} };
int x = myNumbers[1][2];
System.out.println(x);
```

We can also use a for loop inside another for loop to get the elements of a two-dimensional array (we still must point to the two indexes):

Example

```
public class Main {
        public static void main(String[] args) {
          int[][] myNumbers = { {1, 2, 3, 4}, {5, 6, 7} };
          for (int i = 0; i < myNumbers.length; ++i) {
            for(int j = 0; j < myNumbers[i].length; ++j) {
              System.out.println(myNumbers[i][j]);
            }
          }
        }
}
```

Array of objects

We use the Class_Name followed by a square bracket [] then object reference name to create an Array of Objects.

```
Class_Name[ ] objectArrayReference;
```

Alternatively, we can also declare an Array of Objects as :

```
Class_Name objectArrayReference[ ];
```

Both the above declarations imply that objectArrayReference is an array of objects.

For example, if you have a class Student then we can create an array of Student objects as given below:

```
Student[ ] studentObjects;
```

Or

```
Student studentObjects[];
```

Instantiate the array of objects –

Syntax:

```
Class_Name obj[ ]= new Class_Name[Array_Length];
```

For example, if you have a class Student, and we want to declare and instantiate an array of Student objects with two objects/object references then it will be written as:

```
Student[ ] studentObjects = new Student[2];
```

```
// Java program to demonstrate initializing
// an array of objects using a method

class GFG {

    public static void main(String args[])
    {

        // Declaring an array of student
        Student[] arr;

        // Allocating memory for 2 objects
        // of type student
        arr = new Student[2];

        // Creating actual student objects
        arr[0] = new Student();
        arr[1] = new Student();

        // Assigning data to student objects
        arr[0].setData(1701289270, "Sara");
        arr[1].setData(1701289219, "Ahmed");

        // Displaying the student data
        System.out.println(
            "Student data in student arr 0: ");
```

```
        arr[0].display();

        System.out.println(
            "Student data in student arr 1: ");
        arr[1].display();
    }
}

// Creating a Student class with
// id and name as a attributes
class Student {

    public int id;
    public String name;

    // Method to set the data to
    // student objects
    public void setData(int id, String name)
    {
        this.id = id;
        this.name = name;
    }

    // display() method to display
    // the student data
    public void display()
    {
        System.out.println("Student id is: " + id + " "
                           + "and Student name is: "
                           + name);
        System.out.println();
    }
}
```

Output

Student data in student arr 0:

Student id is: 1701289270 and Student name is: Sara

Student data in student arr 1:

Student id is: 1701289219 and Student name is: Ahmed