Data Types in Java

Data types specify the different sizes and values that can be stored in the variable. There are two types of data types in Java:

1. **Primitive data types:** The primitive data types include boolean, char, byte, short, int, long, float and double.
2. **Non-primitive data types:** The non-primitive data types include Classes, Interfaces, and Arrays.

Java Primitive Data Types

In Java language, primitive data types are the building blocks of data manipulation. These are the most basic data types available in Java language.

Java is a statically-typed programming language. It means, all variables must be declared before its use. That is why we need to declare variable's type and name.

There are 8 types of primitive data types:

- boolean data type
- byte data type
- char data type
- short data type
- int data type
- long data type
- float data type
- double data type

Data Type	Default Value	Default size
boolean	False	1 bit
Char	'\u0000'	2 byte
Byte	0	1 byte
short	0	2 byte
Int	0	4 byte
Long	0L	8 byte
Float	0.0f	4 byte
double	0.0d	8 byte

The Boolean data type is used to store only two possible values: true and false. This data type is used for simple flags that track true/false conditions.

The Boolean data type specifies one bit of information, but its "size" can't be defined precisely.

Example: Boolean one = false

Byte Data Type

The byte data type is an example of primitive data type. It isan 8-bit signed two's complement integer. Its value-range lies between -128 to 127 (inclusive). Its minimum value is -128 and maximum value is 127. Its default value is 0.

The byte data type is used to save memory in large arrays where the memory savings is most required. It saves space because a byte is 4 times smaller than an integer. It can also be used in place of "int" data type.

Example: byte a = 10, byte b = -20

Short Data Type

The short data type is a 16-bit signed two's complement integer. Its value-range lies between -32,768 to 32,767 (inclusive). Its minimum value is -32,768 and maximum value is 32,767. Its default value is 0.

The short data type can also be used to save memory just like byte data type. A short data type is 2 times smaller than an integer.

Example: short s = 10000, short r = -5000

Int Data Type

The int data type is a 32-bit signed two's complement integer. Its value-range lies between - 2,147,483,648 (-2^31) to 2,147,483,647 (2^31 -1) (inclusive). Its minimum value is - 2,147,483,648and maximum value is 2,147,483,647. Its default value is 0.

The int data type is generally used as a default data type for integral values unless if there is no problem about memory.

Example: int a = 100000, int b = -200000

Long Data Type

The long data type is a 64-bit two's complement integer. Its value-range lies between -9,223,372,036,854,775,808(-2^63) to 9,223,372,036,854,775,807(2^63 -1)(inclusive). Its minimum value is -9,223,372,036,854,775,808and maximum value is 9,223,372,036,854,775,807. Its default value is 0. The long data type is used when you need a range of values more than those provided by int.

Example: long a = 100000L, long b = -200000L

Float Data Type

The float data type is a single-precision 32-bit IEEE 754 floating point.Its value range is unlimited. It is recommended to use a float (instead of double) if you need to save memory in large arrays of floating point numbers. The float data type should never be used for precise values, such as currency. Its default value is 0.0F.

Example: float f1 = 234.5f

Double Data Type

The double data type is a double-precision 64-bit IEEE 754 floating point. Its value range is unlimited. The double data type is generally used for decimal values just like float. The double data type also should never be used for precise values, such as currency. Its default value is 0.0d.

Example: double d1 = 12.3

Char Data Type

The char data type is a single 16-bit Unicode character. Its value-range lies between '\u0000' (or 0) to '\uffff' (or 65,535 inclusive).The char data type is used to store characters.

Example: char letterA = 'A'

Why char uses 2 byte in java and what is \u0000 ?

It is because java uses Unicode system not ASCII code system. The \u0000 is the lowest range of Unicode system. To get detail explanation about Unicode visit next page.

Java Variables

A variable is a container which holds the value while the java program is executed. A variable is assigned with a datatype.

Variable is a name of memory location. There are three types of variables in java: local, instance and static.

There are two types of data types in java: primitive and non-primitive.

Variable

Variable is name of *reserved area allocated in memory*. In other words, it is a *name of memory location*. It is a combination of "vary + able" that means its value can be changed.

1. int data=50;//Here data is variable

Types of Variables

There are three types of variables in java:

- local variable
- instance variable
- static variable

1) Local Variable

A variable declared inside the body of the method is called local variable. You can use this variable only within that method and the other methods in the class aren't even aware that the variable exists.

A local variable cannot be defined with "static" keyword.

2) Instance Variable

A variable declared inside the class but outside the body of the method, is called instance variable. It is not declared as static.

It is called instance variable because its value is instance specific and is not shared among instances.

3) Static variable

A variable which is declared as static is called static variable. It cannot be local. You can create a single copy of static variable and share among all the instances of the class. Memory allocation for static variable happens only once when the class is loaded in the memory.

Example to understand the types of variables in java

```
1. class A{
2. int data=50;//instance variable
3. static int m=100;//static variable
4. void method(){
5. int n=90;//local variable
6. }
7. }//end of class
```

Java Variable Example: Add Two Numbers

```
class Simple{

public static void main(String[] args){

int a=10;

int b=10;

int c=a+b;

System.out.println(c);

}}
```

Output:

```
20
```

Java Variable Example: Widening

```
class Simple{
public static void main(String[] args){
int a=10;
float f=a;
System.out.println(a);
System.out.println(f);
}}
```

Output:

```
10
10.0
```

Java Variable Example: Narrowing (Typecasting)

```
class Simple{
public static void main(String[] args){
float f=10.5f;
//int a=f;//Compile time error
int a=(int)f;
System.out.println(f);
System.out.println(a);
}}
```

Output:

```
10.5
10
```

Java Variable Example: Overflow

```
class Simple{
public static void main(String[] args){
//Overflow
int a=130;
byte b=(byte)a;
System.out.println(a);
System.out.println(b);
}}
```

Output:

```
130
-126
```

Java Variable Example: Adding Lower Type

```
class Simple{
public static void main(String[] args){
byte a=10;
byte b=10;
//byte c=a+b;//Compile Time Error: because a+b=20 will be int
byte c=(byte)(a+b);
System.out.println(c);
}}
```

Output:

```
20
```

Simple Java Program Structure

```
public class first
{

    public static void main(String[] args)
    {

        System.out.println("Hello World");

    }

}
```

Java Data Types

Data Type	Default Value	Default size
boolean	false	1 bit
char	'\u0000'	2 byte
byte	0	1 byte
short	0	2 byte
int	0	4 byte
long	0L	8 byte
float	0.0f	4 byte
double	0.0d	8 byte

Operators in Java

Operator Type	Category	Precedence
Unary	postfix	expr++ expr--
	prefix	++expr --expr +expr -expr ~ !
Arithmetic	multiplicative	* / %
	additive	+ -
Shift	shift	<< >> >>>
Relational	comparison	< > <= >= instanceof
	equality	== !=
Bitwise	bitwise AND	&
	bitwise exclusive OR	^
	bitwise inclusive OR	|
Logical	logical AND	&&
	logical OR	||
Ternary	ternary	? :
Assignment	assignment	= += -= *= /= %= &= ^= |= <<= >>= >>>=

Reading Input from User

```
import java.util.*;
public class first {

public static void main(String[] args) {
// TODO Auto-generated method stub

System.out.println("Enter your name");
Scanner test= new Scanner(System.in);
String name=test.nextLine();
System.out.println(name);

}

}
```

If else Statement

- If
- If else
- If else ladder
- Nested if else statement

Syntax - if

```
if(condition)
{
  //code to be executed
 }
```

Syntax – if else

```
if(condition)
{
  //Statements that will be executed if condition is
true
}else
{
  //Statements that will be executed if condition is
false
}
```

Nested If else

```
if(condition1)
  {
  //execute statements if condition1 is true
  }else if(condition2){
  //statements to be executed if condition2 is true
  }
  else if(condition3){
  //statements to be executed if condition3 is true
  }
  ...
  else{
  //statements to be executed if all the conditions are false
  }
```

Switch Statement Syntax

```
switch(expression)
{
    case value1:
     //statements;
     break;
    case value2:
     //statements;
     break;
    .....

    default:
     Code to be executed
    Break;
}
```

Loops in Java

- For
- While
- Do while

Syntax For Loop

```
for(initialization;condition;incr/decr)
{
  //statements
}
```

Loops in Java

```
public class hello {

public static void main(String[] args) {
// TODO Auto-generated method stub

for(int i=0;i<5;i++)
{
System.out.println("Hello");
}

}

}
```

While loop Syntax

```
while(condition)
{
    //statements
}
```

While Loop

```
public class hello {

public static void main(String[] args) {
// TODO Auto-generated method stub

int i=0;
while(i<5)
{
System.out.println("Hello");
i++;
}

}

}
```

Do While Loop Syntax

```
do
{
  //statements

}while(condition);
```

Do while loop

```
public class hello {

public static void main(String[] args)
{
// TODO Auto-generated method stub

   int i = 0;
   do
   {
      System.out.println("Hello");
      i++;
   } while (i < 5);

}

}
```

Arrays in Java

- An array is a container that holds data (values) **of one single type**.
- For example, you can create an array that can hold 100 values of int type.

Array Declaration

```
int[] arrayname = new int[5];
```

Initializing an array

```
int[] test = {1,2,3,4,5};
```

Multidimensional arrays in Java

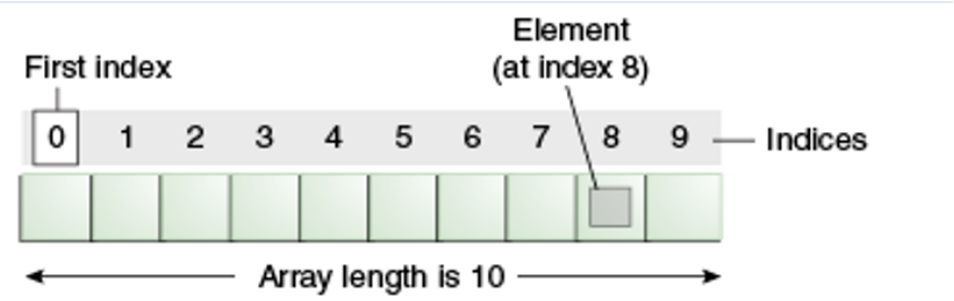
```
int[][] test = {{1,1},{2,2}};
```

Java Array

Normally, an array is a collection of similar type of elements that have a contiguous memory location.

Java array is an object which contains elements of a similar data type. It is a data structure where we store similar elements. We can store only a fixed set of elements in a Java array.

Array in java is index-based, the first element of the array is stored at the 0 index.



Advantages

- **Code Optimization:** It makes the code optimized, we can retrieve or sort the data efficiently.
- **Random access:** We can get any data located at an index position.

Disadvantages

- **Size Limit:** We can store only the fixed size of elements in the array. It doesn't grow its size at runtime. To solve this problem, collection framework is used in Java which grows automatically.

Types of Array in java

There are two types of array.

- Single Dimensional Array
- Multidimensional Array

Single Dimensional Array in Java

Syntax to Declare an Array in Java

1. dataType[] arr; (or)
2. dataType []arr; (or)
3. dataType arr[];

Instantiation of an Array in Java

1. arrayRefVar=new datatype[size];

Example of Java Array

Let's see the simple example of java array, where we are going to declare, instantiate, initialize and traverse an array.

1. //Java Program to illustrate how to declare, instantiate, initialize
2. //and traverse the Java array.

```
class Testarray{

public static void main(String args[]){

int a[]=new int[5];//declaration and instantiation

a[0]=10;//initialization

a[1]=20;

a[2]=70;

a[3]=40;

a[4]=50;

//traversing array

for(int i=0;i<a.length;i++)//length is the property of array

System.out.println(a[i]);

}}
```

Output:

```
10
20
70
40
50
```

Declaration, Instantiation and Initialization of Java Array

We can declare, instantiate and initialize the java array together by:

```
1. int a[]={33,3,4,5};//declaration, instantiation and initialization
```

Let's see the simple example to print this array.

```
1. //Java Program to illustrate the use of declaration, instantiation
2. //and initialization of Java array in a single line
3. class Testarray1{
4. public static void main(String args[]){
5. int a[]={33,3,4,5};//declaration, instantiation and initialization
6. //printing array
7. for(int i=0;i<a.length;i++)//length is the property of array
8. System.out.println(a[i]);
9. }}
```

Output:

```
33
3
4
5
```

Passing Array to Method in Java

We can pass the java array to method so that we can reuse the same logic on any array.

Let's see the simple example to get the minimum number of an array using a method.

```
1. //Java Program to demonstrate the way of passing an array
2. //to method.
3. class Testarray2{
4. //creating a method which receives an array as a parameter
5. static void min(int arr[]){
6. int min=arr[0];
7. for(int i=1;i<arr.length;i++)
8.  if(min>arr[i])
9.   min=arr[i];
10.
```

```
System.out.println(min);
}

public static void main(String args[]){
int a[]={33,3,4,5};//declaring and initializing an array
min(a);//passing array to method
}}
```

Output:

```
3
```

Anonymous Array in Java

Java supports the feature of an anonymous array, so you don't need to declare the array while passing an array to the method.

```
//Java Program to demonstrate the way of passing an anonymous array
//to method.
public class TestAnonymousArray{
//creating a method which receives an array as a parameter
static void printArray(int arr[]){
for(int i=0;i<arr.length;i++)
System.out.println(arr[i]);
}

public static void main(String args[]){
printArray(new int[]{10,22,44,66});//passing anonymous array to method
}}
```

Output:

```
10
22
44
66
```

Returning Array from the Method

We can also return an array from the method in Java.

```
//Java Program to return an array from the method
class TestReturnArray{
//creating method which returns an array
static int[] get(){
return new int[]{10,30,50,90,60};
}

public static void main(String args[]){
//calling method which returns an array
int arr[]=get();
//printing the values of an array
```

```
12. for(int i=0;i<arr.length;i++)
13. System.out.println(arr[i]);
14. }}
```

Output:

```
10
30
50
90
60
```

ArrayIndexOutOfBoundsException

The Java Virtual Machine (JVM) throws an ArrayIndexOutOfBoundsException if length of the array in negative, equal to the array size or greater than the array size while traversing the array.

```
1. //Java Program to demonstrate the case of
2. //ArrayIndexOutOfBoundsException in a Java Array.
3. public class TestArrayException{
4. public static void main(String args[]){
5. int arr[]={50,60,70,80};
6. for(int i=0;i<=arr.length;i++){
7. System.out.println(arr[i]);
8. }
9. }}
```

Output:

```
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: 4
      at TestArrayException.main(TestArrayException.java:5)
50
60
70
80
```

Multidimensional Array in Java

In such case, data is stored in row and column based index (also known as matrix form).

Syntax to Declare Multidimensional Array in Java

```
1. dataType[][] arrayRefVar; (or)
2. dataType [][]arrayRefVar; (or)
3. dataType arrayRefVar[][]; (or)
4. dataType []arrayRefVar[];
```

Example to instantiate Multidimensional Array in Java

```
1. int[][] arr=new int[3][3];//3 row and 3 column
```

Example to initialize Multidimensional Array in Java

```
1. arr[0][0]=1;
2. arr[0][1]=2;
3. arr[0][2]=3;
4. arr[1][0]=4;
5. arr[1][1]=5;
6. arr[1][2]=6;
7. arr[2][0]=7;
8. arr[2][1]=8;
9. arr[2][2]=9;
```

Example of Multidimensional Java Array

Let's see the simple example to declare, instantiate, initialize and print the 2Dimensional array.

```
1. //Java Program to illustrate the use of multidimensional array
2. class Testarray3{
3. public static void main(String args[]){
4. //declaring and initializing 2D array
5. int arr[][]={{1,2,3},{2,4,5},{4,4,5}};
6. //printing 2D array
7. for(int i=0;i<3;i++){
8.  for(int j=0;j<3;j++){
9.    System.out.print(arr[i][j]+" ");
10. }
11. System.out.println();
12. }
13. }}
```

Output:

```
1 2 3
2 4 5
4 4 5
```

Jagged Array in Java

If we are creating odd number of columns in a 2D array, it is known as a jagged array. In other words, it is an array of arrays with different number of columns.

```
//Java Program to illustrate the jagged array
class TestJaggedArray{
    public static void main(String[] args){
        //declaring a 2D array with odd columns
        int arr[][] = new int[3][];
        arr[0] = new int[3];
        arr[1] = new int[4];
        arr[2] = new int[2];
        //initializing a jagged array
        int count = 0;
        for (int i=0; i<arr.length; i++)
            for(int j=0; j<arr[i].length; j++)
                arr[i][j] = count++;

        //printing the data of a jagged array
        for (int i=0; i<arr.length; i++){
            for (int j=0; j<arr[i].length; j++){
                System.out.print(arr[i][j]+" ");
            }
            System.out.println();//new line
        }
    }
}
```

Output:

```
0 1 2
3 4 5 6
7 8
```

What is the class name of Java array?

In Java, an array is an object. For array object, a proxy class is created whose name can be obtained by getClass().getName() method on the object.

```
//Java Program to get the class name of array in Java

class Testarray4{

public static void main(String args[]){

//declaration and initialization of array

int arr[]={4,4,5};

//getting the class name of Java array

Class c=arr.getClass();

String name=c.getName();

//printing the class name of Java array

System.out.println(name);



}}
```

Output:

```
I
```

Copying a Java Array

We can copy an array to another by the arraycopy() method of System class.

Syntax of arraycopy method

```
1. public static void arraycopy(
2. Object src, int srcPos,Object dest, int destPos, int length
3. )
```

Example of Copying an Array in Java

```
1. //Java Program to copy a source array into a destination array in Java
2. class TestArrayCopyDemo {
3.     public static void main(String[] args) {
4.         //declaring a source array
5.         char[] copyFrom = { 'd', 'e', 'c', 'a', 'f', 'f', 'e',
6.                 'i', 'n', 'a', 't', 'e', 'd' };
7.         //declaring a destination array
8.         char[] copyTo = new char[7];
9.         //copying array using System.arraycopy() method
10.        System.arraycopy(copyFrom, 2, copyTo, 0, 7);
11.        //printing the destination array
12.        System.out.println(String.valueOf(copyTo));
13.    }
14. }
```

Output:

```
caffein
```

Addition of 2 Matrices in Java

Let's see a simple example that adds two matrices.

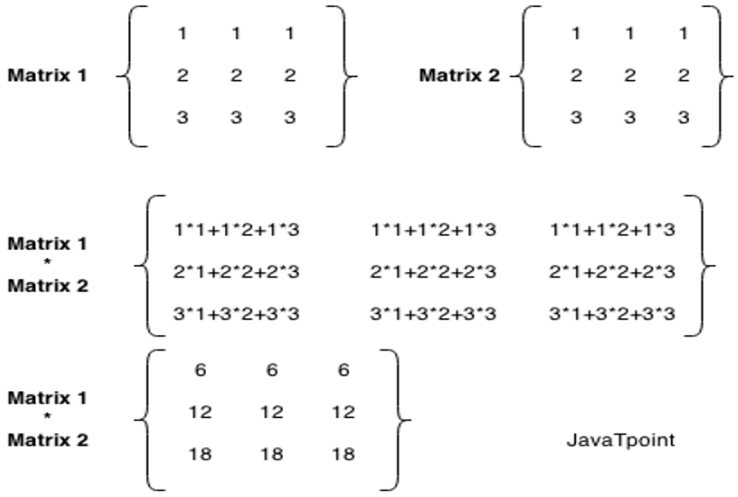
```
1. //Java Program to demonstrate the addition of two matrices in Java
2. class Testarray5{
3. public static void main(String args[]){
4. //creating two matrices
5. int a[][]={{1,3,4},{3,4,5}};
6. int b[][]={{1,3,4},{3,4,5}};
7.
8. //creating another matrix to store the sum of two matrices
9. int c[][]=new int[2][3];
10.
11. //adding and printing addition of 2 matrices
12. for(int i=0;i<2;i++){
13. for(int j=0;j<3;j++){
14. c[i][j]=a[i][j]+b[i][j];
15. System.out.print(c[i][j]+" ");
16. }
17. System.out.println();//new line
18. }
19.
20. }}
```

Output:

```
2 6 8
6 8 10
```

Multiplication of 2 Matrices in Java

In the case of matrix multiplication, a one-row element of the first matrix is multiplied by all the columns of the second matrix which can be understood by the image given below.



Let's see a simple example to multiply two matrices of 3 rows and 3 columns.

```
1. //Java Program to multiply two matrices
2. public class MatrixMultiplicationExample{
3. public static void main(String args[]){
4. //creating two matrices
5. int a[][]={{1,1,1},{2,2,2},{3,3,3}};
6. int b[][]={{1,1,1},{2,2,2},{3,3,3}};
7.
8. //creating another matrix to store the multiplication of two matrices
9. int c[][]=new int[3][3];  //3 rows and 3 columns
10.
11. //multiplying and printing multiplication of 2 matrices
12. for(int i=0;i<3;i++){
13. for(int j=0;j<3;j++){
14. c[i][j]=0;
15. for(int k=0;k<3;k++)
16. {
17. c[i][j]+=a[i][k]*b[k][j];
18. }//end of k loop
19. System.out.print(c[i][j]+" ");  //printing matrix element
20. }//end of j loop
21. System.out.println();//new line
22. }
23. }}
```

Output:

```
6 6 6
12 12 12
18 18 18
```

Java For-each Loop | Enhanced For Loop

The Java for-each loop or enhanced for loop is introduced since J2SE 5.0. It provides an alternative approach to traverse the array or collection in Java. It is mainly used to traverse the array or collection elements. The advantage of the for-each loop is that it eliminates the possibility of bugs and makes the code more readable. It is known as the for-each loop because it traverses each element one by one.

The drawback of the enhanced for loop is that it cannot traverse the elements in reverse order. Here, you do not have the option to skip any element because it does not work on an index basis. Moreover, you cannot traverse the odd or even elements only.

But, it is recommended to use the Java for-each loop for traversing the elements of array and collection because it makes the code readable.

Advantages

- It makes the code more readable.
- It eliminates the possibility of programming errors.

Syntax

The syntax of Java for-each loop consists of data_type with the variable followed by a colon (:), then array or collection.

```
for(data_type variable : array | collection){
//body of for-each loop
}
```

How it works?

The Java for-each loop traverses the array or collection until the last element. For each element, it stores the element in the variable and executes the body of the for-each loop.

For-each loop Example: Traversing the array elements

```
//An example of Java for-each loop
class ForEachExample1{
 public static void main(String args[]){
  //declaring an array
  int arr[]={12,13,14,44};
  //traversing the array with for-each loop
  for(int i:arr){
    System.out.println(i);
  }
 }
}
```

2.

Test it Now

Output:

```
12
12
14
44
```

Let us see another of Java for-each loop where we are going to total the elements.

```
1.  class ForEachExample1{
2.    public static void main(String args[]){
3.     int arr[]={12,13,14,44};
4.     int total=0;
5.     for(int i:arr){
6.       total=total+i;
7.     }
8.    System.out.println("Total: "+total);
9.   }
10. }
```

Output:

```
Total: 83
```

Java String

In Java, string is basically an object that represents sequence of char values. An array of characters works same as Java string. For example:

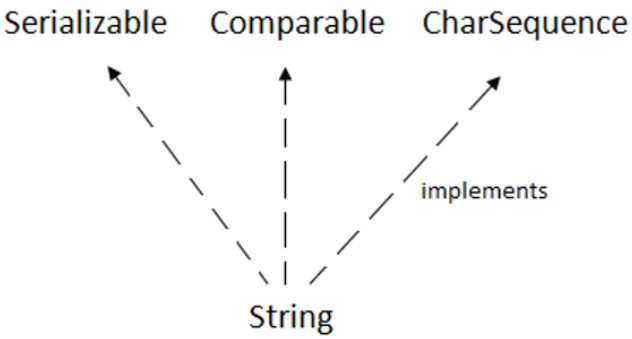
```
1. char[] ch={'j','a','v','a','t','p','o','i','n','t'};
2. String s=new String(ch);
```

is same as:

```
1. String s="javatpoint";
```

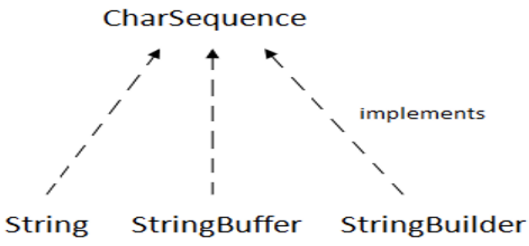
Java String class provides a lot of methods to perform operations on string such as compare(), concat(), equals(), split(), length(), replace(), compareTo(), intern(), substring() etc.

The java.lang.String class implements *Serializable*, *Comparable* and *CharSequence* interfaces.



CharSequence Interface

The CharSequence interface is used to represent the sequence of characters. String, StringBuffer and StringBuilder classes implement it. It means, we can create strings in java by using these three classes.



The Java String is immutable which means it cannot be changed. Whenever we change any string, a new instance is created. For mutable strings, you can use StringBuffer and StringBuilder classes.

We will discuss immutable string later. Let's first understand what is String in Java and how to create the String object.

What is String in java

Generally, String is a sequence of characters. But in Java, string is an object that represents a sequence of characters. The java.lang.String class is used to create a string object.

How to create a string object?

There are two ways to create String object:

1. By string literal
2. By new keyword

---

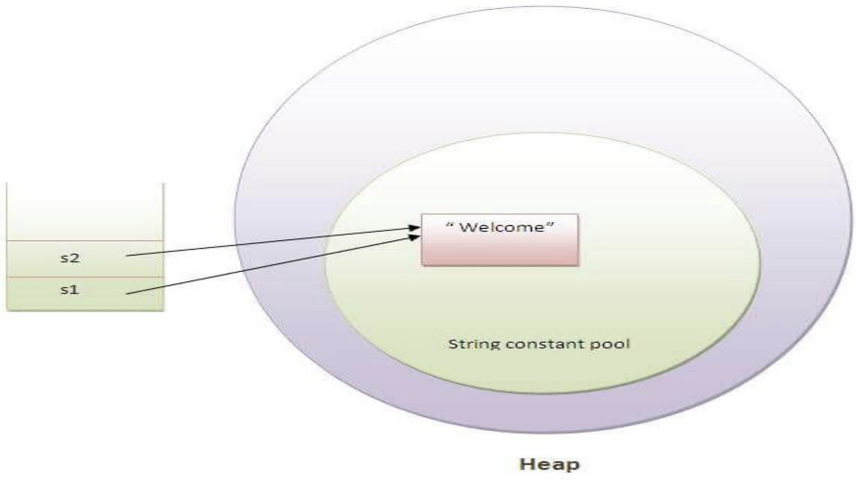
1) String Literal

Java String literal is created by using double quotes. For Example:

```
1. String s="welcome";
```

Each time you create a string literal, the JVM checks the "string constant pool" first. If the string already exists in the pool, a reference to the pooled instance is returned. If the string doesn't exist in the pool, a new string instance is created and placed in the pool. For example:

```
1. String s1="Welcome";
2. String s2="Welcome";//It doesn't create a new instance
```



In the above example, only one object will be created. Firstly, JVM will not find any string object with the value "Welcome" in string constant pool, that is why it will create a new object. After that it will find the string with the value "Welcome" in the pool, it will not create a new object but will return the reference to the same instance.

Note: String objects are stored in a special memory area known as the "string constant pool".

Why Java uses the concept of String literal?

To make Java more memory efficient (because no new objects are created if it exists already in the string constant pool).

---

2) By new keyword

```
1. String s=new String("Welcome");//creates two objects and one reference variable
```

In such case, JVM will create a new string object in normal (non-pool) heap memory, and the literal "Welcome" will be placed in the string constant pool. The variable s will refer to the object in a heap (non-pool).

Java String Example

```
1. public class StringExample{
2. public static void main(String args[]){
3. String s1="java";//creating string by java string literal
4. char ch[]={'s','t','r','i','n','g','s'};
5. String s2=new String(ch);//converting char array to string
6. String s3=new String("example");//creating java string by new keyword
7. System.out.println(s1);
8. System.out.println(s2);
9. System.out.println(s3);
10. }}
```

```
java
strings
example
```

Java String class methods

The java.lang.String class provides many useful methods to perform operations on sequence of char values.

The java.lang.String class provides a lot of methods to work on string. By the help of these methods, we can perform operations on string such as trimming, concatenating, converting, comparing, replacing strings etc.

Java String is a powerful concept because everything is treated as a string if you submit any form in window based, web based or mobile application.

Let's see the important methods of String class.

No.	Method	Description
1	char charAt(int index)	returns char value for the particular index
2	int length()	returns string length
3	static String format(String format, Object... args)	returns a formatted string.
4	static String format(Locale l, String format, Object... args)	returns formatted string with given locale.
5	String substring(int beginIndex)	returns substring for given begin index.
6	String substring(int beginIndex, int endIndex)	returns substring for given begin index and end index.
7	boolean contains(CharSequence s)	returns true or false after matching the sequence of char value.
8	static String join(CharSequence delimiter, CharSequence... elements)	returns a joined string.
9	static String join(CharSequence delimiter, Iterable<? extends CharSequence> elements)	returns a joined string.
10	boolean equals(Object another)	checks the equality of string with the given object.
11	boolean isEmpty()	checks if string is empty.
12	String concat(String str)	concatenates the specified string.
13	String replace(char old, char new)	replaces all occurrences of the specified char value.

14	String replace(CharSequence old, CharSequence new)	replaces all occurrences of the specified CharSequence.
15	static String equalsIgnoreCase(String another)	compares another string. It doesn't check case.
16	String[] split(String regex)	returns a split string matching regex.
17	String[] split(String regex, int limit)	returns a split string matching regex and limit.
18	String intern()	returns an interned string.
19	int indexOf(int ch)	returns the specified char value index.
20	int indexOf(int ch, int fromIndex)	returns the specified char value index starting with given index.
21	int indexOf(String substring)	returns the specified substring index.
22	int indexOf(String substring, int fromIndex)	returns the specified substring index starting with given index.
23	String toLowerCase()	returns a string in lowercase.
24	String toLowerCase(Locale l)	returns a string in lowercase using specified locale.
25	String toUpperCase()	returns a string in uppercase.
26	String toUpperCase(Locale l)	returns a string in uppercase using specified locale.
27	String trim()	removes beginning and ending spaces of this string.
28	static String valueOf(int value)	converts given type into string. It is an overloaded method.

Java String format()

The **java string format()** method returns the formatted string by given locale, format and arguments.

If you don't specify the locale in String.format() method, it uses default locale by calling *Locale.getDefault()* method.

The format() method of java language is like *sprintf()* function in c language and *printf()* method of java language.

Internal implementation

```
1.  public static String format(String format, Object... args) {
2.          return new Formatter().format(format, args).toString();
3.      }
```

Signature

There are two type of string format() method:

1. **public static** String format(String format, Object... args)
2. and,
3. **public static** String format(Locale locale, String format, Object... args)

Parameters

locale : specifies the locale to be applied on the format() method.

format : format of the string.

args : arguments for the format string. It may be zero or more.

Returns

formatted string

Throws

NullPointerException : if format is null.

IllegalFormatException : if format is illegal or incompatible.

Java String format() method example

```
1. public class FormatExample{
2. public static void main(String args[]){
3. String name="sonoo";
4. String sf1=String.format("name is %s",name);
5. String sf2=String.format("value is %f",32.33434);
6. String sf3=String.format("value is %32.12f",32.33434);//returns 12 char fractional part filling with 0
7.   System.out.println(sf1);
8. System.out.println(sf2);
9. System.out.println(sf3);
10. }}
```

Test it Now

```
name is sonoo
value is 32.334340
value is                 32.334340000000
```

Format Specifier	Data Type	Output
%a	floating point (except *BigDecimal*)	Returns Hex output of floating point number.
%b	Any type	"true" if non-null, "false" if null
%c	character	Unicode character
%d	integer (incl. byte, short, int, long, bigint)	Decimal Integer
%e	floating point	decimal number in scientific notation
%f	floating point	decimal number
%g	floating point	decimal number, possibly in scientific notation depending on the precision and value.
%h	any type	Hex String of value from hashCode() method.
%n	none	Platform-specific line separator.
%o	integer (incl. byte, short, int, long, bigint)	Octal number
%s	any type	String value
%t	Date/Time (incl. long, Calendar, Date and TemporalAccessor)	%t is the prefix for Date/Time conversions. More formatting flags are needed after this. See Date/Time conversion below.
%x	integer (incl. byte, short, int, long, bigint)	Hex string.

Java String join()

The **java string join()** method returns a string joined with given delimiter. In string join method, delimiter is copied for each elements.

In case of null element, "null" is added. The join() method is included in java string since JDK 1.8.

There are two types of join() methods in java string.

Signature

The signature or syntax of string join method is given below:

1. **public static** String join(CharSequence delimiter, CharSequence... elements)
2. and
3. **public static** String join(CharSequence delimiter, Iterable<? **extends** CharSequence> elements)

Parameters

delimiter : char value to be added with each element

elements : char value to be attached with delimiter

Returns

joined string with delimiter

Java String join() method example

```
1. public class StringJoinExample{
2. public static void main(String args[]){
3. String joinString1=String.join("-","welcome","to","javatpoint");
4. System.out.println(joinString1);
5. }}
```

Test it Now

```
welcome-to-javatpoint
```

Java String join() Method Example 2

We can use delimeter to format the string as we did in the below example to show date and time.

```
1.  public class StringJoinExample2 {
2.      public static void main(String[] args) {
3.          String date = String.join("/","25","06","2018");
4.          System.out.print(date);
5.          String time = String.join(":", "12","10","10");
6.          System.out.println(" "+time);
7.      }
8.  }
```

```
25/06/2018 12:10:10
```

Java String split()

The **java string split()** method splits this string against given regular expression and returns a char array.

Signature

There are two signature for split() method in java string.

```
1.  public String split(String regex)
2.  and,
3.  public String split(String regex, int limit)
```

Parameter

regex : regular expression to be applied on string.

limit : limit for the number of strings in array. If it is zero, it will returns all the strings matching regex.

Java String split() method example

The given example returns total number of words in a string excluding space only. It also includes special characters.

```
public class SplitExample{
public static void main(String args[]){
String s1="java string split method by javatpoint";
String[] words=s1.split("\\s");//splits the string based on whitespace
//using java foreach loop to print elements of string array
for(String w:words){
System.out.println(w);
}
}}
```

Test it Now

```
java
string
split
method
by
javatpoint
```

Java String split() method with regex and length example

```
1. public class SplitExample2{
2. public static void main(String args[]){
3. String s1="welcome to split world";
4. System.out.println("returning words:");
5. for(String w:s1.split("\\s",0)){
6. System.out.println(w);
7. }
8. System.out.println("returning words:");
9. for(String w:s1.split("\\s",1)){
10. System.out.println(w);
11. }
12. System.out.println("returning words:");
13. for(String w:s1.split("\\s",2)){
14. System.out.println(w);
15. }
16.
17. }}
```

Test it Now

```
returning words:
welcome
to
split
world
returning words:
welcome to split world
returning words:
welcome
to split world
```

Java String intern()

The **java string intern()** method returns the interned string. It returns the canonical representation of string.

It can be used to return string from memory, if it is created by new keyword. It creates exact copy of heap string object in string constant pool.

Java String intern() method example

```
1. public class InternExample{
2. public static void main(String args[]){
3. String s1=new String("hello");
4. String s2="hello";
5. String s3=s1.intern();//returns string from pool, now it will be same as s2
6. System.out.println(s1==s2);//false because reference variables are pointing to different instance
7. System.out.println(s2==s3);//true because reference variables are pointing to same instance
8. }}
```

Test it Now

```
false
true
```

Java String intern() Method Example 2

Let's see one more example to understand the string intern concept.

```
1. public class InternExample2 {
2.     public static void main(String[] args) {
3.         String s1 = "Javatpoint";
4.         String s2 = s1.intern();
5.         String s3 = new String("Javatpoint");
6.         String s4 = s3.intern();
7.         System.out.println(s1==s2); // True
8.         System.out.println(s1==s3); // False
9.         System.out.println(s1==s4); // True
10.        System.out.println(s2==s3); // False
11.        System.out.println(s2==s4); // True
12.        System.out.println(s3==s4); // False
13.    }
14. }
```

Test it Now

```
true
false
true
false
true
false
```

Java String toUpperCase() and toLowerCase() method

The java string toUpperCase() method converts this string into uppercase letter and string toLowerCase() method into lowercase letter.

```
1. String s="Sachin";
2. System.out.println(s.toUpperCase());//SACHIN
3. System.out.println(s.toLowerCase());//sachin
4. System.out.println(s);//Sachin(no change in original)
```

Output

```
SACHIN
sachin
Sachin
```

Java String trim() method

The string trim() method eliminates white spaces before and after string.

```
1. String s="  Sachin  ";
2. System.out.println(s);//  Sachin
3. System.out.println(s.trim());//Sachin
```

Output

```
Sachin
Sachin
```

Java String startsWith() and endsWith() method

```
1. String s="Sachin";
2.  System.out.println(s.startsWith("Sa"));//true
3.  System.out.println(s.endsWith("n"));//true
```

Output

```
true
true
```

Java String charAt() method

The string charAt() method returns a character at specified index.

```
1. String s="Sachin";
2. System.out.println(s.charAt(0));//S
3. System.out.println(s.charAt(3));//h
```

Output

```
S
h
```

Java String length() method

The string length() method returns length of the string.

```
1. String s="Sachin";
2. System.out.println(s.length());//6
```

Output

```
6
```

Java String intern() method

A pool of strings, initially empty, is maintained privately by the class String.

When the intern method is invoked, if the pool already contains a string equal to this String object as determined by the equals(Object) method, then the string from the pool is returned. Otherwise, this String object is added to the pool and a reference to this String object is returned.

```
1. String s=new String("Sachin");
2. String s2=s.intern();
3. System.out.println(s2);//Sachin
```

Output

```
Sachin
```

Java String valueOf() method

The string valueOf() method coverts given type such as int, long, float, double, boolean, char and char array into string.

```
1. int a=10;
2. String s=String.valueOf(a);
3. System.out.println(s+10);
```

Output:

```
1010
```

Java String replace() method

The string replace() method replaces all occurrence of first sequence of character with second sequence of character.

```
1. String s1="Java is a programming language. Java is a platform. Java is an Island.";
2. String replaceString=s1.replace("Java","Kava");//replaces all occurrences of "Java" to "Kava"
3. System.out.println(replaceString);
```

Output:

```
Kava is a programming language. Kava is a platform. Kava is an Island.
```