# Java String

In [Java](https://www.javatpoint.com/java-tutorial), string is basically an object that represents sequence of char values. An [array](https://www.javatpoint.com/array-in-java) 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 strings such as compare(), concat(), equals(), split(), length(), replace(), compareTo(), intern(), substring() etc.

Difference between StringBuffer and StringBuilder

Java provides three classes to represent a sequence of characters: String, StringBuffer, and StringBuilder. The String class is an immutable class whereas StringBuffer and StringBuilder classes are mutable. There are many differences between StringBuffer and StringBuilder. The StringBuilder class is introduced since JDK 1.5.

A list of differences between StringBuffer and StringBuilder are given below:

|  |  |  |
| --- | --- | --- |
| **No.** | **StringBuffer** | **StringBuilder** |
| 1) | StringBuffer is *synchronized* i.e. thread safe. It means two threads can't call the methods of StringBuffer simultaneously. | StringBuilder is *non-synchronized* i.e. not thread safe. It means two threads can call the methods of StringBuilder simultaneously. |
| 2) | StringBuffer is *less efficient* than StringBuilder. | StringBuilder is *more efficient* than StringBuffer. |

1. StringBuffer buffer=**new** StringBuffer("hello");
2. buffer.append("java");
3. System.out.println(buffer);
4. StringBuilder builder=**new** StringBuilder("hello");
5. builder.append("java");
6. System.out.println(builder);

# Wrapper classes in Java

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

Since J2SE 5.0, **autoboxing** and **unboxing** feature convert primitives into objects and objects into primitives automatically. The automatic conversion of primitive into an object is known as autoboxing and vice-versa unboxing.

## **Use of Wrapper classes in Java**

Java is an object-oriented programming language, so we need to deal with objects many times like in Collections, Serialization, Synchronization, etc

The eight classes of the java.lang package are known as wrapper classes in Java. The list of eight wrapper classes are given below:

|  |  |
| --- | --- |
| **Primitive Type** | **Wrapper class** |
| boolean | [Boolean](https://www.javatpoint.com/java-boolean) |
| char | [Character](https://www.javatpoint.com/post/java-character) |
| byte | [Byte](https://www.javatpoint.com/java-byte) |
| short | [Short](https://www.javatpoint.com/java-short) |
| int | [Integer](https://www.javatpoint.com/java-integer) |
| long | [Long](https://www.javatpoint.com/java-long) |
| float | [Float](https://www.javatpoint.com/java-float) |
| double | [Double](https://www.javatpoint.com/java-double) |

## **Autoboxing**

The automatic conversion of primitive data type into its corresponding wrapper class is known as autoboxing, for example, byte to Byte, char to Character, int to Integer, long to Long, float to Float, boolean to Boolean, double to Double, and short to Short.

Since Java 5, we do not need to use the valueOf() method of wrapper classes to convert the primitive into objects.

**Wrapper class Example: Primitive to Wrapper**

1. //Java program to convert primitive into objects
2. //Autoboxing example of int to Integer
3. **public** **class** WrapperExample1{
4. **public** **static** **void** main(String args[]){
5. //Converting int into Integer
6. **int** a=20;
7. Integer i=Integer.valueOf(a);//converting int into Integer explicitly
8. Integer j=a;//autoboxing, now compiler will write Integer.valueOf(a) internally
10. System.out.println(a+" "+i+" "+j);
11. }}

## **Unboxing**

The automatic conversion of wrapper type into its corresponding primitive type is known as unboxing. It is the reverse process of autoboxing. Since Java 5, we do not need to use the intValue() method of wrapper classes to convert the wrapper type into primitives.

**Wrapper class Example: Wrapper to Primitive**

1. //Java program to convert object into primitives
2. //Unboxing example of Integer to int
3. **public** **class** WrapperExample2{
4. **public** **static** **void** main(String args[]){
5. //Converting Integer to int
6. Integer a=**new** Integer(3);
7. **int** i=a.intValue();//converting Integer to int explicitly
8. **int** j=a;//unboxing, now compiler will write a.intValue() internally
10. System.out.println(a+" "+i+" "+j);
11. }}

Output:

3 3 3

# Java toString() Method

If you want to represent any object as a string, **toString() method** comes into existence.

The toString() method returns the String representation of the object.

If you print any object, Java compiler internally invokes the toString() method on the object. So overriding the toString() method, returns the desired output, it can be the state of an object etc. depending on your implementation.

### **Advantage of Java toString() method**

By overriding the toString() method of the Object class, we can return values of the object, so we don't need to write much code.

39.9M

908

OOPs Concepts in Java

### **Understanding problem without toString() method**

Let's see the simple code that prints reference.

**Student.java**

1. **class** Student{
2. **int** rollno;
3. String name;
4. String city;
6. Student(**int** rollno, String name, String city){
7. **this**.rollno=rollno;
8. **this**.name=name;
9. **this**.city=city;
10. }
12. **public** **static** **void** main(String args[]){
13. Student s1=**new** Student(101,"Raj","lucknow");
14. Student s2=**new** Student(102,"Vijay","ghaziabad");
16. System.out.println(s1);//compiler writes here s1.toString()
17. System.out.println(s2);//compiler writes here s2.toString()
18. }
19. }

**Output:**

Student@1fee6fc

Student@1eed786

As you can see in the above example, printing s1 and s2 prints the hashcode values of the objects but I want to print the values of these objects. Since Java compiler internally calls toString() method, overriding this method will return the specified values. Let's understand it with the example given below:

## **Example of Java toString() method**

Let's see an example of toString() method.

**Student.java**

1. **class** Student{
2. **int** rollno;
3. String name;
4. String city;
6. Student(**int** rollno, String name, String city){
7. **this**.rollno=rollno;
8. **this**.name=name;
9. **this**.city=city;
10. }
12. **public** String toString(){//overriding the toString() method
13. **return** rollno+" "+name+" "+city;
14. }
15. **public** **static** **void** main(String args[]){
16. Student s1=**new** Student(101,"Raj","lucknow");
17. Student s2=**new** Student(102,"Vijay","ghaziabad");
19. System.out.println(s1);//compiler writes here s1.toString()
20. System.out.println(s2);//compiler writes here s2.toString()
21. }
22. }