



JAVA NOTES

String Handling & Library Classes

In Java, String is an object which contains a sequence of characters. String class is used to create and manipulate strings. The String class is available in java.lang package.

Declaration and Assigning a String

```
1. //Declaration:
2. String <variable>;
3. //Ex:
4. String str;
5. //Assigning:
6. <variable>=<String literal>;
7. //Ex:
8. str="Hello World!";
```

Input a String


```
1. //For a string without any space (For a single word):
2. <variable>=<Scanner object>.next();
3. Str=sc.next();
4. //For a String with spaces (For a sentence):
5. <variable>=<Scanner object>.nextLine();
6. Str=sc.nextLine();
```

String Functions

For all the below examples, **str="COMPUTER"**; **Output** will be displayed as a **single line comment (//)**.



Quick Tip

1. Function names start with lowercase and then the second word starts with uppercase letters. Eg: `indexOf()`;
2. Topics asked in board questions are marked with 



.length() (int)

This function is used to return the **length** of the string.

Syntax with example:

```
1. <int variable>=<string var>.length();  
2. int Len=str.length();  
3. //8
```

.charAt() (char)

This function returns the **character** from the given index.

Syntax with example:

```
1. <char variable>=<string var>.charAt(<index>);  
2. char ch=str.charAt(2);  
3. //0
```

.indexOf() (int)

This function returns the **index** of **first occurrence** of a character.

Syntax with example:

```
1. <int variable>=<string var>.indexOf(<character>);  
2. int idx=str.indexOf('M');  
3. //2
```

.indexOf(char ch, int start_index) (int)

This function returns the **index** of a given **character** from the given **index**.

Syntax with example:

```
1. <int var>=<String var>.indexOf(<char var>,<int var>);  
2. char ch='M';  
3. int ind=str.indexOf(ch, 1);  
4. //2
```



.lastIndexOf(char ch) (int)

This function returns the **index** of the **last occurrence** of a given character.

Syntax with example:

```
1. <int var>=<String var>.lastIndexOf(char ch);  
2. int ind=str.lastIndexOf('E');  
3. //6
```

.substring(int start_index, int last_index) (String)

This function is used to extract a **set of characters** simultaneously from a given index upto the end of the String or till a given index.

Syntax with example:

```
1. <String var>=<String var>.substring(<int var>,<int var>);  
2. String ext=str.substring(3);  
3. //PUTER
```

.toLowerCase() (String)

This function is used to convert a given String to **lowercase** letters (entire string).

Syntax with example:

```
1. <String var>=<String var>.toLowerCase();  
2. String lc=str.toLowerCase();  
3. //computer
```

.toUpperCase() (String)

This function is used to convert a given String to **uppercase** letters (entire string).

Syntax with example:

```
1. <String var>=<String var>.toUpperCase();  
2. String uc=str.toUpperCase(ind);  
3. //COMPUTER
```



.replace(char old, char new) (String)

This function is used to **replace** a **character or a sequence of characters** in a String with a new character or sequence of characters. (**NOTE:** This does not work with int values)

Syntax with example:

```
1. <String var>=<String var>.replace(<char var>,<char var>);
2. String rep=str.replace("PUTER","PUTE");
3. //COMPUTE
```

.concat(String second) (String)

This function is used to **concatenate/join** two Strings together. (**NOTE:** This does not add any spaces in-between)

Syntax with example:

```
1. <String var>=<String var>.concat(s);
2. String s="STUDENT";
3. String con=str.(s);
4. //COMPUTERSTUDENT
```

.equals(String str) (boolean)

This function is used to check for **equality** between two Strings. (**NOTE:** This function returns a **boolean** value. This function cannot be used for characters. //You can simply use == for characters. This can be used in if statements)

Syntax with example:

```
1. <boolean var>=<String var>.equals(<String var>);
2. String s="COMPUTER";
3. boolean chk=str.equals(s);
4. //true
```

.equalsIgnoreCase(String str) (boolean)

This function does the same function of .equals() function. The only difference is that it does not care about the case (It **ignores the case**).

Syntax with example:

```
1. <boolean var>=<String var>.equalsIgnoreCase(<String var>);
2. boolean chk=str.equalsIgnoreCase("cOmPuTeR"); //true
```





.compareTo(String str) (int)

This function is used to **compare** two Strings. It also checks whether a String is **bigger or smaller** than the other and returns a suitable **int value**. It returns **0** if both are **equal**. A **positive** value when the **first is bigger** than the second and a **negative** value when the **second String is bigger** than the first. It returns the **no. of additional characters** when both the Strings' **first sequence of characters are equal** but the other has additional characters.

Syntax with example:

```
1. <int var>=<String var>.compareTo(<String var>);
2. String s="SCIENCE";
3. int cmp=str.compareTo(s);
4. //A, B, C, (C is the 3rd letter in the Alphabet and S is the 19th)
5. //the value of cmp will be -16 because (3-19=-16)
```

.compareToIgnoreCase(String str) (int)

This function does the same function as .compareTo but it **ignores the case**.

Syntax with example:

```
1. <int var>=<String var>.compareToIgnoreCase(<String var>);
2. int cmp=str.compareToIgnoreCase("cOmPuTeR");
3. //0
```

.trim() (String)

This function removes **spaces** at the **start and end** of the String. (**NOTE:** This function does not remove spaces in-between characters)

Syntax with example:

```
1. <String var>=<String var>.trim();
2. Str="    He llo World! ";
3. String trm=str.trim();
4. //He llo World!
```



.startsWith(String str) (boolean)

This function is used to check if the given String is a **prefix** to the other.

Syntax with example:

```
1. <boolean var>=<String var>.startsWith(<String var>);
2. pfx="COM"
3. boolean chk=str.startsWith(pfx);
4. //true
```


.endsWith(String str) (boolean)

This function is used to check if a given String has a specified **suffix**.

Syntax with example:

```
1. <boolean var>=<String var>.ends with(<String var>);
2. boolean chk=str.endsWith("TER");
```

.equals()	.compareTo()
Returns a Boolean value	Returns a an int value
It checks for equality between two Strings	It checks if a String is equal, bigger or smaller than the other.


Difference Between equals() and compareTo() functions 

Library Classes & Wrapper Classes

For better understanding:

Before we get into Library Classes & Wrapper Classes, it's important to know what is a primitive and composite data types.


Primitive Data Type: These are **fundamental built-in data types** of **fixed sizes**. **Ex:** int, long, float

 **Composite/Reference/User-Defined Data Type:** These are data types **created by the user**. The availability of these data types depends upon their **scope and sizes** depend upon their **constituent members**. **Ex:** array, class, object



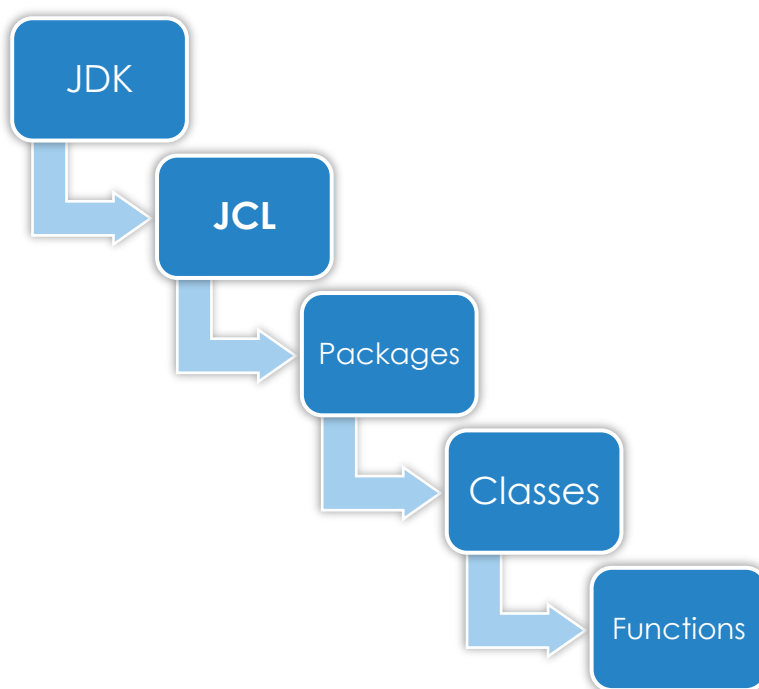


Primitive data type	Composite data type
These are built in data types	Created by user
The sizes of these objects are fixed	The sizes of these data types depend upon their constituent members
These data types are available in all parts of a java program	The availability of these data types depends upon their scope

Difference between primitive and composite data type. 

Library Classes

JDK (Java Development Kit) V1.5 and above contains Java Class Library (JCL) which contains various packages. Each package contains various classes containing different built-in functions.



Ex: `java.lang`, `java.math`

Wrapper Class

Wrapper Classes are a part of `java.lang` (A Library Class Package). Wrapper classes **contain primitive data values in terms of objects**/ Wrapper Class **wraps a primitive data type to an object**. There are 8 wrapper classes in Java. **Ex:** Integer, Byte, Double



(**NOTE:** Wrapper Classes always start with an uppercase letter

Ex: Integer, Boolean, Float)

Need for Wrapper Classes

- To store primitive values in the objects
- To convert a string data into other primitive types and vice-versa

Wrapper Class	Primitive Type
Byte	Byte
Short	short
Integer	int
Long	long
Float	float
Double	double
Character	char
Boolean	boolean

Wrapper Classes and their primitive types 

Functions/Methods of Wrapper Classes

Conversion from String to Primitive types

For converting String to any primitive data type, Wrapper Class functions can be used. For any primitive data Wrapper Class, the **parse**<prm data type>(<String arg>) (or) **valueOf**(<String arg>) functions can be used.

Eg: `int i=Integer.parseInt(s); int j=Integer.valueOf(s);`

For better understanding:

```

1. <prm data type var>=<prm data type Wrapper Class>.parse<prm data
   type name>(<String arg>);
2. <prm data type var>=<prm data type wrapper class>.valueOf(<String
   arg>);
3.
4. //Examples:
5. int a=Integer.parseInt("238");
6. double b=Double.parseDouble("23.45");
7. int c=Integer.valueOf("37");
8. float d=Float.valueOf("42.87");

```




Examples of each <> (In the above syntax):

prm data type: int a | double b

prm data type name: Int | Long | Double

prm data wrapper class: Integer | Double

String arg: "38.743" | "1874293856"

Conversion from primitive type data to String

For converting a primitive type data to a String, the **toString()** Wrapper Class function can be used.

Ex: Integer.toString() | Double.toString()

```
1. <String var>=<Wrapper Class>.toString(<prm data arg>);  
2. String cnv=Integer.toString(38);  
3. String dbl=Double.toString(94.53);
```

Boxing, Unboxing & Autoboxing

Boxing

Conversion of primitive type data **to an object**.

Syntax with example:

```
1. <wrapper class> <object name>=new <wrapper class>(<prm type arg>);  
2. Int a=239;  
3. Integer x=new Integer(a);
```

Unboxing

Conversion of an object **to primitive type data**.

Syntax with example:

```
1. <int var>=<wrapper class obj>  
2. int b=x;
```

Autoboxing

Boxing is the mechanism and autoboxing is the feature of the compiler which generates the boxing code.

Syntax with example:

```
1. <wrapper class> <object name>=new <wrapper class>(<prm type arg>);  
2. Int a=239;  
3. Integer x=new Integer(a);
```



Character

Character is defined as a letter, a digit or any special symbol/UNICODE enclosed within single quotes. **Ex:** '@', 's', '5'

Assigning a character

A Character is declared under char data type.

Syntax with example:

```
1. char <var name>='<char literal>';  
2. char ch='a';
```

Input a character

A Character is declared under char data type.

Syntax with example:

```
1. <char var>=<Scanner obj>.next().charAt(0);  
2. ch=sc.next().charAt(0);
```

Character Functions

Character.isLetter() (boolean)

This function is used to check if a given argument is a letter or not.

Syntax with example:

```
1. <boolean var>=Character.isLetter(<char arg>);  
2. boolean chk=Character.is('A'); //true
```

Character.isDigit() (boolean)

This function is used to check if a given argument is a digit or not.

Syntax with example:

```
1. <boolean var>=Character.isDigit(<char arg>);  
2. boolean chk=Character.is('7'); //true
```



Character.isLetterOrDigit() (boolean)

This function is used to check if a given argument is either a letter or a digit or none of these.

Syntax with example:

```
1. <boolean var>=Character.is(<char arg>);  
2. boolean chk=Character.is('A'); //true
```

Character.isWhitespace() (boolean)

This function is used to check if a given argument is a blank/gap/space or not.

Syntax with example:

```
1. <boolean var>=Character.is(<char arg>);  
2. boolean chk=Character.is('A'); //false
```

Character.isUpperCase() (boolean)

This function is used to check if a given argument is an uppercase letter or not.

Syntax with example:

```
1. <boolean var>=Character.is(<char arg>);  
2. boolean chk=Character.is('A'); //true
```

Character.isLowerCase() (boolean)

This function is used to check if a given argument is a or not.

Syntax with example:

```
1. <boolean var>=Character.is(<char arg>);  
2. boolean chk=Character.is('A'); //false
```

Character.toUpperCase() (char)

This function is used to convert/returns a given argument/character/letter to/in uppercase character/letter.

Syntax with example:

```
1. <char var>=Character.toUpperCase(<char arg>);  
2. char uc=Character.toUpperCase('a'); //A
```



Character.toLowerCase() (char)

This function is used to convert/returns a given argument/character/letter to/in lowercase character/letter.

Syntax with example:

```
1. <char var>=Character.toLowerCase(<char arg>);  
2. char lc=Character.toLowerCase('A'); //a
```

Revision/Review Questions

From the last 15yrs' Board Papers

(NOTE: Some questions have been modified and rephrased to suit the type of question in a way that it retains the essence of the question)

MCQ

-
1. What does compareTo() return? **[ICSE 2014]**
 - a) boolean
 - b) double
 - c) int
 - d) A detailed list of analysis of the values of the Strings
 2. What does equals() return? **[ICSE 2014]**
 - a) int
 - b) float
 - c) String
 - d) Boolean
 3. Name a function that removes the blank spaces at the start and at the end of the String. **[ICSE 2015]**
 - a) removeSpace()
 - b) trim()
 - c) delSpace()
 - d) Dear Examiner, there is no such function which does that. -Candidate



4. What is the value of ind if ind=str.indexOf('a'); while str="Tata, bye bye!";
- a) 2
 - b) 2,4
 - c) 1
 - d) ERROR. Please check the question!

Answer the following

1. Name any two wrapper classes **[ICSE 2013]**
2. What is the return type of the following library functions
e) isWhiteSpace() b) Math.random **[ICSE 2013]**
3. What are library classes? Give an example. **[ICSE 2011]**
4. Why is class known as composite data type? **[ICSE 2009]**
5. A method that converts a String to an Integer primitive data type **[ICSE 2009]**

Programs (Includes questions out of board papers)

1. Write a program to replace a particular index with a given character in a String.
Ex:
input: FrozenNotes | 5 | d (3 separate inputs)
output: FrozedNotes
2. Write a program to display a word in reverse.
3. Write a program to delete alt letters in a word.
Ex:
Input: FrozenNotes
Output: F o e n o e s

Answer Key

MCQ

1. c
2. d
3. b
4. c





Answer the following

1. > Character > Integer
2. a) boolean b) double
3. JCL is present in JDK. It includes various packages with build-in functions to provide effective support to users for development of productive logic.
Ex: java.util.*; java.io.*;
4. Class is known as composite data type because it encapsulates one or more primitive data types together as a single data type.
5. Integer.parseInt();

Programs

Uhm... It would be nice if you try it yourself.

Syllabus

String handling

String class, methods of String class, implementation of String class methods, String array. Outputs based on all the above methods, Programs based on the above methods, extracting, and modifying characters of a string, alphabetical order of the strings in an array [Bubble sort technique], searching for a string in a string array using linear search technique. SIMPLE Programs based on extraction of characters.

Library classes

Introduction to wrapper classes, methods of wrapper class and their usage with respect to numeric and character data types. Autoboxing and Unboxing in wrapper classes. Class as a composite type, distinction between primitive data type and composite data type or class types. Class may be considered as a new data type created by the user, which has its own functionality. The distinction between primitive and composite types should be discussed through examples. Show how classes allow user defined types in programs. All primitive types have corresponding class wrappers. Introduce Autoboxing and Unboxing with their definition and simple examples.

GENERAL NOTICE

The notes are as per the latest ICSE curriculum (ICSE 2023). A content of India.

COPYRIGHT NOTICE

©2022-2025 FROZENNOTES

Subject to **(CC BY-NC-SA)** Creative Commons [Attribution-NonCommercial-ShareAlike 4.0 International](https://creativecommons.org/licenses/by-nc-sa/4.0/)

CC- Creative Commons

BY – Attribution Not Required for this document

NC – Non-Commercial

SA – Adaptations must be shared under the same licence

No individual/team/group/organisation/company/cooperate is allowed to earn profit from/with/through this document.

EDUCATION IS NOT FOR PROFIT

Additional permission requests can be made (Approval guarantee - subject to request)

FROZENNOTES reserves the right to change the terms anytime without (or) with prior notice

Any doubts/questions/suggestions/permission request regarding the document can be sent to

https://frozennotes.github.io/ICSE_Resources (or)

sanjayrohith@outlook.com

Enhanced with Microsoft Editor AI

Not affiliated with Microsoft or ICSE



©2022-2025 FROZENNOTES

