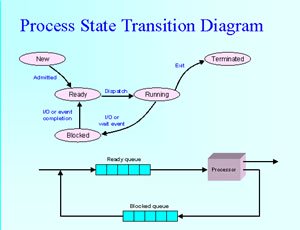
# Java General Questions

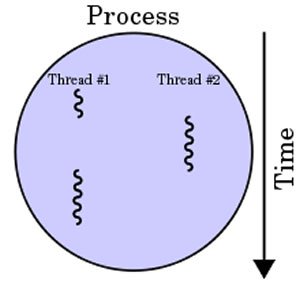
* What is difference between thread and process?

##### **Key difference: Thread and Process are two closely related terms in multi-threading. The main difference between the two terms is that the threads are a part of a process, i.e. a process may contain one or more threads, but a thread cannot contain a process.**

In programming, there are two basic units of execution: processes and threads. They both execute a series of instructions. Both are initiated by a program or the operating system. This article helps to differentiate between the two units.

A process is an instance of a program that is being executed. It contains the program code and its current activity. Depending on the operating system, a process may be made up of multiple threads of execution that execute instructions concurrently. A program is a collection of instructions; a process is the actual execution of those instructions.

A process has a self-contained execution environment. It has a complete set of private basic run-time resources; in particular, each process has its own memory space. Processes are often considered similar to other programs or applications. However, the running of a single application may in fact be a set of cooperating processes. To facilitate communication between the processes, most operating systems use Inter Process Communication (IPC) resources, such as pipes and sockets. The IPC resources can also be used for communication between processes on different systems. Most applications in a virtual machine run as a single process. However, it can create additional processes using a process builder object.



In computers, a thread can execute even the smallest sequence of programmed instructions that can be managed independently by an operating system. The applications of threads and processes differ from one operating system to another. However, the threads are made of and exist within a process; every process has at least one. Multiple threads can also exist in a process and share resources, which helps in efficient communication between threads.

On a single processor, multitasking takes place as the processor switches between different threads; it is known as multithreading. The switching happens so frequently that the threads or tasks are perceived to be running at the same time. Threads can truly be concurrent on a multiprocessor or multi-core system, with every processor or core executing the separate threads simultaneously.

In summary, threads may be considered lightweight processes, as they contain simple sets of instructions and can run within a larger process. Computers can run multiple threads and processes at the same time.

Comparison between Process and Thread:

|  |  |  |
| --- | --- | --- |
|  | **Process** | **Thread** |
| Definition | An executing instance of a program is called a process. | A thread is a subset of the process. |
| Process | It has its own copy of the data segment of the parent process. | It has direct access to the data segment of its process. |
| Communication | Processes must use inter-process communication to communicate with sibling processes. | Threads can directly communicate with other threads of its process. |
| Overheads | Processes have considerable overhead. | Threads have almost no overhead. |
| Creation | New processes require duplication of the parent process. | New threads are easily created. |
| Control | Processes can only exercise control over child processes. | Threads can exercise considerable control over threads of the same process. |
| Changes | Any change in the parent process does not affect child processes. | Any change in the main thread may affect the behavior of the other threads of the process. |
| Memory | Run in separate memory spaces. | Run in shared memory spaces. |
| File descriptors | Most file descriptors are not shared. | It shares file descriptors. |
| File system | There is no sharing of file system context. | It shares file system context. |
| Signal | It does not share signal handling. | It shares signal handling. |
| Controlled by | Process is controlled by the operating system. | Threads are controlled by programmer in a program. |
| Dependence | Processes are independent. | Threads are dependent |

# **Java String Interview Questions**

* What is String in Java? String is a data type?

String is a Class in java and defined in java.lang package. It’s not a primitive data type like int and long. String class represents character Strings. String is used in almost all the Java applications and there are some interesting facts we should know about String. String in immutable and final in Java and JVM uses String Pool to store all the String objects.  
Some other interesting things about String is the way we can instantiate a String object using double quotes and overloading of “+” operator for concatenation.

* What are different ways to create String Object?

We can create String object using new operator like any normal java class or we can use double quotes to create a String object. There are several constructors available in String class to get String from char array, byte array, StringBuffer and StringBuilder.

String str = new String("abc");

String str1 = "abc";

When we create a String using double quotes, JVM looks in the String pool to find if any other String is stored with same value. If found, it just returns the reference to that String object else it creates a new String object with given value and stores it in the String pool.  
When we use new operator, JVM creates the String object but don’t store it into the String Pool. We can use intern() method to store the String object into String pool or return the reference if there is already a String with equal value present in the pool.

* Write a method to check if input String is Palindrome?

A String is said to be Palindrome if it’s value is same when reversed. For example “aba” is a Palindrome String.  
String class doesn’t provide any method to reverse the String but StringBuffer and StringBuilder class has reverse method that we can use to check if String is palindrome or not.

private static boolean isPalindrome(String str) {

if (str == null)

return false;

StringBuilder strBuilder = new StringBuilder(str);

strBuilder.reverse();

return strBuilder.toString().equals(str);

}

Sometimes interviewer asks not to use any other class to check this, in that case we can compare characters in the String from both ends to find out if it’s palindrome or not.

private static boolean isPalindromeString(String str) {

if (str == null)

return false;

int length = str.length();

System.out.println(length / 2);

for (int i = 0; i < length / 2; i++) {

if (str.charAt(i) != str.charAt(length - i - 1))

return false;

}

return true;

}

* Write a method that will remove given character from the String?

We can use replaceAll method to replace all the occurance of a String with another String. The important point to note is that it accepts String as argument, so we will use Character class to create String and use it to replace all the characters with empty String.

private static String removeChar(String str, char c) {

if (str == null)

return null;

return str.replaceAll(Character.toString(c), "");

}

* How to compare two Strings in java program?

Java String implements Comparable interface and it has two variants of compareTo() methods.

**compareTo(String anotherString)** method compares the String object with the String argument passed lexicographically. If String object precedes the argument passed, it returns negative integer and if String object follows the argument String passed, it returns positive integer. It returns zero when both the String have same value, in this case equals(String str) method will also return true.

**compareToIgnoreCase(String str)**: This method is similar to the first one, except that it ignores the case. It uses String CASE\_INSENSITIVE\_ORDER Comparator for case insensitive comparison. If the value is zero then equalsIgnoreCase(String str) will also return true.

Check this post for String compareTo example.

* How to convert String to char and vice versa?

This is a tricky question because String is a sequence of characters, so we can't convert it to a single character. We can use use charAt method to get the character at given index or we can use toCharArray()method to convert String to character array.

String class has three methods related to char. Let’s look at them before we look at a java program to convert string to char array.

char[] toCharArray(): This method converts string to character array. The char array size is same as the length of the string.

char charAt(int index): This method returns character at specific index of string. This method throws StringIndexOutOfBoundsException if the index argument value is negative or greater than the length of the string.

getChars(int srcBegin, int srcEnd, char dst[], int dstBegin): This is a very useful method when you want to convert part of string to character array. First two parameters define the start and end index of the string; the last character to be copied is at index srcEnd-1. The characters are copied into the char array starting at index dstBegin and ending at dstBegin + (srcEnd-srcBegin) – 1.

public class StringToCharJava {

public static void main(String[] args) {

String str = "journaldev";

//string to char array

char[] chars = str.toCharArray();

System.out.println(chars.length);

//char at specific index

char c = str.charAt(2);

System.out.println(c);

//Copy string characters to char array

char[] chars1 = new char[7];

str.getChars(0, 7, chars1, 0);

System.out.println(chars1);

}

}

* Does String is thread-safe in Java?

Strings are immutable, so we can't change it's value in program. Hence it's thread-safe and can be safely used in multi-threaded environment.

* Why String is popular HashMap key in Java?

Since String is immutable, its hashcode is cached at the time of creation and it doesn’t need to be calculated again. This makes it a great candidate for key in a Map and it’s processing is fast than other HashMap key objects. This is why String is mostly used Object as HashMap keys.

* What does String intern() method do?

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.

This method always return a String that has the same contents as this string, but is guaranteed to be from a pool of unique strings.

* What is String Pool?

As the name suggests, String Pool is a pool of Strings stored in Java heap memory. We know that String is special class in java and we can create String object using new operator as well as providing values in double quotes.

* Why String is immutable or final in Java

There are several benefits of String because it's immutable and final.

1. String Pool is possible because String is immutable in java.
2. It increases security because any hacker can't change its value and it's used for storing sensitive information such as database username, password etc.
3. Since String is immutable, it's safe to use in multi-threading and we don't need any synchronization.
4. Strings are used in java classloader and immutability provides security that correct class is getting loaded by Classloader.
5. Since String is immutable, its hashcode is cached at the time of creation and it doesn’t need to be calculated again. This makes it a great candidate for key in a Map and it’s processing is fast than other HashMap key objects. This is why String is mostly used Object as HashMap keys.

* Difference between String, StringBuffer and StringBuilder?

String is immutable and final in java, so whenever we do String manipulation, it creates a new String. String manipulations are resource consuming, so java provides two utility classes for String manipulations - StringBuffer and StringBuilder.

StringBuffer and StringBuilder are mutable classes. StringBuffer operations are thread-safe and synchronized where StringBuilder operations are not thread-safe. So when multiple threads are working on same String, we should use StringBuffer but in single threaded environment we should use StringBuilder.

StringBuilder performance is fast than StringBuffer because of no overhead of synchronization.

* + Java Program to find all Permutations of a String

To get all the permutations, we will first take out the first char from String and permute the remaining chars.

If String = “ABC”

First char = A and remaining chars permutations are BC and CB.

Now we can insert first char in the available positions in the permutations.

BC -> ABC, BAC, BCA

CB -> ACB, CAB, CBA

So we can write a recursive function call to return the permutations and then another function call to insert the first characters to get the complete list of permutations.

import java.util.HashSet;

import java.util.Set;

/\*\*

\* Java Program to find all permutations of a String

\* @author pankaj

\*

\*/

public class StringHelper {

public static Set<String> permutationFinder(String str) {

Set<String> perm = new HashSet<String>();

//Handling error scenarios

if (str == null) {

return null;

} else if (str.length() == 0) {

perm.add("");

return perm;

}

char initial = str.charAt(0); // first character

String rem = str.substring(1); // Full string without first character

Set<String> words = permutationFinder(rem);

for (String strNew : words) {

for (int i = 0;i<=strNew.length();i++){

perm.add(charInsert(strNew, initial, i));

}

}

return perm;

}

public static String charInsert(String str, char c, int j) {

String begin = str.substring(0, j);

String end = str.substring(j);

return begin + c + end;

}

public static void main(String[] args) {

String s = "AAC";

String s1 = "ABC";

String s2 = "ABCD";

System.out.println("\nPermutations for " + s + " are: \n" + permutationFinder(s));

System.out.println("\nPermutations for " + s1 + " are: \n" + permutationFinder(s1));

System.out.println("\nPermutations for " + s2 + " are: \n" + permutationFinder(s2));

}

}

* + Can we use String in switch case?

This is a tricky question used to check your knowledge of current Java developments. Java 7 extended the capability of switch case to use Strings also, earlier java versions doesn't support this.

If you are implementing conditional flow for Strings, you can use if-else conditions and you can use switch case if you are using Java 7 or higher versions.

* + What is the output of below program?

package com.journaldev.strings;

public class Test {

public void foo(String s) {

System.out.println("String");

}

public void foo(StringBuffer sb){

System.out.println("StringBuffer");

}

public static void main(String[] args) {

new Test().foo(null);

}

}

The above program will not compile with error as "The method foo(String) is ambiguous for the type Test".

* + What is the output of below code snippet?

String s1 = new String("abc");

String s2 = new String("abc");

System.out.println(s1 == s2);

It will print false because we are using new operator to create String, so it will be created in the heap memory and both s1, s2 will have different reference. If we create them using double quotes, then they will be part of string pool and it will print true.

* + What will be output of below code snippet?

String s1 = "abc";

StringBuffer s2 = new StringBuffer(s1);

System.out.println(s1.equals(s2));

It will print false because s2 is not of type String. If you will look at the equals method implementation in the String class, you will find a check using instanceof operator to check if the type of passed object is String? If not, then return false.

* + How many String objects got created in below code snippet?

String s1 = new String("Hello");

String s2 = new String("Hello");

Answer is 3.

First - line 1, "Hello" object in the string pool.

Second - line 1, new String with value "Hello" in the heap memory.

Third - line 2, new String with value "Hello" in the heap memory. Here "Hello" string from string pool is reused.