Strings

# What is String in Java? Is String is data type?

String in Java is not a primitive data type like int, long or double. String is a class or in more simple term a user defined type. This is confusing for someone who comes from C background. String is defined in java.lang package and wrappers its content in a character array. String provides equals() method to compare two String and provides various other method to operate on String like toUpperCase() to convert String into upper case, replace() to replace String contents, substring() to get substring, split() to split long String into multiple String.

# Why String is immutable or final in Java:

Though there could be many possible answer for this question, and only designer of String class can answer this, I think below two does make sense

- Imagine StringPool facility without making string immutable , its not possible at all because in case of string pool one string object/literal e.g. "Test" has referenced by many reference variables , so if any one of them change the value of others will be automatically gets affected i.e. lets say

String A = "Test"

String B = "Test"

Now String B called "Test".toUpperCase() which change the same object into "TEST" , so A will also be "TEST" which is not desirable.

-String has been widely used as parameter for many Java classes e.g. for opening network connection, you can pass hostname and port number as string , you can pass database URL as string for opening database connection, you can open any file in Java by passing name of file as argument to File I/O classes.

In case, if String is not immutable, this would lead serious security threat, I mean some one can access to any file for which he has authorization, and then can change the file name either deliberately or accidentally and gain access of those file. Because of immutability, you don't need to worry about those kind of threats. This reason also gel with, Why String is final in Java, by making java.lang.String final, Java designer ensured that no one overrides any behaviour of String class.

-Since String is immutable it can be safely shared between many threads, which is very important for multithreaded programming and to avoid any synchronization issues in Java, Immutability also makes String instance thread-safe in Java, means you don't need to synchronize String operation externally. Another important point to note about String is memory leak caused by SubString, which is not a thread related issues but something to be aware of.

- Another reason of Why String is immutable in Java is to allow String to cache its hashcode , being immutable String in Java caches its hashcode, and do not calculate every time we call hashcode method of String, which makes it very fast as hashmap key to be used in hashmap in Java. In short because String is immutable, no one can change its contents once created which guarantees hashCode of String to be same on multiple invocation.

- Another good reason of Why String is immutable in Java suggested by Dan Bergh Johnsson on comments is: The absolutely most important reason that String is immutable is that it is used by the class loading mechanism, and thus have profound and fundamental security aspects. Had String been mutable, a request to load "java.io.Writer" could have been changed to load "mil.vogoon.DiskErasingWriter"

Read more: http://javarevisited.blogspot.com/2010/10/why-string-is-immutable-in-java.html#ixzz2xcgCBH8F

# What is immutable Class in Java:

As said earlier, Immutable classes are those class, whose object can not be modified once created, it means any modification on immutable object will result in another immutable object. best example to understand immutable and mutable objects are, String and StringBuffer. Since String is immutable class, any change on existing string object will result in another string e.g. replacing a character into String, creating substring from String, all result in a new objects. While in case of mutable object like StringBuffer, any modification is done on object itself and no new objects are created. Sometimes this immutability of String can also cause security hole, and that the reason why password should be stored on char array instead of String.

Read more: http://javarevisited.blogspot.com/2013/03/how-to-create-immutable-class-object-java-example-tutorial.html#ixzz2xcm9tARO

# How to create an immutable class in java:

Here is complete code example of writing immutable class in Java. We have followed simplest approach and all rules for making a class immutable, including it making class final to avoid putting immutability at risk due to Inheritance and Polymorphism.

public final class Contacts {

    private final String name;

    private final String mobile;

    public Contacts(String name, String mobile) {

        this.name = name;

        this.mobile = mobile;

    }

    public String getName(){

        return name;

    }

    public String getMobile(){

        return mobile;

    }

}

This Java class is immutable, because its state can not be changed once created. You can see that all of it’s fields are final. This is one of the most simple way of creating immutable class in Java, where all fields of class also remains immutable like String in above case. Some time you may need to write immutable class which includes mutable classes like java.util.Date, despite storing Date into final field it can be modified internally, if internal date is returned to the client. In order to preserve immutability in such cases, its advised to return copy of original object, which is also one of the Java best practice. here is another example of making a class immutable in Java, which includes mutable member variable.

public final class ImmutableReminder{

    private final Date remindingDate;

    public ImmutableReminder (Date remindingDate) {

        if(remindingDate.getTime() < System.currentTimeMillis()){

            throw new IllegalArgumentException("Can not set reminder” +

                        “ for past time: " + remindingDate);

        }

        this.remindingDate = new Date(remindingDate.getTime());

    }

    public Date getRemindingDate() {

        return (Date) remindingDate.clone();

    }

}

In above example of creating immutable class, Date is a mutable object. If getRemindingDate() returns actual Date object than despite remindingDate being final variable, internals of Date can be modified by client code. By returning clone() or copy of remindingDate, we avoid that danger and preserves immutability of class.

Read more: http://javarevisited.blogspot.com/2013/03/how-to-create-immutable-class-object-java-example-tutorial.html#ixzz2xeX9cFWO

# Immutable objects are thread-safe but why/how:

because once an immutable object is created, it cant be modified

# Benefits of immutable objects:

1) Immutable objects are by default thread safe, can be shared without synchronization in concurrent environment.

2) Immutable object simplifies development, because its easier to share between multiple threads without external synchronization.

3) Immutable object boost performance of Java application by reducing synchronization in code.

4) Another important benefit of Immutable objects is reusability, you can cache Immutable object and reuse them, much like String literals and Integers.  You can use static factory methods to provide methods like valueOf(), which can return an existing Immutable object from cache, instead of creating a new one.

Read more: http://javarevisited.blogspot.com/2013/03/how-to-create-immutable-class-object-java-example-tutorial.html#ixzz2xecze900

# Dis-advantage of immutable objects:

immutable object has disadvantage of creating garbage

# How is String class immutable:

# Is Stringobject thread-safe:

Since String object is immutable, it is thread-safe

# What is String pool in Java

Another tough Java question asked in String interview. String pool is a special storage area in Java heap, mostly located on PerGen space, to store String literals like "abc". When Java program creates a new String using String literal, JVM checks for that String in pool and if String literal is already present in pool than same object is returned instead of creating a whole new object. String pool check is only performed when you create String as literal, if you create String using new() operator, a new String object will be created even if String with same content is available in pool.

Read more: <http://javarevisited.blogspot.com/2012/10/10-java-string-interview-question-answers-top.html#ixzz2xef8iRPV>

# What is Difference between String and StringBuffer in Java

This is probably the most common question on String I have seen in Java interviews. Though String and Stringbuffer are two different class they are used in context of concatenating two Strings, Since String is immutable in Java every operation which changes String produces new String, which can be avoided by using Stringbuffer. See String vs StringBuffer for more details.

Read more: <http://javarevisited.blogspot.com/2012/10/10-java-string-interview-question-answers-top.html#ixzz2xefgOahM>

# How do you compare two String in Java?

This is another common String interview question which appears on fresher level interviews. There are multiple ways to compare two String like equals() method, equalsIgnoreCase() etc, You can also see 4 ways to compare String in Java for more examples. Main thing which interviewer checks is that whether candidate mentioned equality operator or not "==", comparing String with equality operator is common mistake which works in some case and doesn't work in other.

# Can we compare String using == operator? What is risk?

As discussed in previous String question, You can compare String using equality operator but that is not suggested or advised because equality operator is used to compare primitives and equals() method should be used to compare objects. As we have seen in pitfall of autoboxing in Java that how equality operator can cause subtle issue while comparing primitive to Object, any way String is free from that issue because it doesn't have corresponding primitive type and not participate in autoboxing. Almost all the time comparing String means comparing contents of String i.e. characters and equals() method is used to perform character based comparison. equals() return true if two String points to same object or two String has same contents while == operator returns true if two String object points to same object but return false if two different String object contains same contents. That explains why sometime it works and sometime it doesn't. In short always use equals method in Java to check equality of two String object.

Read more: http://javarevisited.blogspot.com/2012/10/10-java-string-interview-question-answers-top.html#ixzz2xegkgwwb

# What does intern() method do in Java

As discussed in previous String interview question, String object created by new() operator is by default not added in String pool as opposed to String literal. intern() method allows to put an String object into pool.

Read more: http://javarevisited.blogspot.com/2012/10/10-java-string-interview-question-answers-top.html#ixzz2xeh9wzuA

# 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;

}

# 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.

# 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.

# String being used in thread synchronization

If 2 seperate threads are executing 2 seperate synchronized blocks and lock on the synchronized blocks are held using 2 seperate string variables, but both the string variables will have same value: When one thread will be executing one synchronized block and holding the lock, then the other thread cant execute the other synchronized block.

Eg:

T1

===-

package test.synchronization;

public class T1 implements Runnable{

String s="abc";

@Override

public void run() {

// TODO Auto-generated method stub

synchronized (s) {

for(int i=0;i<10;i++){

System.out.println("T1");

try{

Thread.sleep(1000);

}catch(InterruptedException ie){

ie.printStackTrace();

}

}

}

}

}

T2

===-

package test.synchronization;

public class T2 implements Runnable{

String s="abc";

@Override

public void run() {

// TODO Auto-generated method stub

synchronized (s) {

for(int i=0;i<10;i++){

System.out.println("T2");

try{

Thread.sleep(1000);

}catch(InterruptedException ie){

ie.printStackTrace();

}

}

}

}

}

Main

====-

package test.synchronization;

public class Main {

public static void main(String[] args) {

Thread t1= new Thread(new T1());

Thread t2= new Thread(new T2());

t1.start();

t2.start();

}

}

output:

T1

T1

T1

T1

T1

T1

T1

T1

T1

T1

T2

T2

T2

T2

T2

T2

T2

T2

T2

T2

If s is changed in any class to some other value, then both will execute in parallel.

# String Literal Pool

There are slight differences between the various methods of creating a String object. String allocation, like all object allocation, proves costly in both time and memory. The JVM performs some trickery while instantiating string literals/objects to increase performance and decrease memory overhead. To cut down the number of String objects created, **JVM maintains a special memory called “String literal pool” or “String constant pool”**.

Each time your code creates a string literal, the JVM checks the string literal pool first. If the string already exists in the pool, a reference to the pooled instance is returned. If the string does not exist in the pool, a new String object is created and placed in the pool. JVM keeps at most one object of any String in this pool. String literals always refer to an object in the string pool.

**For example,**

## **Direct Method of creating String object**

|  |  |
| --- | --- |
| 1 | String s1 = "iByteCode"; |

**How this works?**

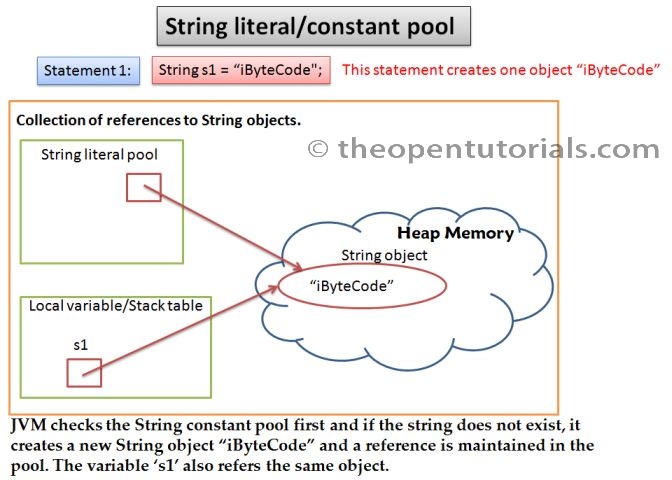
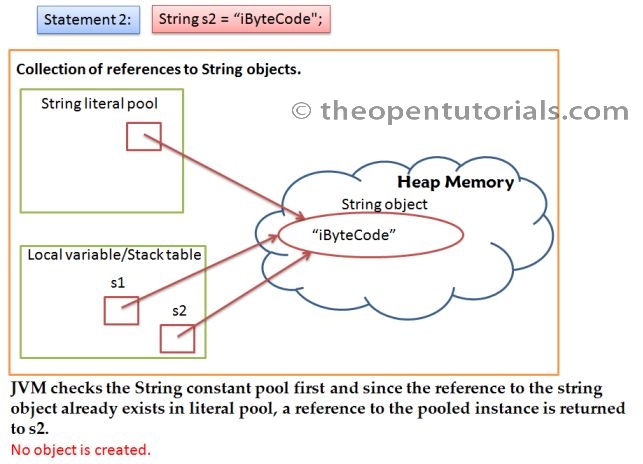
* JVM checks the String constant pool first and if the string does not exist, it creates a new String object “iByteCode” and a reference is maintained in the pool. The variable ‘s1′ also refers the same object.
* This statement creates one String object “iByteCode”.
* Now, let’s see what happens if we have a statement like this:

|  |  |
| --- | --- |
| 1 | String s2 = "iByteCode"; |

* JVM checks the String constant pool first and since the string already exists, a reference to the pooled instance is returned to s2.
* This statement does not create any String object in the memory and ‘s2′ refers the same object as ‘s1′.
* To check this, you can compare two String references using == operator to check whether two references are referring to the same String object in the memory.

|  |  |
| --- | --- |
| 1  2  3  4 | String s1 = "iByteCode";  String s2 = "iByteCode";  if(s1 == s2)      System.out.println("s1 and s2 referring to the same object."); |

s1 and s2 referring to the same object.

[](http://theopentutorials.com/totwp331/wp-content/uploads/string-literal-pool_871/java-string-constant-pool-1.jpg?c3a9c1)  
[](http://theopentutorials.com/totwp331/wp-content/uploads/string-literal-pool_871/java-string-constant-pool-2.jpg?c3a9c1)

Java can make this optimization since strings are immutable and can be shared without fear of data corruption. For example, if several reference variables refer to the same String object then it would be bad if any of them changes the String’s value. This is the reason for making String objects as immutable.

## **Creating String using constructor**

|  |  |
| --- | --- |
| 1 | String s = new String("iByteCode"); |

In this case, because we used ‘new’ keyword a **String object is created in the heap memory** even if an equal string object already exists in the pool **and ‘s’ will refer to the newly created one**.

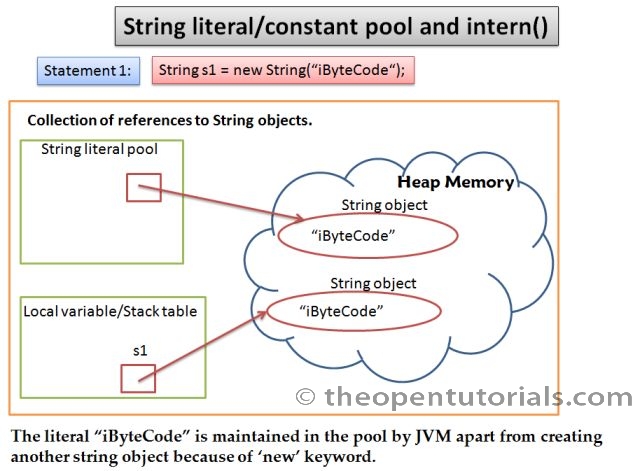
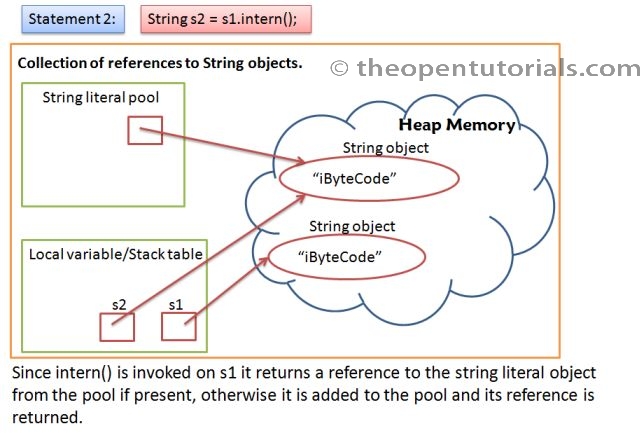
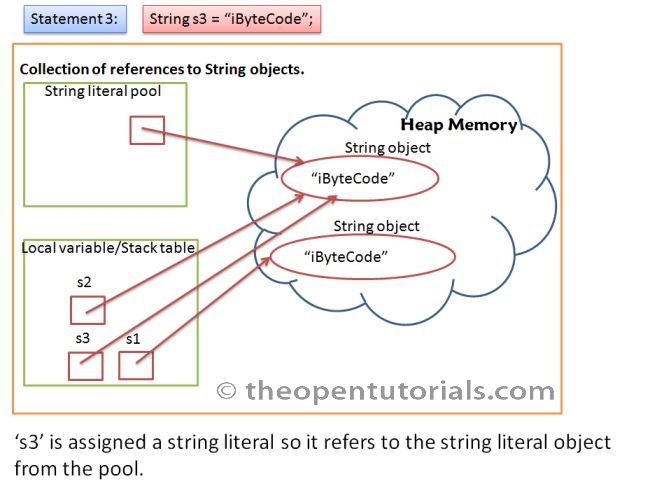
|  |  |
| --- | --- |
| 1  2  3 | String str1 = "iByteCode";  String str2 = new String("iByteCode");  System.out.println(str1 == str2); |

false

**String objects created with the new operator do not refer to objects in the string pool but can be made to using String’s intern() method.** The java.lang.String.intern() returns an interned String, that is, one that has an entry in the global String literal pool. If the String is not already in the global String literal pool, then it will be added. For example,

|  |  |
| --- | --- |
| 1  2  3  4 | String s1 = new String("iByteCode");  String s2 = s1.intern();  String s3 = "iByteCode";  System.out.println(s2 == s3); |

true

[](http://theopentutorials.com/totwp331/wp-content/uploads/string-literal-pool_871/java-string-constant-pool-intern-1.jpg?c3a9c1)  
[](http://theopentutorials.com/totwp331/wp-content/uploads/string-literal-pool_871/java-string-constant-pool-intern-2.jpg?c3a9c1)  
[](http://theopentutorials.com/totwp331/wp-content/uploads/string-literal-pool_871/java-string-constant-pool-intern-3.jpg?c3a9c1)

In the above example, if the change the statement 2 as,

|  |  |
| --- | --- |
| 1 | String s2 = s1; |

Reference variable ‘s2′ will refer to the string object in the heap instead of string literal pool and s1 == s2 will print true.

An object is eligible for garbage collection when it is no longer referenced from an active part of the application. In the case of String literals, they always have a reference to them from the String Literal Pool and are, therefore, not eligible for garbage collection.

All the string literals are created and their references are placed in the pool while JVM loads the class. So, even before a statement like this **String s1 = new String(“iByteCode”);** is executed, the string literal pool contains a reference to “iByteCode”.

# StringBuffer VS StringBuilder

## **StringBuffer**

StringBufferis mutable means one can change the value of the object. The object created through StringBuffer is stored in the heap. StringBuffer has the same methods as the StringBuilder , but **each method in StringBuffer is synchronized** that is **StringBuffer is thread safe** .   
Due to this it does not allow two threads to simultaneously access the same method . Each method can be accessed by one thread at a time.  
But being thread safe has disadvantages too as the performance of the StringBuffer hits due to thread safe property. **Thus StringBuilder is faster than the StringBuffer when calling the same methods of each class.**  
StringBuffer value can be changed, it means it can be assigned to the new value . Nowadays its a most common interview question ,the differences between the above classes .String Buffer can be converted to the string by using toString() method.

## **StringBuilder**

StringBuilder  is same as the StringBuffer , that is it stores the object in heap and it can also be modified . The main difference between the StringBuffer and StringBuilder is that StringBuilder is also not thread safe.   
StringBuilder is fast as it is not thread safe .

# Is String thread safe ????

# Reference:

1)immutable:

-http://www.javaranch.com/journal/2003/04/immutable.htm

-http://javarevisited.blogspot.com/2013/03/how-to-create-immutable-class-object-java-example-tutorial.html

-http://javarevisited.blogspot.sg/2010/10/why-string-is-immutable-in-java.html

2)String:

-http://javarevisited.blogspot.com/2012/10/10-java-string-interview-question-answers-top.html

-http://www.journaldev.com/1321/java-string-interview-questions-and-answers