Chapter 5

String Handling

Introduction

Generally string is a sequence of characters. But in java, string is an object of **String** class. It is not a character array. In Java, we got character array also, but string are given a different treatment because of their extensive use in Programming. JavaSoft people have created a class separately with the name 'String' in **java.lang** package with all necessary method to work with Strings.

In java, string is basically an *immutable* object. We will discuss about immutable string later. Let's first understand what string is and how we can create the string object.

How to create String object?

There are two ways to create strings in Java:

1. By string literal
2. By new keyword

String literal

We can create a string just by assigning a group of characters to a string type variable:

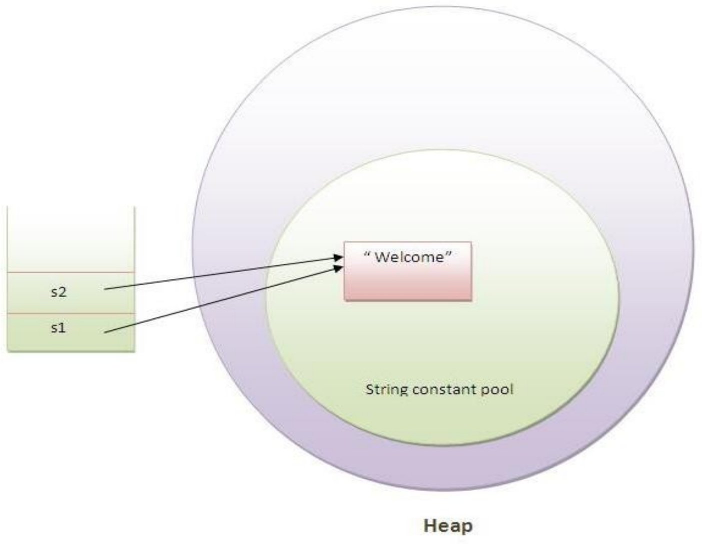
1. String literal is created by double quote. For Example:

   String s="Hello";

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 returns. If the string does not exist in the pool, a new String object instantiates, then is placed in the pool. For example:

Note: In Java when an object or an array is created, memory is allocated to them from heap. The JVM through the use of **new** operator allocates memory from the heap for an object. String constant pool is part of such heap memory.

1. String s1="Welcome";
2. String s2="Welcome";//no new object will be created



In the above example only one object will be created. First time JVM will find no string object with the name "Welcome" in string constant pool, so it will create a new object. Second time it will find the string with the name "Welcome" in string constant pool, so it will not create new object whether will return the reference to the same instance.

Note: String constant pool is a separate memory block inside the heap where the string object are stored by JVM. If a string object is created directly, using assignment operator as: String s1 = "Sachin", then it is stored in string constant pool. **String objects are stored in a special memory area known as string constant pool inside the Heap memory.**

Why java uses concept of string literal?

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

2) By new keyword

String s=new String("Welcome");//creates two objects

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

Immutable String in Java

In java, **string objects are immutable**. Immutable simply means unmodifiable or unchangeable.

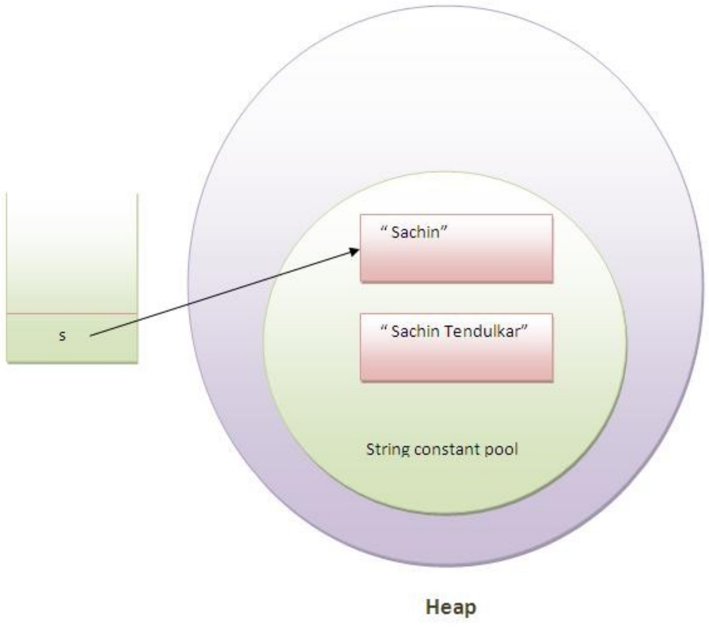
Once string object is created its data or state can't be changed but a new string object is created.

Let's try to understand the immutability concept by the example given below:

```
class Simple{
 public static void main(String args[]){
   String s="Sachin";
   s.concat(" Tendulkar");//concat() method appends the string at the end
   System.out.println(s);//will print Sachin because strings are immutable o
bjects
 }
}
```

Output: Sachin

Now it can be understood by the diagram given below. Here Sachin is not changed but a new object is created with sachintendulkar. That is why string is known as immutable.



As you can see in the above figure that two objects are created but s reference variable still refers to "Sachin" not to "Sachin Tendulkar".

But if we explicitly assign it to the reference variable, it will refer to "Sachin Tendulkar" object. For example:

```
class Simple{
 public static void main(String args[]){
   String s="Sachin";
   s=s.concat(" Tendulkar");
   System.out.println(s);
 }
}
```

Output:Sachin Tendulkar

In such case, s points to the "Sachin Tendulkar". Please notice that still sachin object is not modified.

Why string objects are immutable in java?

Because java uses the concept of string literal. Suppose there are 5 reference variables, all refers to one object "sachin". If one reference variable changes the value of the object, it will be affected to all the reference variables. That is why string objects are immutable in java.

String Comparison

The relational operators like >,<,>=,<=, == and != cannot be used to compare the strings. On the other hand, method like: equals(), compareTo() should be used in string comparision.

We can compare two given strings on the basis of content and reference.

There are three ways to compare String objects:

1. By equals() method
2. By = = operator
3. By compareTo() method

1) By equals() method

equals() method compares the original content of the string. It compares values of string for equality. String class provides two methods:

- **public boolean equals(Object another)**{} compares this string to the specified object.
- **public boolean equalsIgnoreCase(String another)**{} compares this String to another String, ignoring case.

```
class Simple{
 public static void main(String args[]){

   String s1="Sachin";
   String s2="Sachin";
   String s3=new String("Sachin");
   String s4="Saurav";
   System.out.println(s1.equals(s2));//true
   System.out.println(s1.equals(s3));//true
   System.out.println(s1.equals(s4));//false
 }
}
```

Output:

```
true
true
false
```

```
//Example of equalsIgnoreCase(String) method
class Simple{
 public static void main(String args[]){

   String s1="Sachin";
   String s2="SACHIN";

   System.out.println(s1.equals(s2));//false
   System.out.println(s1.equalsIgnoreCase(s2));//true
 }
}
```

Output:

```
false
true
```

2) By == operator

The = = operator compares references not values.

//Example of == operator

```
class Simple{
 public static void main(String args[]){

   String s1="Sachin";
   String s2="Sachin";
   String s3=new String("Sachin");

System.out.println(s1==s2);//true (because both refer to same instance)
System.out.println(s1==s3);//false(because s3 refers to instance created in non
pool)
 }
}
```

Output:

```
true
false
```

When an object is created by JVM, it returns the memory address of the object as a hexadecimal number, which is called **object reference**. When a new object is created, a new reference number is allotted to it. It means every object will have a unique reference.

What is object Reference?
Object reference is a unique hexadecimal number representing the memory address of the object. It is useful to access the members of the object.

Inspite of the fact that both the strings s1 and s3 are same, we are getting *false* output. Let us take the first statement:

String s1 = “Sachin”;

When JVM execute the preceding statement, it creates an object on string constant pool memory of heap and store “Sachin” in it. A reference number, say 5e75b5 is allotted for this object.

Similarly when the following statement is executed,

String s3 = new String(“Hello”);

JVM creates another object and hence allots another reference number, say 20151f. So the statement, if (s1==s3) will compare these reference number, i.e. 5e75b5 and 20151f. Both are not same and hence the output will be “false”. Actually, we should compare the contents of the String objects, not their references. This is the reason == is not used to compare the strings.

3) By compareTo() method:

compareTo() method compares values and returns an int which tells if the values compare less than, equal, or greater than.

Suppose s1 and s2 are two string variables.If:

- **s1 == s2** :0
- **s1 > s2** :positive value
- **s1 < s2** :negative value

```
//Example of compareTo() method:

class Simple{
 public static void main(String args[]){

   String s1="Sachin";
   String s2="Sachin";
   String s3="Ratan";

   System.out.println(s1.compareTo(s2));//0
   System.out.println(s1.compareTo(s3));//1(because s1>s3)
   System.out.println(s3.compareTo(s1));//-1(because s3 < s1 )
 }
}
```

Output:

```
0
1
-1
```

String Concatenation in Java

Concating strings form a new string i.e. the combination of multiple strings.

There are two ways to concat string objects:

1. By + (string concatenation) operator
2. By concat() method

1) By + (string concatenation) operator

String concatenation operator is used to add strings.For Example:

```
//Example of string concatenation operator

class Simple{
 public static void main(String args[]){

   String s="Sachin"+" Tendulkar";
   System.out.println(s);//Sachin Tendulkar
 }
}
```

Output: Sachin Tendulkar

The compiler transforms this to:

```
String s=(new StringBuilder()).append("Sachin").append(" Tendulkar).toString();
```

String concatenation is implemented through the StringBuilder(or StringBuffer) class and its append method. String concatenation operator produces a new string by appending the second operand onto the end of the first operand. The string concatenation operator can concat not only string but primitive values also. For Example:

```
class Simple{
 public static void main(String args[]){

   String s=50+30+"Sachin"+(40+40);
   System.out.println(s);//80Sachin4040
 }
}
```

Output:80Sachin80

Note: If either operand is a string, the resulting operation will be string concatenation. If both operands are numbers, the operator will perform an addition.

2) By concat() method

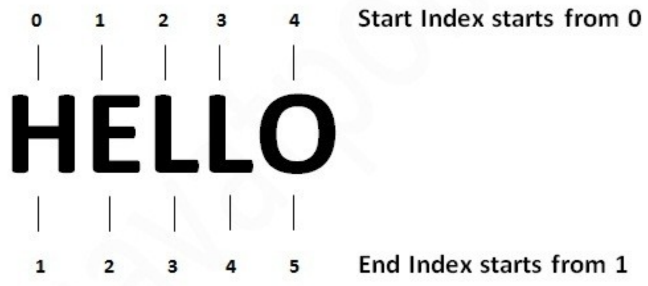
concat() method concatenates the specified string to the end of current string.

Syntax:public String concat(String another){}

```
//Example of concat(String) method

class Simple{
 public static void main(String args[]){

   String s1="Sachin ";
   String s2="Tendulkar";

   String s3=s1.concat(s2);

          System.out.println(s3);//Sachin Tendulkar
         }
       }
```

Output:Sachin Tendulkar

Substring in Java



A part of string is called **substring**. In other words, substring is a subset of another string.

In case of substring startIndex starts from 0 and endIndex starts from 1 or startIndex is inclusive and endIndex is exclusive.

You can get **substring** from the given String object by one of the two methods:

1. **public String substring(int startIndex):** This method returns new String object containing the substring of the given string from specified startIndex (inclusive).
2. **public String substring(int startIndex,int endIndex):** This method returns new String object containing the substring of the given string from specified startIndex to endIndex.

In case of string:

- **startIndex:**starts from index 0(inclusive).
- **endIndex:**starts from index 1(exclusive).

Example of java substring

```
//Example of substring() method

class Simple{
 public static void main(String args[]){

   String s="Sachin Tendulkar";
   System.out.println(s.substring(6));//Tendulkar
   System.out.println(s.substring(0,6));//Sachin
}
        }
```

```
Output:Tendulkar
       Sachin
```

Methods of String class in Java

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 strings etc.

Let's see the important methods of String class.

Method	**Description**
1)public boolean equals(Object anObject)	Compares this string to the specified object.

2)public boolean equalsIgnoreCase(String another)	Compares this String to another String, ignoring case.
3)public String concat(String str)	Concatenates the specified string to the end of this string.
4)public int compareTo(String str)	Compares two strings and returns int
5)public int compareToIgnoreCase(String str)	Compares two strings, ignoring case differences.
6)public String substring(int beginIndex)	Returns a new string that is a substring of this string.
7)public String substring(int beginIndex,int endIndex)	Returns a new string that is a substring of this string.
8)public String toUpperCase()	Converts all of the characters in this String to upper case
9)public String toLowerCase()	Converts all of the characters in this String to lower case.
10)public String trim()	Returns a copy of the string, with leading and trailing whitespace omitted.
11)public boolean startsWith(String prefix)	Tests if this string starts with the specified prefix.
12)public boolean endsWith(String suffix)	Tests if this string ends with the specified suffix.
13)public char charAt(int index)	Returns the char value at the specified index.
14)public int length()	Returns the length of this string.
15)public String intern()	Returns a canonical representation

	for the string object.
16)public byte[] getBytes()	Converts string into byte array.
17)public char[] toCharArray()	Converts string into char array.
18)public static String valueOf(int i)	converts the int into String.
19)public static String valueOf(long i)	converts the long into String.
20)public static String valueOf(float i)	converts the float into String.
21)public static String valueOf(double i)	converts the double into String.
22)public static String valueOf(boolean i)	converts the boolean into String.
23)public static String valueOf(char i)	converts the char into String.
24)public static String valueOf(char[] i)	converts the char array into String.
25)public static String valueOf(Object obj)	converts the Object into String.
26)public void replaceAll(String firstString,String secondString)	Changes the firstString with secondString.

First seven methods have already been discussed. Now Let's take the example of other methods:

toUpperCase() and toLowerCase() method

```
class Simple{
 public static void main(String args[]){

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

```
Output:SACHIN
       sachin
       Sachin
```

trim() method

```
class Simple{
 public static void main(String args[]){

   String s="  Sachin  ";
   System.out.println(s);//  Sachin
   System.out.println(s.trim());//Sachin
 }
}
```

```
Output: Sachin
       Sachin
```

startsWith() and endsWith() method

```
class Simple{
 public static void main(String args[]){

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

```
Output:true
       true
```

charAt() method

```
class Simple{
 public static void main(String args[]){

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

```
Output:S
       h
```

length() method

```
class Simple{
 public static void main(String args[]){

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

Output:6

StringBuffer class:

The StringBuffer class is used to created mutable (modifiable) string. The StringBuffer class is same as String except it is mutable i.e. it can be changed.

Note: StringBuffer class is thread-safe i.e. multiple threads cannot access it simultaneously .So it is safe and will result in an order.

Commonly used Constructors of StringBuffer class:

1. **StringBuffer():** creates an empty string buffer with the initial capacity of 16.
2. **StringBuffer(String str):** creates a string buffer with the specified string.
3. **StringBuffer(int capacity):** creates an empty string buffer with the specified capacity as length.

Commonly used methods of StringBuffer class:

1. **public synchronized StringBuffer append(String s):** is used to append the specified string with this string. The append() method is overloaded like append(char), append(boolean), append(int), append(float), append(double) etc.
2. **public synchronized StringBuffer insert(int offset, String s):** is used to insert the specified string with this string at the specified position. The insert() method is overloaded like insert(int, char), insert(int, boolean), insert(int, int), insert(int, float), insert(int, double) etc.
3. **public synchronized StringBuffer replace(int startIndex, int endIndex, String str):** is used to replace the string from specified startIndex and endIndex.
4. **public synchronized StringBuffer delete(int startIndex, int**

endIndex): is used to delete the string from specified startIndex and endIndex.
5. **public synchronized StringBuffer reverse():** is used to reverse the string.
6. **public int capacity():** is used to return the current capacity.
7. **public char charAt(int index):** is used to return the character at the specified position.
8. **public int length():** is used to return the length of the string i.e. total number of characters.
9. **public String substring(int beginIndex):** is used to return the substring from the specified beginIndex.
10. **public String substring(int beginIndex, int endIndex):** is used to return the substring from the specified beginIndex and endIndex.

What is mutable string?

A string that can be modified or changed is known as mutable string. StringBuffer and StringBuilder classes are used for creating mutable string.

simple example of StringBuffer class by append() method

The append() method concatenates the given argument with this string.

```
class A{

public static void main(String args[]){

StringBuffer sb=new StringBuffer("Hello ");
sb.append("Java");//now original string is changed

System.out.println(sb);//prints Hello Java
}
}
```

Example of insert() method of StringBuffer class

The insert() method inserts the given string with this string at the given position.

```
class A{
public static void main(String args[]){
```

```
StringBuffer sb=new StringBuffer("Hello ");
sb.insert(1,"Java");//now original string is changed
System.out.println(sb);//prints HJavaello
}
}
```

Example of replace() method of StringBuffer class

The replace() method replaces the given string from the specified beginIndex and endIndex.

```
class A{
public static void main(String args[]){
StringBuffer sb=new StringBuffer("Hello");
sb.replace(1,3,"Java");

System.out.println(sb);//prints HJavalo
}
}
```

Example of delete() method of StringBuffer class

The delete() method of StringBuffer class deletes the string from the specified beginIndex to endIndex.

```
class A{
public static void main(String args[]){
StringBuffer sb=new StringBuffer("Hello");
sb.delete(1,3);
System.out.println(sb);//prints Hlo
}
}
```

Example of reverse() method of StringBuffer class

The reverse() method of StringBuilder class reverses the current string.

```
class A{
public static void main(String args[]){
StringBuffer sb=new StringBuffer("Hello");
sb.reverse();
System.out.println(sb);//prints olleH
}
}
```

StringBuilder class:

The ***StringBuilder*** class is used to create mutable (modifiable) string. The *StringBuilder* class is same as *StringBuffer* class except that it is **non-synchronized.** It is available since JDK1.5.

Commonly used Constructors of StringBuilder class:

1. **StringBuilder():** creates an empty string Builder with the initial capacity of 16.
2. **StringBuilder(String str):** creates a string Builder with the specified string.
3. **StringBuilder(int length):** creates an empty string Builder with the specified capacity as length.

How to create Immutable class?

There are many immutable classes like String, Boolean, Byte, Short, Integer, Long, Float, Double etc. In short, all the wrapper classes and String class is immutable. We can also create immutable class by creating final class that have final data members as the example given below:

Example to create Immutable class

In this example, we have created a final class named Employee. It have one final datamember, a parameterized constructor and getter method.

```
public final class Employee{
final String pancardNumber;
public Employee(String pancardNumber){
this.pancardNumber=pancardNumber;
}
public String getPancardNumber(){
return pancardNumber;
}
}
```

The above class is immutable because:

- The instance variable of the class is final i.e. we cannot change the value of it after creating an object.

- The class is final so we cannot create the subclass.
- There is no setter methods i.e. we have no option to change the value of the instance variable.

These points makes this class as immutable.

Understanding 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. depends on your implementation.

Advantage of the 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.

Understanding problem without toString() method

Let's see the simple code that prints reference.

```
class Student{
 int rollno;
 String name;
 String city;

 Student(int rollno, String name, String city){
 this.rollno=rollno;
 this.name=name;
 this.city=city;
 }
  public static void main(String args[]){
    Student s1=new Student(101,"Sachin","Dehradun");
    Student s2=new Student(102,"Vijay","Delhi");
    System.out.println(s1);//compiler writes here s1.toString()
    System.out.println(s2);//compiler writes here s2.toString()
  }
 }
```

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 toString() method

Now let's see the real example of toString() method.

```
class Student{
 int rollno;
 String name;
 String city;

 Student(int rollno, String name, String city){
 this.rollno=rollno;
 this.name=name;
 this.city=city;
 }

 public String toString(){//overriding the toString() method
  return rollno+" "+name+" "+city;
 }
 public static void main(String args[]){
   Student s1=new Student(101,"Raj","Dehradun");
   Student s2=new Student(102,"Vijay","Delhi");
   System.out.println(s1);//compiler writes here s1.toString()
   System.out.println(s2);//compiler writes here s2.toString()
 }
}
```

Recommended Reading

Q1. What is String?

String is a class in java which is present in java.lang package. According to Oracle docs,
The String class represents character strings. Strings are constant, their values can not be changed after they are created.

Q2 Is String immutable in java?

Yes, String class is immutable in java. Immutable means once the object is created, its value can not be changed.

Q3 Is String a keyword in java ?

No, String is not a keyword in java.

Q4 How many objects are created using *new* keyword?

```
String str = new String("JavaHungry");
```

Two objects are created by the above statement. One object in the heap memory and one object in the String constant pool.

Q5 How to convert String to char Array?

You can convert String to char Array using *toCharArray()* method.

Q6 How many different ways you can create a String object?

You can create a String object using two ways. First is using *new* operator and second is using string *literal*. Objects created using *new* operator are stored in the heap memory while string *literals* are stored in the string constant pool.

```
String str = "javahungry";  // String literal
String str = new String("javahungry"); // using new operator
```

Q7. Which class is final among String, StringBuilder and StringBuffer?

All are final classes.

Q8 Is String primitive type or object (derived) type in java?

String is object(derived) type in java.

Q9 Can we use String in switch statement?

Yes, you can use String in switch statement in java 7. Prior to java 7 , you had to use if-else statements to achieve the task.

Q10. What is the difference between String, StringBuffer and StringBuilder in java?

String
<u>String is *immutable*</u> (once created can not be changed) object. The object created as a String is stored in the **Constant String Pool**.
Every immutable object in Java is thread-safe, that implies String is also thread-safe . String can not be used by two threads simultaneously.
String once assigned can not be changed.

```
String  demo = "hello" ;
```

// The above object is stored in constant string pool and its value can not be modified.

```
demo="Bye" ;
```

//new "Bye" string is created in constant pool and referenced by the demo variable

// "hello" string still exists in string constant pool and its value is not <u>overrided</u> but we lost reference to the "hello"string

StringBuffer

StringBuffer is 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.

```
StringBuffer demo1 = new StringBuffer("Hello") ;
```

// The above object stored in heap and its value can be changed .

```
demo1 = new StringBuffer("Bye");
```

// Above statement is right as it modifies the value which is allowed in the StringBuffer

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 not thread safe.**
StringBuilder is fast as it is not thread safe .

```
StringBuilder demo2= new StringBuilder("Hello");
```

// The above object too is stored in the heap and its value can be modified

```
demo2=new StringBuilder("Bye");
```

// Above statement is right as it modifies the value which is allowed in the StringBuilder

	String	***StringBuffer***	***StringBuilder***
Storage Area |	Constant String Pool	Heap	Heap

Modifiable	No (immutable)	Yes(mutable)	Yes(mutable)
Thread Safe	Yes	Yes	No
Performance	Slow	Slow but faster than String	Fast

Q11 Explain the difference between below statements:

```
String str = new String("abc");
```

```
String  str =  "abc";
```

In the first statement, String str = new String("abc");
JVM will create one object in the heap memory. Another object in the String constant pool, if the object is not present. Otherwise,if present in the String constant pool ,it will return the reference to it.

In the second statement, String str = "abc";
JVM checks for the string "abc" in the String constant pool. If the string is not present in the constant pool then it will create a new String object and stores it in pool.
If the string "abc" is found in string constant pool , then it simply creates a reference to it but does not create a new object.

Q12 How many objects will be created for the following code:

```
String str1 = "abc";


String str2 = new String("abc");
```

Two objects are created. Object created using *new* operator is stored in the heap memory (str2).
Object created using String literal str1 is stored in the string constant pool.

Q13 How many objects will be created for the following code:

```
String str1 = "abc";


String str2 = "abc";
```

Only one object is created. String str1 will create a new object in String constant pool, while String str2 will create a reference to the String str1.

Q14 How many objects will be created for the following code:

```
String str1 = new String("abc");

String str2 = new String("abc");
```

Three objects are created. For the first statement(str1) two objects are created one in String constant pool and one in heap memory.
But for the second statement(str2), compulsory 1 new object is created in heap memory but no new object is created in string constant pool as it is already present.

Hence , a total of 2+1 = 3 objects are created.

Q15 What is String intern() method?

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

The task of intern() method is to put String (which is passed to the intern method) into string constant pool.

Q16 Why is String immutable in java ?

There are various reasons to make String immutable in java.
1. Security : String is made immutable to help increase the Security. Sensitive data like username,password can be stored as the Strings can't be modified once created.

2. Class loading : String objects are used for Class loading. It is possible that wrong class has been loaded in the JVM, if the String is mutable i.e modifiable.

3. Thread Safe : Immutable Strings are thread-safe. Synchronization is not required when we use them in the multithreading environment.

Q17 What is the difference between Java String and C,C++ Strings ?

In C and Java both programming language treat String object as char Array.

Java String is an object while C String is a NULL terminated character array. Java String object allows calling different methods like toUpperCase(), length(), substring().

Q18 Is it possible to call String class methods using String literals?

Yes, It is possible to call String class methods using String literals. For example

```
"javahungry".indexOf(u)
"javahungry".charAt(0)
"javahungry".compareTo("javahungry")
```

Q19 How to Split String in java?

You can use split() method of java.lang.String class or StringTokenizer to split a comma separated String. String split() method is easier to use and better because it expects a regular expression and returns an array of String which you can manipulate in the program code.

Q20 What is String Constant Pool? Why java provided String Constant pool as we can store String in the heap memory?

String constant pool is the memory space allocated in the heap memory to store the objects which are
created using String literals. String constant pool is unique, there are no two String o objects which has the same value(content).

Q21. Why String Constant Pool ?

String constant pool increases the reusability of the existing String objects.
It also saves memory as no two objects with same content are created.

Q22 Why char Array is preferred over String in storing passwords?

One of the main reason to prefer char Array over String is security risk of stealing passwords. Since String are reusable in the constant pool , there are high chances that they remain in the memory for the long duration. Anyone who has access to the memory dump can find the password in clear text. That's why password should be encrypted.

Q23 What is Character encoding? Explain the difference between UTF-16

and UTF-8?

When you want to represent Character using bytes, character encoding is used.

The UTF-16 uses 2 bytes or 16 bits to represent a character while UTF-8 uses 1 byte or 8 bits to represent a character.

Q24. Find All Possible Combinations Of String In Java : Code With Example

All combination of string in java is the companion problem to find permutation of the string . The combination generated from the algorithm has range in length from one to the length of the string. Two combinations that differ only in ordering of their characters are the same combination. In other words, "12" and "31" are different combinations from the input string "123", but "21" is the same as "12". Let us understand the string combination algorithm with example :

If **the** input is "wxyz" **then** **the** output of the string **is** " w wx wxy wxyz wxz wy wyz wz x xy xyz xz y yz z "

Logic:

Start with an empty output string and the first character of the input as the input start position. For a given position , select all the letters sequentially from the input start position to the last letter in the input string. For each selected letter , append it to the output string , print the combination and then produce all other combinations starting with this sequence by recursively calling the generating function with the input start position set to the next letter after the one we have just selected. Delete the character we appended to the output after returning from the recursive call so that it create room for next character we select .

String Combinations algorithm Pseudo Code :

```
For each letter from input start position to end
   of input string
         *  Append the letter to the output string
         *  Print letters in output string
If the current letter isn't the last in the input string
         * Generate remaining combinations
           starting at next position with
           iteration starting at next letter
           beyond  the letter just selected

Delete the last character of the output string
```

```
public class Combinations {
   private StringBuilder output = new StringBuilder();
   private final String inputstring;
   public Combinations( final String str ){
      inputstring = str;
      System.out.println("The input string  is  : " + inputstring);
   }
```

```
    public static void main (String args[])
    {
        Combinations combobj= new Combinations("wxyz");
        System.out.println("");
        System.out.println("");
        System.out.println("All possible combinations are :  ");
        System.out.println("");
        System.out.println("");
        combobj.combine();
    }

    public void combine() { combine( 0 ); }
    private void combine(int start ){
        for( int i = start; i < inputstring.length(); ++i ){
            output.append( inputstring.charAt(i) );
            System.out.println( output );
            if ( i < inputstring.length() )
            combine( i + 1);
            output.setLength( output.length() - 1 );
        }
    }
}
```

Q25. Write a java program to check whether two given strings are anagram.

[If two strings contain same set of characters but in different order then the two strings are called anagram.]

for example :

1. **"now"** and **"own"**
2. **"ton"** and **"not"**
3. **"fiber"** and **"brief"**

Now we know what does anagram mean. Let understand the question by writing example.

Input : "now","own"
Output : true

Input : "ton" , "not"
Output : true

Input : "hello" , "he"
Output : false

Pseudo Code for Anagram Program in java using sort() and equals() method:

```
1. Convert the two strings into uppercase and remove all white spaces.
2. Convert the two strings into char arrays using toCharArray().
3. Sort the two character arrays using sort() method of java.util.Arrays class.
4. After sorting,  we compare both the arrays using equals() method.
```

```java
import java.util.*;
import java.io.*;
public class Anagram
{
    public static void main (String[] args) throws java.lang.Exception
    {
        boolean result = isAnagram("now","own");
        System.out.println(result);
    }
    public static boolean isAnagram(String first, String second)
    {
        // remove all whitespaces and convert strings to lowercase
        first  = first.replaceAll("\\s", "").toLowerCase();
        second = second.replaceAll("\\s", "").toLowerCase();

        /* check whether string lengths are equal or not,
        if unequal then not anagram */
        if (first.length() != second.length())
        return false;

        // convert string to char array
        char[] firstArray = first.toCharArray();
        char[] secondArray = second.toCharArray();

        // sort both the arrays
        Arrays.sort(firstArray);
        Arrays.sort(secondArray);

        // checking whether both strings are equal or not
        return Arrays.equals(firstArray,secondArray);
    }
}
```

Pseudo Code for Anagram Program in java using Iterative method:

1. In this method we go on checking whether each character of first string is present in second string.
2. If the character is present in second string, we remove the character from the second string, and proceed to the next character of first string.
3. If any character of first string is not present in second string, we break the loop and result would be first string is not anagram of second string.

```
import java.util.*;
import java.lang.*;
import java.io.*;

public class Anagram
{

 public static void main (String[] args) throws java.lang.Exception
 {
  boolean result = isAnagram("now","own");
  System.out.println(result);
 }
 public static boolean isAnagram(String first, String second)
 {
  // remove all whitespaces and convert strings to lowercase
  first  = first.replaceAll("\\s", "").toLowerCase();
  second = second.replaceAll("\\s", "").toLowerCase();
  /* check whether string lengths are equal or not,
     if unequal then not anagram */
  if (first.length() != second.length())
   return false;

  // convert first string to char array
  char[] firstArray = first.toCharArray();

  // check whether each character of firstArray is present in second string
  for (char c : firstArray)
  {
   int index = second.indexOf(c);

   // indexOf function returns -1 if the character is not found
   if (index == -1)
    return false;

   // if character is present in second string, remove that character from
second string
   second  = second.substring(0,index) + second.substring(index+1,
second.length());
  }
  return true;
 }
}
```