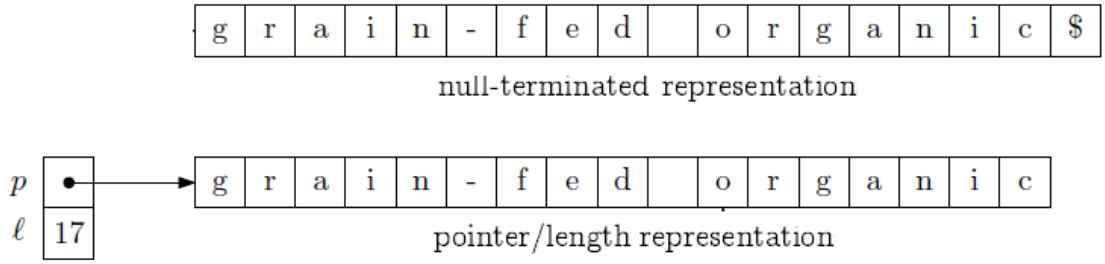
Strings data structures

A string is collection of characters grouped together. For example, "hello" is a string consisting of the characters 'h', 'e', 'l', 'l', and 'o'.

In most programming languages, strings are generally understood as a data type are built in as part of the language and they take one of two basic representations as illustrated in following figures:

1**. null-terminated representation**, in this, strings are represented as an array of characters that ends with the special null terminator $.

2. **pointer/length representation,** in this representation a string is represented as (a pointer to) an array of characters along with a integer that stores the length of the string. The pointer/length representation is more efficient for some operations. For example, in the pointer/length representation, determining the length of the string takes constant time, since it is already stored.



String in java

In java, string is basically an object that represents sequence of char values. it is often implemented as:

- an array that stores a sequence of elements or as a reference type , mean string variable holds the *address* of the memory location where the actual body of the string is stored.
  For example:

```
char[] ch={'j','a','v','a','i','s','o','o','p'};
String s=new String(ch);
```

is same as:
String s="javaisoop";

- Java Strings can't change the characters, but string variables can point to different strings:

```
String s;
    s = "java language";
    s = "java is oop";
```

Q: How to create String object?

There are two ways to create String object:

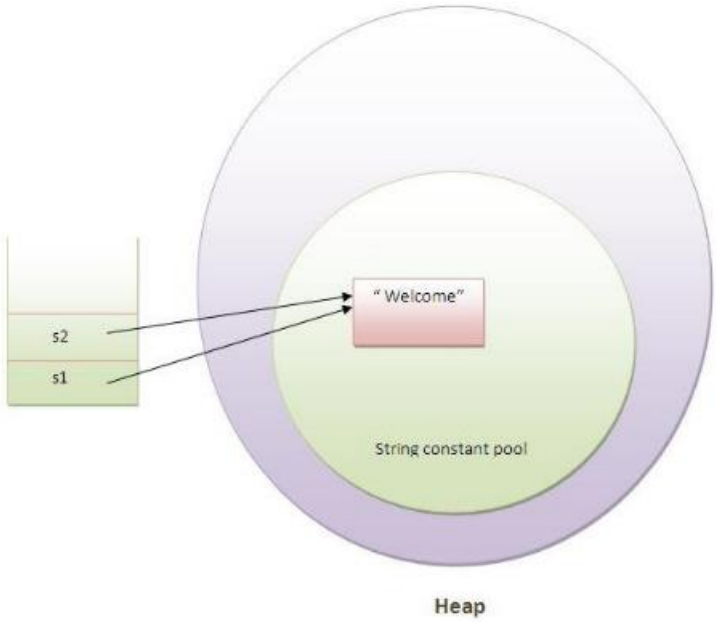
1. By string literal
2. By new keyword

String Literal

Java String literal is created by using double quotes. For Example:
String s="welcome";

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 is returned. If string doesn't exist in the pool, a new string instance is created and placed in the pool. For example:

1. String s1="Welcome";
2. String s2="Welcome";//will not create new instance



In the above example only one object will be created. Firstly JVM will not find any string object with the value "Welcome" in string constant pool, so it will create a new object. After that it will find the string with the value "Welcome" in the pool, it will not create new object but will return the reference to the same instance.

Note: String objects are stored in a special memory area known as string constant pool.

Q: Why java uses concept of string literal?
Ans.: To make Java more memory efficient (because no new objects are created if it exists already in string constant pool).

new keyword

In the following shown example of create string by using new keyword:

Example

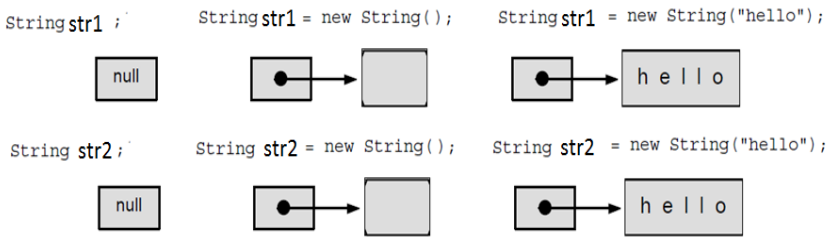
CODE: **MEMORY:**

```
String  st = new  String ("hello");
```

Each time a new declaration a compiler would create different object in memory

Example:

```
String str1 = new String("Hello");
String str2 = new String("Hello");
```



In this case compiler would create two different object in memory having the same text.

Q: **What is the output of the following program?**

```
public class StringExample{
public static void main(String args[]){
   String s1="java";                      //creating string by java string literal
   char ch[]={'s','t','r','i','n','g','s'};
   String s2=new String(ch);              //converting char array to string
   String s3=new String("example");    //creating java string by new keyword
   System.out.println(s1);
   System.out.println(s2);
   System.out.println(s3);
 }
}
```

Java String class methods

The java.util.String class provides many useful methods to perform operations on sequence of char values.

Method	Description
char charAt(int index)	returns char value for the particular index
int length()	returns string length
String substring(int beginIndex)	returns substring for given begin index
String substring(int beginIndex, int endIndex)	returns substring for given begin index and end index
boolean contains(CharSequence s)	returns true or false after matching the sequence of char value
boolean equals(Object another)	checks the equality of string with object
boolean isEmpty()	checks if string is empty
String concat(String str)	concatinates specified string
String replace(char old, char new)	replaces all occurrences of specified char value
String replace(CharSequence old, CharSequence new)	replaces all occurrences of specified CharSequence
static String equalsIgnoreCase(String another)	compares another string. It doesn't check case.
String[] split(String regex)	returns splitted string matching regex

String[] split(String regex, int limit)	returns splitted string matching regex and limit
int indexOf(int ch)	returns specified char value index
int indexOf(int ch, int fromIndex)	returns specified char value index starting with given index
int indexOf(String substring)	returns specified substring index
int indexOf(String substring, int fromIndex)	returns specified substring index starting with given index
String toLowerCase()	returns string in lowercase.
String toUpperCase()	returns string in uppercase.
String trim()	removes beginning and ending spaces of this string.
static String valueOf(int value)	converts given type into string. It is overloaded.

Assignment of string:

While primitive types are *value types*:

```
int  x = 5;
int  y = x;
```

} The memory of x and y both

But with *reference type*:

```
String  x = "a";
String  y =  x;
```

} The memory of x and y both contain a *pointer* to the character "a"