CHAPTER

9

STRINGS

Most of the data that transmits on Internet will be in the form of groups of characters. Such groups of characters are called 'strings'. For example, in a business order form, a person enters details like his name, credit card number, address, etc., which are all nothing but strings only. So a string represents a group of characters.

In C/C++ languages, a string represents an array of characters, where the last character will be '\0' (called null character). This last character being '\0' represents the end of the string. But this is not valid in Java. In Java, a string is an object of String class. It is not a character array. In Java, we got character arrays also, but strings are given a different treatment because of their extensive use on Internet. JavaSoft people have created a class separately with the name 'String' in java.lang (language) package with all necessary methods to work with strings.

Even though, String is a class, it is used often in the form of a data type, as:

```
String s = "Java";
```

Here, s is a variable of the data type 'String'.

Important Interview Question

Is String a class or data type?

String is a class in java.lang package. But in Java, all classes are also considered as data types. So we can take String as a data type also.

Can we call a class as a data type?

Yes, a class is also called 'user-defined' data type. This is because a user can create a class.

Creating Strings

There are three ways to create strings in Java:

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

```
String s; //declare String type variable
        s = "Hello"; //assign a group of characters to it
```

- Preceding two statements can be combined and written as:

```
String s = "Hello";
```

In this, case JVM creates an object and stores the string: "Hello" in that object. This object is referenced by the variable 's'. Remember, creating object means allotting memory for storing data.

- We can create an object to String class by allocating memory using new operator. This is just like creating an object to any class, like given here:

```
String s = new String("Hello");
```

Here, we are doing two things. First, we are creating object using new operator. Then, we are storing the string: "Hello" into the object.

- The third way of creating the strings is by converting the character arrays into strings. Let us take a character type array: arr[ ] with some characters, as:

```
char arr[ ] = {'c','h','a','i','r','s'};
```

- Now create a string object, by passing the array name to it, as:

```
String s = new String(arr);
```

Now the String object 's' contains the string: "chairs". This means all the characters of the array are copied into the string. If we do not want all the characters of the array into the string, then we can also mention which characters we need, as:

```
String s = new String(arr, 2,3);
```

Here, starting from 2nd character a total of 3 characters are copied into the string s. The original characters are c-h-a-i-r-s. Since counting starts from 0, the 0th character in the array is 'c' and the 2nd character is 'a'. Starting from 'a', a total of three characters implies 'air'. So these three characters are copied into the string s.

String Class Methods

Let us have a look at which methods are available in String class and how they can be used:

- String concat(String s)

Here, 'concat' is the method name. Since this method belongs to 'String' class, it can be called using a String class object as s1.concat(). Here, 's1' is a String class object. Please observe the 'String s' in the parenthesis after the method name. It represents that you should pass another String object to the concat() method while it is called. Something like this, s1.concat(s2). Again observe the word 'String' before the method name. This indicates that the method returns a String object, as a result.

So the preceding method can be used as:

```
String s3 = s1.concat(s2);
```

Now concat() method concatenates or joins two strings (s1 and s2) and returns a third string (s3) as a result. So, if s1="Hydera" and s2="bad", then we can expect s3 will be "Hyderabad".

The same concatenation can be done by using '+' operator which is called 'String concatenation operator'.

For example, we can use '+' as: s1+s2+s3. Here the three strings are joined by '+'.

- `int length()`: This method returns the length or number of characters of a string. For example, s1.length() gives the number of characters in the string s1.
- `char charAt(int i)`: This method returns the character at the specified location i. Suppose we call this method as: s1.charAt(5), then it gives the 5th character in the string s1.
- `int compareTo(String s)`: This method is useful to compare two strings and to know which string is bigger or smaller. This should be used as: s1.compareTo(s2). Now s1 and s2 are compared. If s1 and s2 strings are equal, then this method gives 0. If s1 is greater than s2, then it returns a positive number. If s1 is less than s2, then it returns a negative number.

When we arrange the strings in dictionary order, which ever string comes first is lesser than the string that comes next. Thus "Box" is lesser than "Boy". This method is case sensitive. This means, "BOX" and "box" are not same for this method.

- `int compareToIgnoreCase(String s)`: This is same as 'compareTo()' method but this does not take the case of strings into consideration. This means "BOX" and "box" look same for this method.
- `boolean equals(String s)`: This method returns true if two strings are same, otherwise false. This is case sensitive. We can use this method as: s1.equals(s2).
- `boolean equalsIgnoreCase(String s)`: This is same as preceding but it performs case insensitive comparison.
- `boolean startsWith(String s)`: This method returns true if a string is beginning with the sub string 's'. To call this method, we use the format s1.startsWith(s2). If s1 starts with s2 then, it returns true, otherwise false. This method is case sensitive.
- `boolean endsWith(String s)`: This method tests the ending of a string. If a string ends with the sub string 's', then it returns true, other wise false. This method is also case sensitive.
- `int indexOf(String s)`: This method is called in the form, `s1.indexOf(s2)`, and it returns an integer value. If s1 contains s2 as a sub string, then the first occurrence (position) of s2 in the string s1 will be returned by this method. For example, s1= "This is a book", s2= "is", to know the position of the substring s2 in s1, we write

```
int n = s1.indexOf(s2);
```

This method searches for substring "is" in the main string "This is a book". Please observe that the sub string is found at two positions, 2nd and 5th. But this method returns first position only, so it returns 2. If the substring is not found in the main string then this method returns some negative value.

- `int lastIndexOf(String s)`: This method is similar to the preceding method, but returns the last occurrence of the sub string 's' in the main string. If 's' is not found, then it returns negative value.
- `String replace(char c1, char c2)`: This method replaces all the occurrences of character 'c1' by a new character 'c2'. For example, s1 = "Hello" and we are using as s1.replace('l', 'x'), then the returned string will be "Hexxo".
- `String substring(int i)`: This method is useful to extract sub string from a main string. It returns a new string consisting of all characters starting from the position 'i' until the end of the string. For example, s1.substring(5) returns characters starting from 5th character till the end of s1.
- `String substring(int i1, int i2)`: This method returns a new string consisting of all characters starting from i1 till i2. The character at i2 is excluded. For example, s1.substring(5,10) returns the characters of s1 starting from 5th to 9th positions.
- `String toLowerCase()`: This method converts all characters of the string into lower case, and returns that lower-cased string.

- `String toUpperCase()`: This method converts all characters into upper case, and returns that upper-cased string.
- `String trim()`: This method removes spaces from the beginning and ending of a string. If a string is written as " Ravi Kiran ", the spaces before "Ravi" and after "Kiran" are unnecessary and should be removed. This is achieved by trim(). Note that this method does not remove the spaces in the middle of the string. For example, the space between "Ravi" and "Kiran" is not removed.
- `void getChars(int i1, int i2, char arr[ ], int i3)`: This method copies characters from a string into a character array. The characters starting from position i1 to i2-1 in the string are copied into the array 'arr' to a location starting from i3. Counting of characters in the string will start from 0th position.
- `String[ ] split(delimiter)`: This method is useful to break a string into pieces at places represented by the delimiter. The resultant pieces are returned into a String type array. Suppose, the delimiter is a comma, then the string is cut into pieces wherever a comma (,) is found. Let us have a look on program which will help us to understand strings.

Program 1: Write a program which will help us to understand how to create strings and how to use some important methods of String class.

```
//Demo of String class methods
class StrDemo
{
      public static void main(String args[ ])
      {
            //create strings in 3 ways
            String s1= "A book on Java";
            String s2= new String("I like it");
            char arr[]= {'D','r','e','a','m','t','e','c','h','
            ','P','r','e','s','s'};
            String s3= new String(arr);

            //display all the 3 strings
            System.out.println(s1);
            System.out.println(s2);
            System.out.println(s3);

            //find length of first string
            System.out.println("Length of s1= "+ s1.length());

            //concatenate two strings
            System.out.println("s1 and s2 joined= "+ s1.concat(s2));

            //concatenate three strings with +
            System.out.println(s1+" from "+ s3);

            //test if string s1 starts with A
            boolean x= s1.startsWith("A");
            if(x) System.out.println("s1 starts with \'A\'");
            else System.out.println("s1 does not start with \'A\'");

            //extract substring from s2, starting from 0th char to 6th char
            String p = s2.substring(0,7);

            //extract substring from s3, starting from 0th char to 8th char
            String q = s3.substring(0,9);

            //concatenate the strings p and q
            System.out.println(p+q);

            //convert s1 into uppercase and lowercase
            System.out.println("Upper s1= "+s1.toUpperCase());
            System.out.println("Lower s1= "+s1.toLowerCase());
      }
```

```
    }
```

Output:

```
C:\> javac StrDemo.java
C:\> java StrDemo
A book on Java
I like it
Dreamtech Press
Length of s1= 14
s1 and s2 joined= A book on JavaI like it
A book on Java from Dreamtech Press
s1 starts with 'A'
I like Dreamtech
Upper s1= A BOOK ON JAVA
Lower s1= a book on java
```

To understand how to convert a string into a character array, we can use `getChars()` method. If the string is 'str' and the array is 'arr', then we can use this method, for example, as:

```
str.getChars(7,21,arr,0);
```

It means we are copying from 7th character to 20th character (21st character is not copied) into 'arr'. These characters are copied starting from the position 0 onwards in the array. Now notice the Program 2.

Program 2: Let us take a string and copy some of the characters of the string into a character array 'arr' using `getChars()` method.

```
//Copying a string into an array
class Strcpy
{
    public static void main(String args[ ])
    {
        String str = "Hello, this is a book on Java";
        char arr[ ] = new char[20];

        //copy from str into arr starting from 7th character to 20th
        character
        str.getChars(7,21,arr,0);

        System.out.println(arr);
    }
}
```

Output:

```
C:\> javac Strcpy.java
C:\> java Strcpy
this is a book
```

Let us write another program to understand how to split a string into several pieces. For this purpose, `split()` method is used. This method is used as:

```
s = str.split("delimiter");
```

Here, "delimiter" is a string that contains a character or group of characters which represent where to split the string. For example, a space as a delimiter splits the string wherever a space is found. A comma splits it at a comma, and a colon splits it at a colon in the string. After splitting is done, the pieces are stored into a String type array 's'. This can be seen in the Program 3.

Program 3: Write a program for splitting a string into pieces wherever a space is found.

```
//Splitting a string
class Strsplit
{
      public static void main(String args[ ])
      {
            //take a string str which is to be broken
            String str = "Hello, this is a book on Java";

            //declare a string type array s to store pieces
            String s[ ];

            //split the string where a space is found in str
            s = str.split(" ");

            //display the pieces from s
            for(int i=0; i<s.length; i++)
            System.out.println(s[i]);
      }
}
```

Output:

```
C:\> javac Strsplit.java
C:\> java Strsplit
Hello,
this
is
a
book
on
Java
```

String Comparison

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

Program 4: Let us write a program to compare two strings using '==' operator, and see the result.

```
//String comparison using ==
class Strcompare
{
      public static void main(String args[ ])
      {
            String s1= "Hello";
            String s2= new String("Hello");

            if(s1==s2)
            System.out.println("Both are same");
            else System.out.println("Not same");
      }
}
```

Output:

```
C:\> javac Strcompare.java
C:\> java Strcompare
Not same
```

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.

Important Interview Question

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 s2 are same, we are getting wrong output in Program 4. Let us take the first statement:

```
String s1= "Hello";
```

When JVM executes the preceding statement, it creates an object on heap and stores "Hello" in it. A reference number, say 3e25a5 is allotted for this object.

Similarly when the following statement is executed,

```
String s2= new String("Hello");
```

JVM creates another object and hence allots another reference number, say `19821f`. So the statement, `if(s1==s2)` will compare these reference numbers, i.e., 3e25a5 and 19821f. Both are not same and hence the output will be "Not same". Actually, we should compare the contents of the String objects, not their references. This is the reason == is not used to compare the strings. See the Figure 9.1.

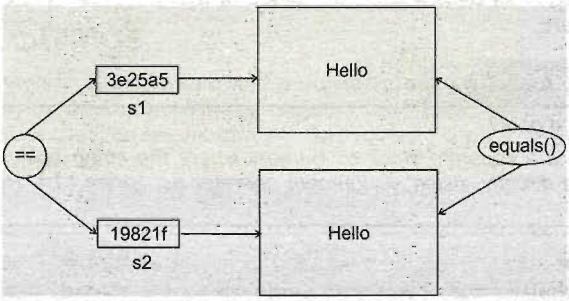

Figure 9.1 Comparing the strings using == and equals()

To compare the contents of String objects, we should use 'equals()'method, as shown here:

```
if(s1.equals(s2))
        System.out.println("Both are same");
        else System.out.println("Not same");
```

Now, we can get the output "Both are same" which is correct.

Important Interview Question

What is the difference between == and equals() while comparing strings? Which one is reliable?

== operator compares the references of the string objects. It does not compare the contents of the objects. equals() method compares the contents. While comparing the strings, equals() method should be used as it yields the correct result.

Program 5: Let us re-write the earlier program with a slight change in creation of strings.

```
//String comparison using ==
class Strcompare
{
      public static void main(String args[ ])
      {
            String s1= "Hello";
            String s2= "Hello";

            if(s1==s2)
            System.out.println("Both are same");
            else System.out.println("Not same");
      }
}
```

Output:

```
C:\> javac Strcompare.java
C:\> java Strcompare
Both are same
```

In Program 5, please notice how the strings are created as well as the output of the program also The first statement in the program is:

```
String s1 = "Hello";
```

Here, JVM creates a String object and stores "Hello" in it. Observe that we are not using nev operator to create the string. We are using assignment operator (=) for this purpose. So, afte creating the String object, JVM uses a separate block of memory which is called *string constant poo* and stores the object there.

Important Interview Question

What is a string constant pool?

String constant pool is a separate block of memory where the string objects are held by JVM. If a string object is created directly, using assignment operator as: String s1= "Hello", then it is stored in string constant pool.

When the next statement, `String s2 = "Hello";` is executed by the JVM, it searches in the strir constant pool to know whether the object with same content is already available there or not. Sinc the same object is already available there (which is s1), then JVM does not create another object. simply creates another reference variable (s2) to the same object, and copies the reference numb of s1 into s2. So, we have same value in s1 and s2. Hence, the output "Both are same". Please s Figure 9.2.

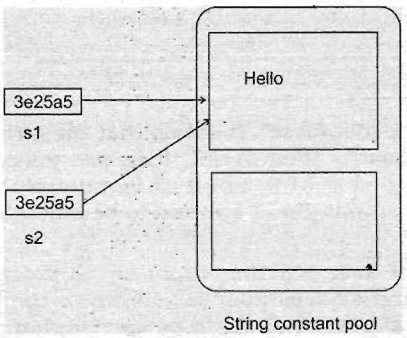

Figure 9.2 String comparison when the strings are created using assignment

Important Interview Question

Explain the difference between the following two statements:

1. String s = "Hello";

2. String s = new String("Hello");

In the first statement, assignment operator is used to assign the string literal to the String variable s. In this case, JVM first of all checks whether the same object is already available in the string constant pool. If it is available, then it creates another reference to it. If the same object is not available, then it creates another object with the content "Hello" and stores it into the string constant pool.

In the second statement, new operator is used to create the string object. In this case, JVM always creates a new object without looking in the string constant pool.

Immutability of Strings

We can divide objects broadly as, mutable and immutable objects. Mutable objects are those objects whose contents can be modified. Immutable objects are those objects, once created can not be modified. And String class objects are *immutable*. Let us take a program to understand whether the String objects are immutable or not.

Program 6: Write a program to test the immutability of strings.

```
//Immutable or Mutable?
class Test
{
      public static void main(String args[])
      {
            String s1 = "data";
            String s2 = "base";

            //join s1 and s2 and store in s1
            s1 = s1+s2;
            System.out.println(s1);

      }
}
```

Output:

```
C:\> javac Test.java
C:\> java Test
Database
```

Please observe the statement:

```
s1 = s1+s2;
```

And the output of the program, "database". It seems that the string s1 content is modified. Earlie s1 got "data" and s2 had "base". After s1+s2, they are joined and the total string become "database". This string is assigned to s1 again. If s1 is mutable, it gets the new string "database" This is what we can see in the output. So s1 appears to be mutable. But we learned that strings ar immutable.

s1 is definitely immutable. But then why the output of the program is like that? In the program JVM creates two objects, s1 and s2 separately, as shown in the Figure 9.3. When s1+s2 is done JVM creates a new object and stores the string "database" in that object. But it does not modify th contents of the string s1. After creating the new object, the reference s1 is adjusted to refer to tha new object. The point we should observe here is that the contents of the string s1 are not modified This is the reason, strings are called immutable. The old object that contains "data" has lost it reference. So it is called 'unreferenced object' and garbage collector will remove it from memory.

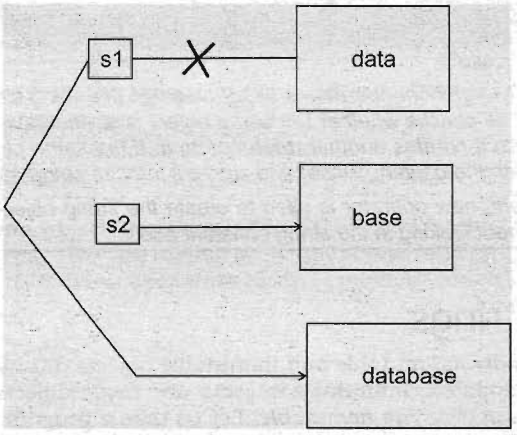

Figure 9.3 Immutability of String objects

Because String class objects are immutable, their contents can not be modified between variou applications. So, different applications can share the contents of String class objects. This is th reason why String objects are made immutable. To maintain the immutable nature, none of th String class methods should be able to modify the contents of the objects. So, JavaSoft people d not provide the methods in String class which modify the contents of the same object. Whenever v try to modify the contents, JVM will create a new object with modified data. It will not modify th same object. Also there should not be any scope to override those String class methods. This is th reason, String class is made 'final' class. The methods of a final class cannot be overridden. Th concepts of final class and method overriding will be discussed in detail later.

Let us write a program to search for a string in a given group of strings. In this program, first of a we accept the strings from keyboard and store them in a string type array str[ ]. Then we ask th user to enter the string to be searched. Let this string be 'search'. Then the logic is to compare th 'search' string with each string of the array str[ ] using equals() or equalsIgnoreCase() methods, as:

```
if(search.equalsIgnoreCase(str[i]))
{
     System.out.println("Found at position: "+ (i+1));
}
```

Here, we are making case insensitive comparison. If the search string is found in the array str[ ], then the position of the string in the array will be 'i'. Since i values in the array start from 0 onwards, and our counting will start from 1, we add 1 to i value found. So i+1 gives the position of the string in the array. Here, we are comparing 'search' with str[i], where i represents all the strings in the array one by one. This type of searching is called 'linear search' or 'sequential search'.

Program 7: Search for a given string in an array of strings. This is achieved by comparing the searching string with each of the array elements one by one in a loop.

```
//Searching for a string - Linear search
import java.io.*;
class Search
{
      public static void main(String args[ ]) throws IOException
      {
            //to accept data from keyboard
            BufferedReader br = new BufferedReader(new
            InputStreamReader(System.in));

            //ask how many strings
            System.out.print("How many strings? ");
            int n = Integer.parseInt(br.readLine());

            //create a string type array with size n
            String str[] = new String[n];

            //store strings into str[ ]
            for(int i=0; i<n; i++)
                  {
                        System.out.print("Enter a string: ");
                        str[i]= br.readLine();
                  }

                  //accept the string to search
                  System.out.print("Enter string to search: ");
                  String search = br.readLine();

                  //take a boolean var.
                  boolean found = false;

                  //search for the string in str[ ]
                  for(int i=0; i<n; i++)
                  {
                        if(search.equalsIgnoreCase(str[i]))
                              {
                                    System.out.println("Found at
                                    position: "+ (i+1));
                                    found = true; //string found
                              }
                  }

                  //if not found, display message
                  if(!found)
                  System.out.println("Not found in the group");

      }
}
```

Output:

```
C:\> javac Search.java
C:\> java Search
How many strings? 5
Enter a string: Hyderabad
Enter a string: New Delhi
Enter a string: Bangalore
```

```
Enter a string: HYDERABAD
Enter a string: Bhubaneswar
Enter string to search: Hyderabad
Found at position: 1
Found at position: 4
```

Conclusion

Strings are very important since they are most often used on a network while passing data from one system to another system. The data thus passed will be generally in the form of strings. Java's String class comes with necessary methods to work with strings. Since String class objects are immutable, they can be shared between different applications.