Chapter 14

STRINGS

In this chapter, you will learn about strings. Java supports two types of strings, String and StringBuffer. String type strings are rigid, whereas StringBuffer strings are flexible for manipulation. Methods to handle these two types of strings are dealt in this chapter.

In Java language, a string is defined as a sequence of characters. The strings used in Java are handled by two classes, **String** and **StringBuffer**. String class deals with string that are not altered after creation. **StringBuffer** class deals with strings that need alteration after they are created. Strings dealt by **String** class are more efficient to handle than strings dealt by **StringBuffer**. Both the classes are available in java.lang package.

14.1 The String Class

String objects are created using the following constructors:

String()
: Creates a String object with no character

String(char charray[])
: Creates a String using the charray

String(char charray[], int start, int len)
: Creates a String from charray, starting at charray[start] with len number of chars

String(String strObject)
Creates a String from String object strObject

Examples

1. String s1 = new String()
   Creates s1 with no characters

2. char name[ ] = { 'R', 'a', 'm', 'a', 'n' };
   String s2 = new String (name);
   Creates a String s2 with string "Raman"

3. char name[ ] = { 'R', 'a', 'm', 'a', 'n' };
   String s3 = new String (name, 2, 3)
   Creates a String s3 with string "man"

4. String s4 = new String (s2);
   Creates a String s4 with string "Raman"

It is also possible to use byte array as parameters in constructors used in examples 2 and 3 above to handle the 8-bit ASCII characters.

When constructors of **String** class are used to create objects, each time the new operator is called, an instance of the **String** is created. Each object takes up a memory space. Suppose, two objects contain identical strings, even then they occupy two separate memory locations. In the above examples 2 and 4, the objects s2 and s4, though contains identical strings, take up two separate memory locations. This can be avoided and memory can be saved if strings are created from String literals. Consider the following examples:

5. String s5 = "Raman";
6. String s6 = "Raman";

In this method, the string objects s5 and s6 contain identical strings "Raman". In this case Java does not store two objects, but only one. The same reference, where the object is stored, is given to s5 and s6. Fig. 14.1 illustrates this.

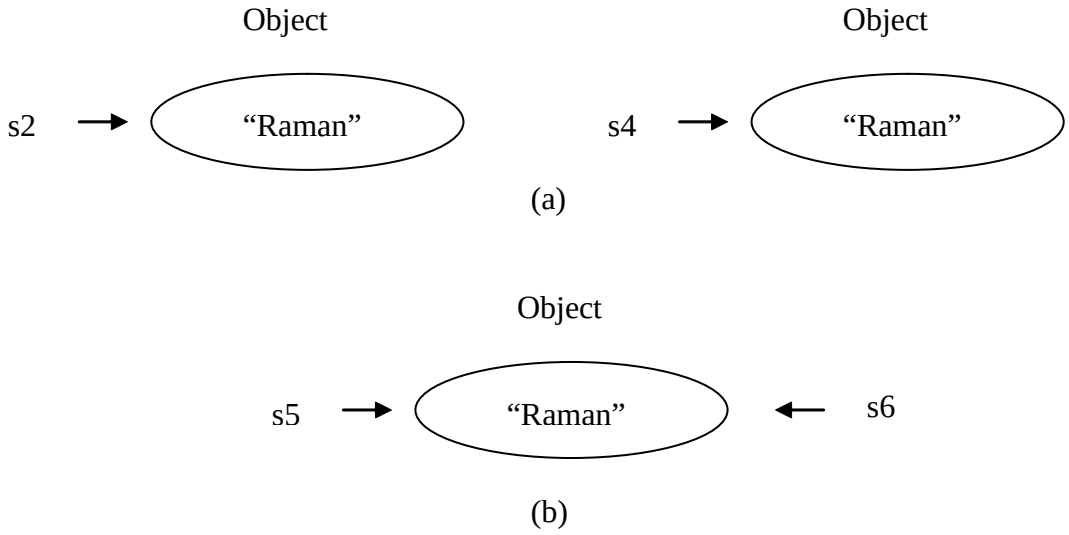

Fig.14.1(a) Separate memory is allocated when string objects are created using new operator even if strings are identical.
(b) Same memory is allocated when string objects are created using String literal with identical strings.

Therefore, using **String** literals to create **String** objects with identical strings is memory-efficient.

When an object reference variable, say s6, is copied to another String variable (say s7), a copy of the object reference is copied to the new variable (s7) and no object is created. Both variables s6 and s7 will refer to the same object.

14.1.1 Equality (= =) Operator and equals Method

We have already come across the use of the equality operator, = =, and equals method. Equality operator (==) checks whether the contents of object reference variables are equal, while the **equals** method checks whether the contents of the objects are equal.

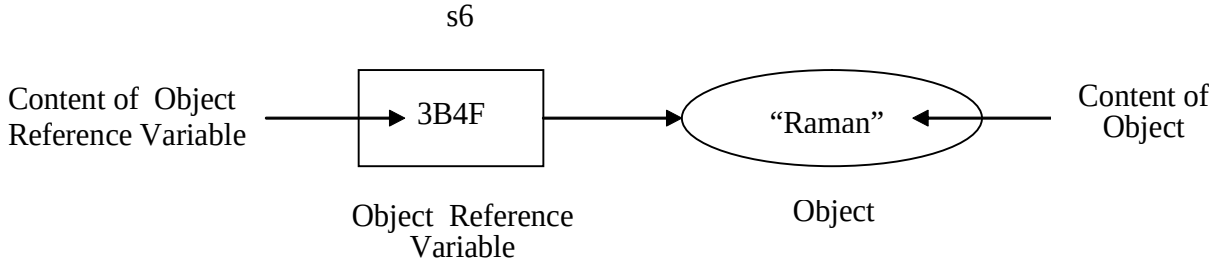

Fig.14.2 Object and Object Reference Variable

Referring to fig. 14.1 (a), it is obvious that:

(s2 ==s4) --> false
s2.equals(s4) --> true

Similarly, referring to fig. 14.1(b), the following result will be obtained:

(s5 == s6) --> true
s5.equals(s6) --> true

Creation of String and the use of == operator and **equals** method are illustrated in the program 14.1.

Program 14.1

```
/* This program demonstrates creation of strings using
constructor of String  and  String literal. The Strings
created are tested using  == operator  and equals method.
*/
class Stringcompare
    {
    public static void main(String args [])
        {
        String s1, s2, s3, s4, s5, s6, s7;
        char name [] ={'R','a','m','a','n'};
        s1 = new String();
        s2 = new String(name);
        s3 = new String(name, 2, 3);
        s4 = new String(s2);
        s5 = "Raman";
        s6 = "Raman";
        s7 = s2;
        System.out.println("\n s1 = " + s1);
        System.out.println("\n s2 = " + s2);
        System.out.println("\n s3 = " + s3);
        System.out.println("\n s4 = " + s4);
        System.out.println("\n s5 = " + s5);
        System.out.println("\n s6 = " + s6);
        System.out.println("\n s7 = " + s7);
        System.out.println("\n is( s2==s4) ? : " + (s2 ==
                           s4));
        System.out.println("\n is(s2.equals(s4)) ? : " +
                           s2.equals(s4));
        System.out.println("\n is(s5.equals(s4) ? : " +
                        s5.equals(s4));
        System.out.println("\n is(s5==s4) ? : " + (s5 ==
                           s4));
        System.out.println("\n is(s5.equals(s6)) ? : " +
                           s5.equals(s6));
        System.out.println("\n is(s5 == s6) ? : " + (s5 ==
                           s6));
```

```
        System.out.println("\n is(s2.equals(s7)) ? : " +
                                s2.equals(s7));
        System.out.println("\n is(s2 == s7) ? : " + (s2 ==
                                s7));
        }
    }
```

The above program gives the following output:

```
s1 =
s2 = Raman
s3 = man
s4 = Raman
s5 = Raman
s6 = Raman
s7 = Raman
is( s2==s4) ? : false
is(s2.equals(s4)) ? : true
is(s5.equals(s4) ? : true
   is(s5==s4) ? : false
is(s5.equals(s6)) ? : true
is(s5 == s6) ? : true
is(s2.equals(s7)) ? : true
is(s2 == s7) ? : true
```

14.1.2 String Concatenation With +

There are different ways of concatenation of strings. One way is using + operator. In Java, when + operator is used for strings, it concatenates the operand strings.

Examples:

1. String s1 = "this is demo" + "text";
   The String s1 will take up
   s1 = "This is a demo text".
2. int mark = 75;
   String s2 = "Your mark is " + mark;
   System.out.println(s2);

   The above will give :

   Your mark is 75

The + operator can be used with string and numerical variables. In such cases, the numerical values are converted to String type and appended to the preceding String object. Following are further examples for String concatenation:

3. String s1 = "Do it";
   String s2 = "by yourself";

```
String s3 = s1 + s2;
System.out.println(s3)
```

The above statements will give the following result:
Do it by yourself.

4. System.out.println ("Sum of 3 and 5 is " + 3+5):
This statement will give the following result:
Sum of 3 and 5 is 35

5. System.out.println ("Sum of 3 and 5 is" + (3+5));
This statement will give the following result:
Sum of 3 and 5 is 8

As it can be seen from the above examples 4 and 5, the operator precedence is to be observed while concatenating String with numerical types.

There are several useful methods for String objects and some of them are given in table 14.1.

Table 14.1 Methods in String Class

	Method	Purpose of the Method
	Character Extraction	
1.	char charAt(int index)	Returns the character at the index position of the invoking String object; the index is counted as 0 for the first character.
2.	void getChars(int sourceBegin, int sourceEnd, char target[ ], int targetBegin)	Copies characters from Object String starting at sourceBegin up to and inclusive of sourceEnd -1 characters into the target, starting at targetBegin
3.	byte[ ] getBytes()	Returns an array of characters as bytes from the String object
	String Comparison	
4.	boolean equals(Object str)	Returns true if str contains the same string as that in the invoking object, otherwise false
5.	boolean equalsIgnoreCase(String str)	Returns true if str contains the same string as in the invoking object by ignoring the case, otherwise false

6.	boolean regionMatches (int startIndex, String s2, int s2startIndex, int numchars)	Compares a region of the invoking object starting at startIndex with String s2 starting at s2startIndex for numchars; returns true if the region compared is equal, otherwise false
7.	boolean regionMatches (boolean ignorecase, int startIndex, String s2 int s2startIndex, int numchars)	Compares a region of the invoking object starting at startIndex with String s2 starting at s2startIndex for numchars; if ignoreCase is true, then the case of characters are ignored for comparison. Returns true if the region compared is equal, otherwise false
8.	boolean endsWith(String str)	Returns true if the invoking String object ends with str, otherwise false
9.	boolean startsWith(String str)	Returns true if the invoking String object starts with str, otherwise false
10.	int compareTo(String str)	Compares the invoking String object with str; returns a negative value if the String object is less than str, zero if both are equal and positive if String object is greater than str
	Searching Substrings	
11.	int indexOf(int ch)	Returns the index of the first occurrence of the character ch in the invoking String object
12.	int lastIndexOf(int ch)	Returns the index of the last occurrence of the character ch in the invoking String object
13.	int indexOf(String str)	Returns the index of the first occurrence of the string str in the invoking String object
14.	int lastIndexOf(Sring str)	Returns the index of the last occurrence of the string str in the invoking String object
15.	int indexOf(int ch, int startIndex)	Searches for the character ch starting at startIndex of the invoking String object till the end of the string; returns the index of the first occurrence of the character ch

16.	int lastIndexOf(int ch, int endIndex)	Searches for the character ch from the zero index till the endIndex of the invoking String object; returns the index of the last occurrence of the character ch.
17.	int indexOf(String str, int startIndex)	Searches for the substring str starting at startIndex till the end of the invoking String object; returns the index of the first occurrence of the string str
18.	int lastIndexOf(String str, int endIndex)	Searches for the substring str starting at 0 index till the endIndex of the invoking String Object; returns the index of the last occurrence of the string str
	String Modification	
19.	String substring(int startIndex)	Returns a substring starting at startIndex till the end of the invoking String object
20.	String substring(int startIndex, int endIndex)	Returns a substring starting at startIndex up to endIndex, but excluding the endIndex character of the invoking string object
21.	String concat(String str)	Returns a new String after appending the str to the invoking String object
22.	String replace(char existingChar, char newChar)	Returns a new String created by replacing existingChar with newChar
23.	String trim()	Returns a new String after removing the leading and trailing white spaces of the invoking String object
	Case Conversion	
24.	String toLowerCase()	Converts the uppercase characters of invoking String object to lowercase
25.	String toUpperCase()	Converts the lowercase characters of the invoking String object to uppercase
	Other Method	
26.	int length()	Returns the number of characters in the invoking String object

Program 14.2. shows methods used for character extraction.

Program 14.2

```
// This program illustrates the use of character extraction
// methods of String class
class Stringextract
    {
    public static void main(String args [])
      {
      char test [] = new char[10];
      String s1 = “This is a demo text”;
      s1.getChars(8, 14, test, 0);
      System.out.println(test);
      byte bytebuf [] = s1.getBytes();
      int count = bytebuf.length;
      System.out.println(); // new line
      for (int i = 0; i < count; i++)
            System.out.print((char)bytebuf[i]);
      }
    }
```

The above program gives the following output:

```
a demo
This is a demo text
```

Program 14.3 sorts the given strings in alphabetical order using **compareTo** method.

Program 14.3

```
/*This program illustrates the use of compareTo method
to sort the strings in alphabetical order.
*/
class Stringsort
    {
    public static void main(String args [])
      {
      String name [] =
            {
            “Zahir Khan”,
            “Raman”,
            “Magesh”,
            “Kumar”,
            “Sathish”
            };
      String temp;
```

```
    int count = name.length;
    System.out.println("\nThe given names\n");
    for (int i = 0; i < count; i++)
          System.out.println(name[i]);
    System.out.println("\nSorted names\n");
    for (int i = 0; i < count; i++)
          for (int j = i + 1; j < count; j++)
              if (name[i].compareTo(name[j]) > 0)
                  {
                  temp = name[i];
                  name[i] = name[j];
                  name[j] = temp;
                  }
    for (int i = 0; i < count; i++)
          System.out.println(name[i]);
    }
}
```

The above program gives the following output:

```
The given names

Zahir Khan
Raman
Magesh
Kumar
Sathish

Sorted names

Kumar
Magesh
Raman
Sathish
Zahir Khan
```

The toString() method

When an object is created using new operator, a reference to the object is created. A default **toString()** method in the super Object converts this reference to a human readable string form and stores it in the object reference. This string can be printed out using the **println** method. Program 14.4 shows the printing of the object reference of a class.

Program 14.4

```
// Program to illustrate the use of default  toString
// method in the super Object.
class Democlass
    {
     int x, y;
     Democlass(int a, int b)
```

```
        {
        x = a;
        y = b;
        }
    }
class DefaulttoString
    {
    public static void main(String args [])
       {
        Democlass dc = new Democlass(15, 45);
        System.out.println(dc);
       }
    }
```

The above program gives the following output:

```
Democlass@7a27c51c
```

This **toString()** method can be overridden by an user to get other messages needed about the object. Program 14.5 shows the overriding **toString()** method to generate the message about the initial values assigned to the instance variables.

Program 14.5

```
// Program to illustrate the use of overridden  toString
// method
class Democlass
    {
    int x, y;
    Democlass(int a, int b)
        {
        x = a;
        y = b;
        }
    public String toString()
        {
        return "Object created with x = " + x + " and  y = "
                                              + y;
        }
    }
class toStringDemo
    {
    public static void main(String args [])
       {
        Democlass dc = new Democlass(15, 45);
        System.out.println(dc);
       }
    }
```

The above program gives the following output:

```
Object created with x = 15 and  y = 45
```

14.2 The StringBuffer Class

Strings that need modification are handled by **StringBuffer** class. After creating a **StringBuffer**, new strings can be inserted or appended to it. The size of the **StringBuffer** can grow whenever needed. **StringBuffer** objects can be dynamically altered. When a **StringBuffer** is created, space for 16 more characters is always appended with it. This helps the **StringBuffer** object to grow by 16 more characters without any other process. That is, the size of the **StringBuffer** is the number of characters in that string plus 16. When the string grows beyond the free 16 character space, the **StringBuffer** is relocated to a new memory space with the required size. So, memory management for handling **StringBuffer** will not be as efficient as that of the fixed-length string objects.

The constructors in **StringBuffer** class help to create **StringBuffer** objects. The constructors are :

StringBuffer()
: Creates an empty StringBuffer Object; it has 16-character space.

StringBuffer(int size)
: Creates a StringBuffer object with a buffer of size capacity

StringBuffer(String str)
: Creates a StringBuffer object with the string str plus space for 16 more characters

The **equals()** method defined in **String** class is also available in **StringBuffer** class. The **equals()** method can be used to compare the same type of string objects and not to be mixed. That is, a **String** object cannot be compared with **StringBuffer** Object.

The equals() method should not be used to compare objects of different types.

Methods defined in **StringBuffer** class are given in table 14.2.

Table 14.2 Methods Defined in StringBuffer Class

	Method	Purpose of the Method
1.	int length()	Returns the total number of characters in the invoking object
2.	int capacity()	Returns the total allocated capacity for the invoking object
3.	void ensureCapacity (int capacity)	Sets the capacity of the buffer of the invoking object to the desired capacity
4.	void setLength(int len)	Sets the length of the StringBuffer object to len characters; if len is larger than the length of the object, empty spaces will be created. If len is less than the length of the object, characters after len will be lost.
5.	char charAt(int index)	Returns a character at the index position of the invoking object
6.	void setCharAt(int index, char ch)	Sets the character ch at the index position of the invoking object
7.	void getChars(int sourceBegin, int sourceEnd, char target[], int targetBegin)	Copies characters from object starting at sourceBegin up to and inclusive of sourceEnd -1 characters into the target, starting at targetBegin
8.	StringBuffer append(String str)	Returns a StringBuffer after appending str to the invoking object
9.	StringBuffer append(int num)	Returns a StringBuffer after appending the num to the invoking object
10.	StringBuffer append(object obj)	Returns a StringBuffer after appending obj to the invoking object
11.	StringBuffer insert(int index, String str)	Returns a StringBuffer after inserting str at index position of the invoking object
12.	StringBuffer reverse()	Returns a StringBuffer after reversing the characters in the string of the invoking object
13.	StringBuffer delete (int startIndex, int endIndex)	Returns a StringBuffer after deleting characters from startIndex to endIndex of the invoking object

14.	StringBuffer deleteCharAt (int index)	Returns a StringBuffer after deleting a character at the index position of the invoking object
15.	StringBuffer replace (int startIndex, int endIndex, String str)	Returns a StringBuffer after replacing the characters included in the range startIndex to endIndex of the invoking object by str
16.	String substring(int startIndex)	Returns a String starting at startIndex to the end of the invoking StringBuffer object
17.	String substring(int startIndex, int endIndex)	Returns a String included in the range startIndex to the endIndex of the invoking StringBuffer object
18.	String toString()	Returns a String after freezing the buffer size of the invoking StringBuffer object

Program 14.6 illustrates the use of some of the methods in **StringBuffer** class.

Program 14.6

```
// This program illustrates some of the methods in
// StringBuffer class.

class StringBuf
    {
    public static void main(String args [])
      {
      StringBuffer name = new
           StringBuffer("Somasundaram");
      int count = name.length();
      System.out.println("Name = " + name);
      System.out.println("Length of name = " + count);
      System.out.println("Capacity  of name = " +
           name.capacity());
      System.out.println("Substring of name from 5th
                character = " + name.substring(4));
      System.out.println("Name after inserting initial =
                          " + name.insert(0, "K. "));
      System.out.println("Name after appending the
                degree = " + name.append(" Ph.D"));
      System.out.println("Reversed name  = " +
           name.reverse());
      }
    }
```

The above program gives the following output:

```
Name = Somasundaram
Length of name = 12
Capacity  of name = 28
Substring of name from 5th character = sundaram
Name after inserting initial = K. Somasundaram
Name after appending the degree  = K. Somasundaram Ph.D
Reversed name  = D.hP maradnusamoS .K
```

After reading this chapter, you should have learned the following:

- Java supports two types of strings, String and StringBuffer.
- String type is rigid to manipulate, whereas the StringBuffer type is more flexible for manipulation.

In the next chapter, you will learn about threads.

Worked Out Problems-14

Problem 14.1w

Write a program to read the following list through keyboard entry, arrange the names in alphabetical order and print out.

Name	Sex
Ramasamy K.	m
Mahesh S.	m
Kumar S.	m
Charles B.	m
Parvathi M.	f
Cindrela U.	f
Ramesh K.	m
Gomathi S.	f
Balan N.	m

Program 14.1w

```
/*-------------------------------------------------------------
This program reads  the data from keyborad, arrange
the names in alphabetical order and prints out the list.

somasundaramk@yahoo.com
------------------------------------------------------------- */
```

```
import java.io.*;
class Prob141
    {
    public static void main(String args [])
        {
     String [] name = new String[50];
     String [] sex = new String[50];
     String tname, tsex;
     int pn = 50;
     String choice;
     int count = 0;
     try
          {
          // Read the keyboard to get the input
          InputStreamReader isr = new
                    InputStreamReader(System.in);
          BufferedReader br = new BufferedReader(isr);
          while (true)
               {
               System.out.println("Type the name");
               name[count] = br.readLine();
               System.out.print("Type the  sex (m/f) : ");
               sex[count] = br.readLine();
               System.out.print("Any more (y/n) : ");
               count++;
               choice = br.readLine();
               if (choice.equals("n"))
                        break;
               }
            }
        catch (IOException ie)
            {
            System.out.println("IO Error");
            }
     System.out.println("The data read from the keyboard\n");
     for (int i = 0; i < pn; i++)
            System.out.print("-");
     System.out.println("\n");
     System.out.println("Name\t\tSex");
     for (int i = 0; i < pn; i++)
            System.out.print("-");
     System.out.println("\n");
     for (int i = 0; i < count; i++)
            System.out.println(name[i] + "\t\t" + sex[i]);
     for (int i = 0; i < pn; i++)
            System.out.print("-");
     System.out.println("\n");
     //sort by alphabetical order
     for (int i = 0; i < count; i++)
            for (int j = i + 1; j < count; j++)
```

```
                {
                if (name[j].compareTo(name[i]) < 0)
                     {
                     tname = name[i];
                     name[i] = name[j];
                     name[j] = tname;
                     tsex = sex[i];
                     sex[i] = sex[j];
                     sex[j] = tsex;
                     }
                }
     System.out.println("Sorted List");
     for (int i = 0; i < pn; i++)
          System.out.print("-");

     System.out.println("\n");
     System.out.println("Name \t\tSex ");
     for (int i = 0; i < pn; i++)
            System.out.print("-");
     System.out.println("\n");
     for (int i = 0; i < count; i++)
            System.out.println(name[i] + "\t\t" + sex[i]);
     for (int i = 0; i < pn; i++)
            System.out.print("-");
     System.out.println("\n");
     }
  }
```

The above program when executed and data when fed through the key board gives the following output:

```
 The data read from the keyboard
---------------------------------------------------
Name                Sex
---------------------------------------------------

Ramasamy K          m
Magesh S            m
Kumar S             m
Charles B           m
Parvathi M          f
Cindrela U          f
Ramesh              m
Gomathi S           f
Balan N             m
---------------------------------------------------
```

Sorted List

Name	Sex
Balan N	m
Charles B	m
Cindrela U	f
Gomathi S	f
Kumar S	m
Magesh S	m
Parvathi M	f
Ramasamy K	m
Ramesh	m

Problem 14.2w

Write a program to count the number of characters, words and lines in a given file. The file name is to be given through the command line argument.

Program 14.2w

```
/*-------------------------------------------------------------
This program reads  a file and counts the number of characters,
no of words, no of lines. The file name is fed through the
command line argument.

somasundaramk@yahoo.com
------------------------------------------------------------- */

import java.io.*;
class Prob142
    {
    public static void main(String args [])
       {
        int lines = 0;

        int words = 0;
        int chars = 0;
        if (args.length == 0)
          {
          System.out.println("Give a file name in the
                              command line");
          System.exit(0);
          }
        File f = new File(args[0]);
        try
            {
            int chr;
```

```
            FileInputStream fins = new FileInputStream(f);
           InputStreamReader insr = new
                              InputStreamReader(fins);
           while ((chr = insr.read()) != -1)
               {
               chars++;
               switch ((char)chr)
                   {
                   case '\t':
                   case ' ':
                       words++;
                       break;
                   case '\n':
                       //case '\r':
                       words++;
                       lines++;
                   }
               }
           }
      catch (IOException ioe)
           {
           System.out.println("IO Problem");
           }
      System.out.println(args[0] + " file contains\n" +
              chars + " chars\n" + words + " words\n " +
              lines + " lines");
       }
   }
```

The above program gives the following output:

```
Prob142.java file contains
1063 chars
120 words
 42 lines
```

Problem 14.3w

Write a program to read a text file, “story.txt”, and replace all “his” words with another word “her” in it.

Program 14.3w

```
/*-------------------------------------------------------------
This program reads  a file, finds the word his and replaces it
with her.

somasundaramk@yahoo.com
------------------------------------------------------------- */
```

```
import java.io.*;
class Prob143
    {
    public static void main(String args [])
     {
     String find = "his";
     String replace = "her";
     String ureplace = "Her";
     String word = "";
     char chr;
     int rcount = 0;
     int c;
     File f = new File("story.txt");
     try
            {
             FileInputStream fins = new FileInputStream(f);
             InputStreamReader insr = new
                                  InputStreamReader(fins);
             System.out.println("The given text\n");
             while ((c = insr.read()) != -1)
                    System.out.print((char)c);
             insr.close();
            }
        catch (IOException ioe)
            {
            System.out.println("IO Problem");
            }
        System.out.println("\n");
         System.out.println("Replaced text\n");
        try
            {
            FileInputStream fins = new FileInputStream(f);
            InputStreamReader insr = new
                              InputStreamReader(fins);
            while ((c = insr.read()) != -1)
                {
                chr = (char)c;
                switch (chr)
                    {
                    case '\t':
                    case '\n':
                    case ' ':
                       word = word + chr;
                        if(word.trim()).equalsIgnoreCase(find))
                            {
                            rcount++;
                       if Character.isUpperCase(word.charAt(0)))
                               System.out.print(ureplace + " ");
                            else
                                System.out.print(replace + " ");
```

```
                    }
                else
                    System.out.print(word);
                word = "";
                break;
            default:
                word = word + chr;
            }
        }
      System.out.print(word);
      System.out.println("\n");
     if (rcount > 0)
          System.out.println(rcount + " words are
                                  replaced");
      else
          System.out.println(" No  word is replaced");
      insr.close();
      }
    catch (IOException ioe)
      {
      System.out.println("IO Problem");
      }
    }
}
```

The above program gives the following output:

The given text

Once upon a time, there lived a king called Cholan. He used to go around the country to know whether his citizens are well. Whenever he went around, he always chose to go by walk. This made it possible for all the people of his country to meet him easily. His services were appreciated by his citizens.

Replaced text

Once upon a time, there lived a king called Cholan. He used to go around the country to know whether her citizens are well. Whenever he went around, he always chose to go by walk. This made it possible for all the people of her country to meet him easily. Her services were appreciated by her citizens.

4 words are replaced

Exercise-14

I. Fill in the blanks

14.1. The string that does not need any modification is to be dealt by _____ class.
14.2. The string that needs modification while processing is dealt by _____ class.
14.3. The classes for handling string are available in the _____ package.
14.4. When a String object is copied into another String object reference, a copy of the original object _____ created.
14.5. To compare the contents of two objects, _____ method is used.
14.6. The number of extra character width available in a StringBuffer is _____ .
14.7. The length of the StringBuffer is defined as _____.
14.8. The capacity of the StringBuffer is given by _____ .
14.9. When toString method is applied to a StringBuffer, the buffer size is ____ and the StringBuffer is converted to _____ .

II. Write Java programs for the following:

14.10. Write a program to read a line of text from the console. Find the position of the first and last occurrence of the string "the". Copy all the characters enclosed between the two positions to another String and print it out.

14.11. Write a program to read a line of text from the console. Print out only the vowels (a, e, i, o, u) and their position of occurrence.

14.12. A set of 10 names is given. Write a program to delete the first three characters of the names and arrange the resulting names in alphabetical order and print them out.

14.13. Read in a line of text from the keyboard. Adjust the white space between words so that the whole line is aligned left and right in a line width of 60 characters and print it out.

14.14. A set of 5 words is given. Write a program to reverse each word and arrange the resulting words in alphabetical order.

14.15. Write a program to read a line of text from the console. Change the first character of each word to uppercase letter and print out the resulting string.

* * * * * *