UNIT 3 STRINGS AND CHARACTERS

Structure **Page Nos.**

3.0 INTRODUCTION

In programming we use several data types as per our needs. If you observe throughout your problem solving study next to numeric data types character and strings are the most important data type that are used. In many of the programming languages strings are stored in arrays of characters (*e.g.* C, C++, Pascal, ...). However, in Java strings are a separate object type, called String. You are already using objects of String type in your programming exercises of previous units. A string is a set of character, such as "Hi". The Java platform contains three classes that you can use when working with character data: Character, String and StringBuffer. Java implements strings as object of String class when no change in the string is needed, and objects of StringBuffer class are used when there is need of manipulating of in the contents of string. In this unit we will discuss about different constructors, operations like concatenation of strings, comparison of strings, insertion in a string etc. You will also study about character extraction from a string, searching in a string, and conversion of different types of data into string form.

3.1 OBJECTIVES

After going through this unit you will be able to:

- explain Fundamentals of Characters and Strings;
- use different String and StringBuffer constructors;
- apply special string operations;
- extract characters from a string;
- perform such string searching & comparison of strings;
- data conversion using valueOf ( ) methods, and
- use StringBuffer class and its methods.

3.2 FUNDAMENTALS OF CHARACTERS AND STRINGS

Java provides three classes to deal with characters and strings. These are:

Character: Object of this class can hold a single character value. This class also defines some methods that can manipulate or check single-character data.

String: Objects of this class are used to deal with the strings, which are unchanging during processing.

String Buffer: Objects of this class are used for storing those strings which are expected to be manipulated during processing.

Characters

An object of Character type can contain only a single character value. You use a Character object instead of a primitive char variable when an object is required. For example, when passing a character value into a method that changes the value or when placing a character value into a Java defined data structure, which requires object. Vector is one of data structure that requires objects.

Now let us take an example program to see how Character objects are created and used.

In this program some character objects are created and it displays some information about them.

```
//Program
public class CharacterObj
{
public static void main(String args[])
{
Character Cob1 = new Character('a');
Character Cob2= new Character('b');
Character Cob3 = new Character('c');
int difference = Cob1.compareTo(Cob2);
if (difference == 0)
System.out.println(Cob1+"   is equal to  "+Cob2);
else
System.out.println(Cob1+"  is not equal to  "+Cob2);
System.out.println("Cob1 is " + ((Cob2.equals(Cob3)) ? "equal" : "not equal")+ " to
Cob3.");
}
}
```

Output:
a is not equal to b
Cob1 is not equal to Cob3.

In the above CharacterObj program following constructors and methods provided by the Character class are used.

Character (char): This is the Character class only constructor. It is used to create a Character object containing the value provided by the argument. But once a Character object has been created, the value it contains cannot be changed.

Compare to (Character): This instance method is used to compare the values held by two character objects. The value of the object on which the method is called and the value of the object passed as argument to the method.

For example, the statement

int difference = Cob1.compareTo(Cob2), in the above program.

This method returns an integer indicating whether the value in the current object is greater than, equal to, or less than the value held by the argument.

Equals (Character): This instance method is used to compare the value held by the current object with the value held by another (This method returns true if the values held by both objects are equal) object passed as on argument.

For example, in the statement

Cob2.equals(Cob3)
value held by object **Cob2** and the value held by **Cob3** will be compared.

In addition to the methods used in program CharacterObj, methods given below are also available in Character class.

To_String (): This instance method is used to convert the object to a string. The resulting string will have one character in it and contains the value held by the character object.

Char_Value(): An instance method that returns the value held by the character object as a primitive char value.

IsUpperCase (char): This method is used to determine whether a primitive char value is uppercase or not. It returns true if character is in upper case.

String and StringBuffer Classes

Java provides two classes for handling string values, which are **String** and

Can you tell why two String Classes?

The String class is provided for strings whose value will not change. For example, in program you write a method that requires string data and it is not going to modify the string.

The StringBuffer class is used for strings that will be modified. String buffers are generally used for constructing character data dynamically, for example, when you need to store information, which may change.

Content of strings do not change that is why they are more efficient to use than string buffers. So it is good to use String class wherever you need **fixed-length** objects that are **immutable (**size cannot be altered).

For example, the string contained by myString object given below cannot be changed.
String myString = “This content cannot be changed!”.

☞ Check Your Progress 1

1) Write a program to find the case (upper or lower) of a character.

……

……

2) Explain the use of equal () method with the help of code statements.

……

……

……

3) When should StringBuffer object be preferred over String object?

……

……

……

……

3.3 THE STRING CLASS

Now let us discuss String class in detail. There are many constructors provided in Java to create string objects. Some of them are given below.

public String(): Used to create a String object that represents an empty character sequence.

public String(String value): Used to create a String object that represents the same sequence of characters as the argument; in other words, the newly created string is a copy of the string passed as an argument.

public String(char value[]): New object of string is created using the sequence of characters currently contained in the character array argument.

public String(char value[], int offset, int count): A new string object created contains characters from a sub-array of the character array argument.

public String(byte bytes[], String enc) throws Unsupported Encoding Exception: Used to construct a new String by converting the specified array of bytes using the specified character encoding.

public String(byte bytes[], int offset, int length): Used to create an object of String by converting the specified sub-array of bytes using the platform's default character encoding.

public String(byte bytes[]): Used to create an object of String by converting the specified array of bytes using the platform's default character encoding.

public String(StringBuffer buffer) : Used to create an object of String by using existing StringBuffer object which is passed as argument.

String class provides some important methods for examining individual characters of the strings, for comparing strings, for searching strings, for extracting sub-strings, and for creating a copy of a string with all characters translated to uppercase or to lowercase.

The Java language also provides special support for the string concatenation operator (+), and for conversion of other objects to strings.

Note: When you create a String object, and, if it has the same **value** as another object, Java will point both object references to the same memory location.

3.4 STRING OPERATIONS

String class provides methods for handling of String objects. String class methods can be grouped in index methods, value Of () methods and sub-string methods. Methods of these groups are discussed below:

Index Methods

Methods	Return
public int length()	Return the length of string.
public int indexOf(int ch)	Return location of first occurrence of ch in string, if ch don't exist in string return –1.
public int indexOf(int ch, int fromIndex)	Return location of occurrence of ch in string after fromIndex, if ch don't exist in string return –1.
public int lastIndexOf(int ch)	Return location of last occurrence of ch in string, if ch location does not exist in string return –1.
public int lastIndexOf(int ch, int fromIndex)	Return last of occurrence of ch in string after location fromIndex, if ch does not exist in string after location fromIndex return –1.
public int indexOf(String str)	Return location of first occurrence of substring str in string, if str does not exist in string return –1.
public int indexOf(String str, int fromIndex)	Return location of first occurrence of substring str in after location fromIndex string, if str does not exist in string return –1.
public int lastIndexOf(String str)	Return location of last occurrence of substring str in string , if str does not exist in string return –1.
public int lastIndexOf(String str, int fromIndex)	Return location of last occurrence of substring str in after location from Index string, if str does not exist in string return –1.

You can see in the example program given below, in which the index methods are used. This program will help you to know the use of index methods.

```
public class Index_Methods_Demo
{
public static void main(String[] args)
{
String  str =  "This is a test string";
System.out.println(" The length of str is :"+ str.length());
System.out.println("Index of 't' in str is:"+str.indexOf('t'));
```

```
// Print first occurrence of character t in string
System.out.println("First occurrence of 't' after  13 characters in the str:"+
str.indexOf('t',12));
// Print first occurrence of character t in string after first 13 characters
System.out.println("Last occurrence of 'z' in the str:"+ str.lastIndexOf('z'));
// Print Last  occurrence of character z in string : See output and tell
// what is printed because 'z' is not a character in str.
System.out.println("First occurrence of substring :is substring of string  str:"+
str.indexOf("is"));
// Print first occurrence of substring "is" in str
System.out.println("Last occurrence of substring :ing in the str:"+
str.lastIndexOf("ing"));
// Print first occurrence of substring "ing" in string str
System.out.println("First occurrence of substring :this after  11 characters in the str:"+
str.indexOf("this",10));
// Print first occurrence of substring "this" after first 11 characters in string str
}
}
}
```

Output:

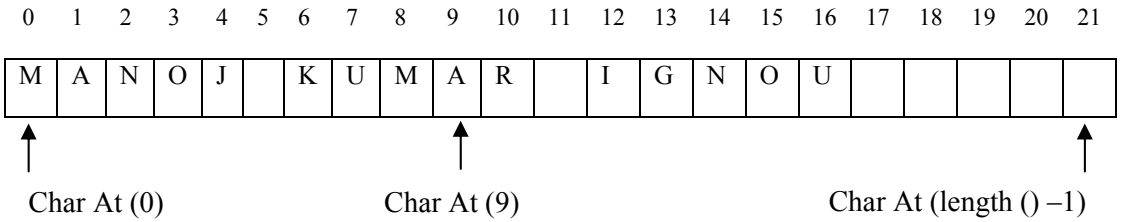
```
The length of str is :21
Index of 't' in str is:10
First occurrence of 't' after  13 characters in the str:13
Last occurrence of 'z' in the str:-1
First occurrence of substring :is substring of string  str:2
Last occurrence of substring :ing in the str:18
First occurrence of substring :this after  11 characters in the str:-1
```

In the output of the above given program you can see that if a character or substring does not exist in the given string then methods are returning: –1.

Now let us see some substring methods, comparisons methods, and string modifying methods provided in string class. These methods are used to find location of a character in string, get substrings, concatenation, comparison of strings, string reasons matching, case conversion etc.

Substring Methods

public char charAt(int index)	Return character at position i in string.



Methods	Returns
public byte[] getBytes()	Return byte array of string characters
public String substring(int beginIndex)	Return substring from index beginIndex to the end of string
public String substring	Return substring from index

(int beginIndex, int endIndex)	beginIndex to the endIndIndex of string
public String concat(String str)	Return a string which have sting on which this method is called Concatenated with str
public char[] toCharArray()	Return character array of string

Comparisons Methods

public boolean equals(Object str)	Return true if strings contain the same characters in the same order.
public boolean equalsIgnoreCase(String aString)	Return true if both the strings, - contain the same characters in the same order, and ignore case in comparison.
public int compareTo(String aString)	Compares to aString returns <0 if a String< sourceString 0 if both are equal and return >0 if aString>sourceString ; this method is case sensitive and used for knowing whether two strings are identical or not.
public boolean regionMatches (int toffset, String other, int offset, int len)	Return true if both the string have exactly same symbols in the given region.
public boolean startsWith (String prefix, int toffset)	Return true if prefix occurs at index toffset.
public boolean startsWith(String prefix)	Return true if string start with prefix.
public boolean endsWith(String suffix)	Return true if string ends with suffix.

Methods for Modifying Strings

public String replace(char oldChar, char newChar)	Return a new string with all oldChar replaced by newChar
public String toLowerCase()	Return anew string with all characters in lowercase
public String toUpperCase()	Return anew string with all characters in uppercase.
public String trim()	Return a new string with whitespace deleted from front and back of the string

In the program given below some of the string methods are used. This program will give you the basic idea about using of substring methods, comparisons methods, and string modifying methods.

```
public class StringResult
{
public static void main(String[] args)
{
String original = " Develop good software   ";
String sub1 = "";
String  sub2 = "Hi";
int index = original.indexOf('s');
sub1 =original.substring(index);
System.out.println("Substring sub1 contain: " + sub1);
System.out.println("Original contain: " + original+".");
original= original.trim();
System.out.println("After trim original contain: " + original+".");
System.out.println("Original contain: " + original.toUpperCase());
if (sub2.startsWith("H"))
System.out.println("Start with H: Yes");
else
System.out.println("Start with H: No");
sub2 = sub2.concat(sub1);
System.out.println("sub2 contents after concatenating sub1:"+sub2);
System.out.println("sub2 and  sub1 are equal:"+sub2.equals(sub1));
}
}
```

Output:

```
---------- Run ----------
Substring sub1 contain: software
Original contain:  Develop good software.
After trim original contain: Develop good software.
Original contain: DEVELOP GOOD SOFTWARE
Start with H: Yes
sub2 contents after concatenating sub1:Hisoftware
sub2 and  sub1 are equal:false
```

Now let us see valueOf() methods, these methods are used to convert different type of values like integer, float double etc. into string form. ValueOf () method is overloaded for all simple types and for Object type too. ValueOf() methods returns a string equivalent to the value which is passed in it as argument.

3.5 DATA CONVERSION USING VALUE OF( ) METHODS

public static String valueOf(boolean b)	Return string representation of the boolean argument
public static String valueOf(char c)	Return string representation of the character argument
public static String valueOf(int i)	Return string representation of the integer argument
public static String valueOf(long l)	Return string representation of the long argument
public static String valueOf(float f)	Return string representation of the float argument

public static String valueOf(double d)	Return string representation of the double argument
public static String valueOf(char data[])	Return string representation of the character array argument
public static String valueOf(char data[], offset, int count)	Return string representation of a int specific sub array of the character array argument

In the program given below you can see how an integer data is converted into a string type. This program also uses one very important operator '+' used for concatenating two strings.

```
class String_Test
{
public static void main(String[] args)
{
String s1 = new String("Your age is: ");
String s2 = new String();
int age = 28;
s2 = String.valueOf(age);
s1 = s1+s2+ " years";
System.out.println(s1);
}
}
```

Output:
Your age is: 28 years

☞ Check Your Progress 2

1) Write a program to find the length of string "Practice in programming is always good". Find the difference between first and last occurrence of 'r' in this string.

……

……

……

2) Write a program which replaces all the occurrence of 'o' in string "Good Morning" with 'a' and print the resultant string.

……

……

……

3) Write a program, which takes full path (directory path and file name) of a file and display Extension, Filename and Path separately for example, for input"/home/mem/index.html", output is Extension = html
Filename = index Path = /home/mem

……

……

……

……

You have seen that String class objects are of fixed length and no modification can be done in the content of strings except replacement of characters. If there is need of strings, which can be modified, then StrinBuffer class object should be used. Now let us discuss about StringBuffer class in detail.

3.6 STRINGBUFFER CLASS AND METHODS

StringBuffer: It is a peer class of String that provides much of the functionality of strings. In contrast to String objects, a StringBuffer object represents growable and writeable character sequences. In the strings created by using StringBuffer you may insert characters and sub-strings in the middle or append at the end. Three constructors of StringBuffer class are given below:

StringBuffer() Constructs a string buffer with no characters in it and an initial capacity of 16 characters.

StringBuffer (int size) constructs a string buffer with no characters in it and an initial capacity specified by the length argument.

StringBuffer (String str) Constructs a string buffer so that it represents the same sequence of characters as the string argument; in other words, the initial contents of the string buffer is a copy of the argument string.

Methods of String Buffer Class

In addition to functionality of String class methods, SrtingBuffer also provide methods for modifying strings by inserting substring, deleting some content of string, appending string and altering string size dynamically. Some of the key methods in StringBuffer are mentioned below:

public int **length**() Returns the length (number of characters)of the string buffer on which it is called.

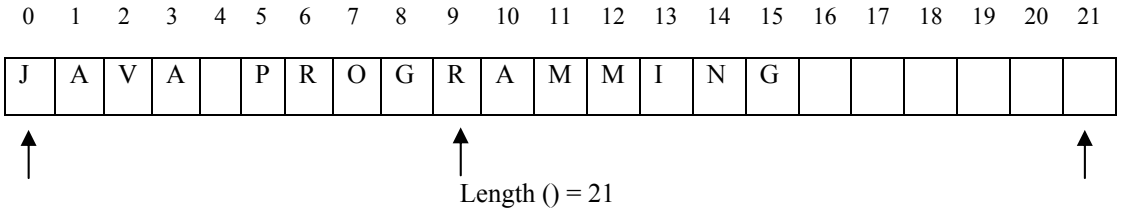


Figure 1: StringBuffer

public int **capacity**() — Returns the total allocated capacity of the String buffer. The capacity is the amount of storage available for characters can be inserted.

public void **ensureCapacity** (int minimumCapacity) — Used to ensures the capacity of the buffer is at least equal to the specified minimum capacity. If the current capacity of the string buffer on which this method is called is less than the argument value, then a new internal buffer is allocated with greater capacity and the new capacity is the larger than

i) The minimumCapacity argument.

ii) Twice the old capacity, plus 2.

	If you pass the minimumCapacity argument a nonpositive, value, then this method takes no action and simply returns.
public void **setLength**(int newLength)	Used to set the length of the string buffer on which it is called, if new length is less than the current length of the string buffer, the string buffer is truncated to contain exactly the number of characters given by the newLength argument. If the newLength argument is greater than or equal to the current length, string buffer is appended with sufficient null characters so that length becomes equal to the new Length argument. Obviously the newLength argument must be greater than or equal to 0.
public void **setCharAt**(int index, char ch)	Set character ch at the position specified by index in the string buffer on which it is called. This alters the character sequence that is identical to the old character sequence, except that now string buffer contain the character ch at position index and existing characters at index and after that in the string buffer are shifted right by one position. The value of index argument must be greater than or equal to 0, and less than the length of this string buffer.
public StringBuffer **append**(String newStr)	Appends the newStr to the string buffer. The characters of the string newStr are appended, to the contents of this string buffer increasing the length of the string buffer on which this method is called by the length of the newStr. Public.
StringBuffer **append** (StringBuffer strbuf)	Appends the characters of the strbuf to the string buffer, this results in increasing the length of the StringBuffer object on which this method is called. The method ensureCapacity is first called on this StringBuffer with the new buffer length as its argument (This ensures that the storage of this StringBuffer is adequate to contain the additional characters being appended).
public StringBuffer **append**(int i)	Appends the string representation of the int argument to the string buffer

	on which it is called. The argument is converted to a string by using method String.valueOf, and then the characters of converted string are then appended to the string buffer.
public StringBuffer **insert** (int offset, String str)	Insert the string str into the string buffer on which it is called at the indicated offset. It moves up characters originally after the offset position are shifted. Thus it increase the length of this string buffer by the length of the argument. The offset argument must be greater than or equal to 0, and less than or equal to the length of this string buffer.
public StringBuffer **insert** (int offset,char[] str)	Inserts the string representation of the array argument character into the string buffer on which this method is called. Insertion is made in the string buffer at the position indicated by offset. The length of this string buffer increases by the length of the argument
public StringBuffer **insert** ch (int offset, char ch)	Inserts the string representation of into the string buffer at the position indicated by offset. The length of this string buffer increases one.
public StringBuffer **reverse**()	Return the reverse of the sequence of the character sequence contained in the string buffer on which it is called. If n is the length of the old character sequence, in the string buffer just prior to execution of the reverse method. Then the character at index k in the new character sequence is equal to the character at index n-k-1 in the old character sequence.
public StringBuffer **delete**(int start, int end)	This method is used to remove total end–start +1 characters starting from location start and up to including index end.
public StringBuffer **deleteCharAt** (int index)	Removes the character at the specified position in the string buffer on which this method is called.
public StringBuffer **replace** (int start, int end, String str)	Replaces the characters of the string buffer from start to end by the characters in string str.

Now let us see use of some of the methods like capacity, append etc. of StringBuffer. As you know capacity method differs from length method, in that it returns the amount of space currently allocated for the StringBuffer, rather than the amount of

space used. For example, in the program given below you can see that capacity of the StringBuffer in the reverseMe method never changes, while the length of the StringBuffer increases by one in each of the iteration of loop.

```
class ReversingString
  {
        public String reverseMe(String source)
        {
       int i, len = source.length();
       StringBuffer dest = new StringBuffer(len);
        System.out.println("Length of dest:"+dest.length());
        System.out.println("Capacity of dest:"+dest.capacity());
       for (i = (len - 1); i >= 0; i--)
       dest.append(source.charAt(i));
        System.out.println("Capacity of dest:"+dest.capacity());
        System.out.println("Length of dest:"+dest.length());
       return dest.toString();
      }
      }
        public class ReverseString
        {
        public static void main ( String args[])
        {
        ReversingString R = new RString();
        String myName = new String("Mangala");
        System.out.println("Length of myName:"+myName.length());
           System.out.println(" myName:"+myName);
        System.out.println("reverseMe call:"+R.reverseMe(myName));
        }
```

Output:

```
Length of myName:7
myName:Mangala
Length of dest:0
Capacity of dest:7
Capacity of dest:7
Length of dest:7
Reverse call:alagnaM
```

Note:

i. In the reverseMe () method of above StringBuffer's to String () method is used to convert the StringBuffer to a String object before returning the String.

ii. You should initialize a StringBuffer's capacity to a reasonable first guess, because it minimizes the number of times memory to be allocated for the situation when the appended character causes the size of the StringBuffer to grow beyond its current capacity. Because memory allocation is a relatively expensive operation, you can make your code more efficient by initializing a StringBuffer's capacity to a reasonable first guess size.

Now let us take one more example program to see how to insert data into the middle of a StringBuffer. This is done with the help of one of StringBufffer's insert methods.

```
class  InsertTest
{
public static void main ( String args[])
{
StringBuffer MyBuffer = new StringBuffer("I got class in B.Sc.");
MyBuffer.insert(6, "First ");
System.out.println(MyBuffer.toString());
}
```

```
}
```
Output:
I got First class in B.Sc.

☞ Check Your Progress 3

1) List any two operations that you can perform on object StringBuffer but cannot perform on object of String class.

……

……

……

……

……

2) Write a program to answer the following:

   i. Find the initial capacity and length of the following string buffer:
   StringBuffer StrB = new StringBuffer("Object Oriented Programming is possible in Java");

   ii. Consider the following string:

   String Hi = "This morning is very good";
   What is the value displayed by the expression Hi.capacity()?
   What is the value displayed by the expression Hi.length()?
   What is the value returned by the method call Hi.charAt(10)?

……

……

……

……

……

3) Write a program that finds initials from your full name and displays them.

……

……

……

……

……

3.7 SUMMARY

Character and strings are the most important data type. Java provides Character, String, and StringBuffer classes for handling characters, and strings. String class is used for creating the objects which are not to be changed. The string objects for which contents are to be changed StringBuffer class, is used. Character class provides various methods like compareTo, equals, isUpperCase.String class provides methods for index operations, substring operations, and a very special group of valueOf methods. Comparison methods are also provided in String class.String class methods

like replace, toLowerCase, trim are available for minor modifications though, in general string classes is not used for dynamic Strings. StringBuffer objects allow to insert character or substring in the middle or append it to the end. StringBuffer also allows deletion of a substring of a string.

3.8 SOLUTIONS/ANSWERS

Check Your Progress 1

1)
```
//Program to test whether the content of a Character object is Upper or Lower
public class CharTest
{
public static void main(String args[])
{
Character MyChar1 = new Character('i');
Character MyChar2 = new Character('J');
//Test for MyChar1
if (MyChar1.isUpperCase(MyChar1.charValue()))
System.out.println("MyChar1 is in Upper Case: "+MyChar1);
else
System.out.println("MyChar1 is in Lower Case: "+MyChar1);
// Test for MyChar2
if (MyChar2.isUpperCase(MyChar2.charValue()))
System.out.println("MyChar2 is in Upper Case: "+MyChar2);
else
System.out.println("MyChar2 is in Lower Case: "+MyChar2);

}
}
```
Output:
MyChar1 is in Lower Case: i
MyChar2 is in Upper Case: J

2) This instance method is used to compare the value held by the current object with the value held by another object. For example let Ch1 and Ch2 be two objects of Character class then
Ch1.equals(Ch2);
method returns true if the values held by both objects are equal, otherwise false.

3) StringBuffer object should be given preference over String objects if there is a need of modification (change may take place) in the content. These include flexibility of increase in size of object.

Check Your Progress 2

1)
```
class  StringLen
{
public static void main(String[] args)
{
String MyStr = new String(" Practice in programming is always good");
System.out.println("The length of MyStr is :"+MyStr.length());
int i = MyStr.lastIndexOf("r")- MyStr.indexOf("r");
System.out.println("Difference between first and last occurence of 'r' in MyStr
is :"+ i);

}
```

```
}
```

Output:
The length of MyStr is: 39
Difference between first and last occurence of 'r' in MyStr is: 15

2)
```
//program
class ReplaceStr
{
public static void main(String[] args)
{
String S1 = new String("Good Morning");
System.out.println("The old String is:"+S1);
String S2= S1.replace('o', 'a');
System.out.println("The new String after rplacement of 'o' with 'a' is:"+S2);
}
}
```

Output:
The old String is:Good Morning
The new String after replacement of 'o' with 'a' is: Good Marning

3)
```
// It is assumed that  fullPath has a directory path, filename, and extension.
class Filename
{
  private String fullPath;
  private char pathSeparator, extensionSeparator;
  public Filename(String str, char separ, char ext)
  {
     fullPath = str;
     pathSeparator = separ;
     extensionSeparator = ext;
  }
  public String extension()
  {
     int dot = fullPath.lastIndexOf(extensionSeparator);
     return fullPath.substring(dot + 1);
   }
  public String filename()
  {
     int dot = fullPath.lastIndexOf(extensionSeparator);
     int separ = fullPath.lastIndexOf(pathSeparator);
     return fullPath.substring(separ + 1, dot);
  }
  public String path()
  {
     int separ = fullPath.lastIndexOf(pathSeparator);
     return fullPath.substring(0, separ);
  }
}
// Main method  is in FilenameDemo class
  public class FilenameDemo1
 {
  public static void main(String[] args)
  {
     Filename myHomePage = new Filename("/HomeDir/MyDir/MyFile.txt",'/', '.');
     System.out.println("Extension = " + myHomePage.extension());
     System.out.println("Filename = " + myHomePage.filename());
     System.out.println("Path = " + myHomePage.path());
```

```
    }
}
```

Output:

```
Extension = txt
Filename = MyFile
Path = /HomeDir/MyDir
```

Note:

In this program you can notice that extension uses dot + 1 as the argument to substring. If the period character is the last character of the string, then dot + 1 is equal to the length of the string which is one larger than the largest index into the string (because indices start at 0). However, substring accepts an index equal to but not greater than the length of the string and interprets it to mean "the end of the string."

Check Your Progress 3

1) i. Insertion of a substring in the middle of a string.

   ii. Reverse the content of a string object.

2)
```
class StrCap
{
public static void main(String[] args)
{
StringBuffer StrB = new StringBuffer("Object Oriented Programming is
possible in Java");
String  Hi = new  String("This morning is very good");
System.out.println("Initial Capacity of StrB is :"+StrB.capacity());
System.out.println("Initial length of StrB is :"+StrB.length());

//System.out.println("value displayed by the expression Hi.capacity() is:
"+Hi.capacity());
System.out.println("value displayed by the expression Hi.length() is:
"+Hi.length());
System.out.println("value displayed by the expression Hi.charAt() is:
"+Hi.charAt(10));
}
}
```

Output:

```
Initial Capacity of StrB is :63
Initial length of StrB is :47
value displayed by the expression Hi.length() is: 25
value displayed by the expression Hi.charAt() is: n
```

Note: The statement "System.out.println("value displayed by the expression Hi.capacity() is: "+Hi.capacity());" is commented for successful execution of program because capacity method is mot available in String class.

3)
```
public class NameInitials
{
public static void main(String[] args)
{
String myNameIs = "Mangala Prasad Mishra";
StringBuffer myNameInitials = new StringBuffer();
System.out.println("The name is : "+myNameIs);
// Find length of name given
int len = myNameIs.length();
```

```
        for (int i = 0; i < len; i++)
                                {
        if (Character.isUpperCase(myNameIs.charAt(i)))
                                    {
        myNameInitials.append(myNameIs.charAt(i));
       }
       }
       System.out.println("Initials of the name: " + myNameInitials);
       }
       }
```

Output:

The name is: Mangala Prasad Mishra
Initials of the name: MPM