

---

## UNIT 10 NUMBER, CHARACTER AND STRINGS

---

### Structure

### Page No

10.0 Introduction	
10.1 Objectives	
10.2 Number Class and its Methods	
10.3 Character Class and its Methods	
10.4 String Class and its Methods	
10.5 StringBuilder Class and its Methods	
10.6 StringBuffer Class and its Methods	
10.7 Autoboxing and Unboxing	
10.8 Summary	
10.9 Solutions/ Answer to Check Your Progress	
10.10 References/Further Readings	

---

## 10.0 INTRODUCTION

---

In programming, we use several data types as per our needs. There are *wrapper* classes for each primitive data types in Java. These are used to create the object of that data type. Each primitive type in Java has a corresponding wrapper class. You will learn about Number Class (an abstract class) and its subclasses in this unit. Also, we use characters and strings frequently in programming. Many programming languages as C and C++, store strings as arrays of characters. However, Java represent strings as a separate object type, called String. You have already used objects of String type in your programs of the previous units of this course. In Java mainly, four classes are there that you can use when working with character data: Character, String, StringBuilder, and StringBuffer. In this unit, you will learn different constructors, and operations like concatenation of strings, comparison of strings, insertion in a string etc. Also, you will learn about character extraction from a string, searching in a string, and conversion of different types of data into string form. Towards the end of this unit, you will learn about Autoboxing and Unboxing.

---

## 10.1 OBJECTIVES

---

After going through this unit you will be able to:

- explain fundamentals of characters and strings;
  - use different constructors of String, StringBuilder, and StringBuffer classes,
  - use special operations on String objects,
  - extract characters from a string,
  - perform different operations such as string searching, comparison of strings etc.,
  - data conversion using `valueOf()` methods,
  - use StringBuilder class and StringBuffer class and their methods, and
  - use Autoboxing and Unboxing.
- 

## 10.2 NUMBER CLASS AND ITS METHODS

---

So far you have used primitive data types for making variables and writing programs. Java primitive types are also called **literals which represent** fixed values in memory. However, sometimes you may need to use objects in place of primitives, for example, when you are using Object wrappers is generics where you have to use some Class, such as collections as List<Integer> or Map<Integer, Double>. Each of the primitive data types in Java platform has corresponding *wrapper* classes. The object of the wrapper class wraps (contains) its respective primitive data type which can be used to create an object of that data type. Figure 10.1 shows the Inheritance tree for primitive types.

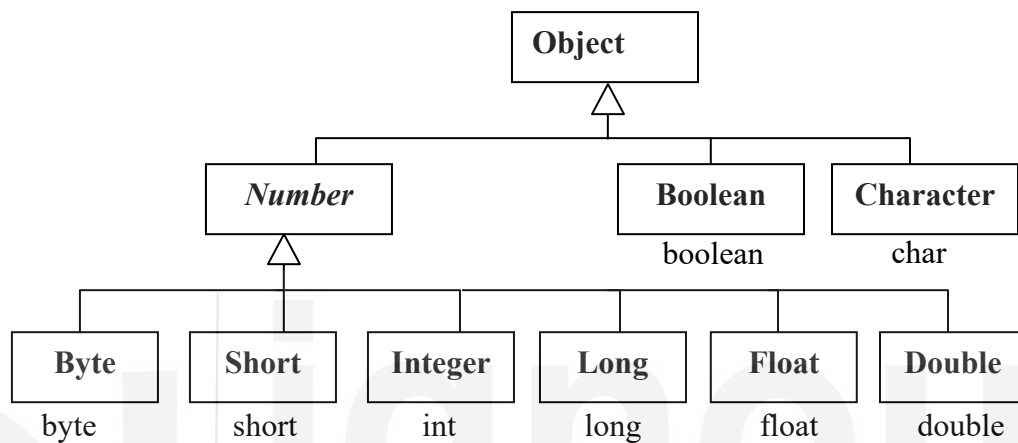


Figure 10.1: Inheritance tree for primitive types

Let us first learn about Number Class (an abstract class) and its subclasses. As you can see in figure 10.1 “each wrapper class has **Object** as a superclass. Byte, Short, Integer, Long Float and Double have **Number** as their direct superclass”. For all the primitives there are classes which “wrap” the primitive in an object. This wrapping is often done by the compiler. Here important point to note is that Objects of the wrapper classes will have a default value *null*, because they are reference types, but as you know, primitive types default value can never be null because in case a primitive type is not assigned a value, the Java will assign default value to it. This important behaviour of default value will help you to choose between using a primitive type or a wrapper object. One more feature of primitive wrapper objects is that they are final, therefore, they are immutable. In Java when a wrapper objects value is modified, the Java compiler create a new object and subsequently reassign that object to the original.

Unlike the primitive data types, the non-primitive objects are created by the users in Java. Some examples include arrays, strings, classes, interfaces etc. In Java when the reference variables are stored, the variable is stored in the stack and the original object is stored in the heap. In object data type although two copies are created, but they both 1 points to the same variable in the heap, hence changes made to any variable reflects the change in both the variables. Unless using a wrapper class is necessary, as a programmer you should generally choose primitive types over wrapper classes. Primitive types are faster than Wrapper Objects, at the same time advantage of Wrapper Objects is that they can be *null*.

### The Numbers Classes

When we are working with numbers, most of the time we use the primitive types such as int, float, double in our program. For example:

```
int age = 20;  
float price = 15.5;
```

In case when you are using a primitive in place where an object is expected, Java compiler *boxes* your primitive in its wrapper class for you. Similarly, when you are using a number object where a primitive is expected, the compiler *unboxes* the object for you. Detail about *Outboxing* and *Inboxing* you will learn in section 10.6 of this unit.

The following are three reasons to use a Number object rather than a primitive:

1. When a method expects an object as an argument .
2. Using constants defined by the class, such as `MIN_VALUE` which lower bounds of the data type and `MAX_VALUE` which provide the upper bounds of the data type.
3. When you need the class methods which convert values to and from other primitive types. For example, when you need to convert to and from strings or to convert between number systems.

The following table 10.1 , lists the methods of all the subclasses of the Number class.

**Table 10.1: Methods Implemented by all Subclasses of Number**

Method	Description
byte <code>byteValue()</code> short <code>shortValue()</code> int <code>intValue()</code> long <code>longValue()</code> float <code>floatValue()</code> double <code>doubleValue()</code>	These are used to convert the value of Number object to the primitive data type returned.
int <code>compareTo(Byte anotherByte)</code> int <code>compareTo(Double anotherDouble)</code> int <code>compareTo(Float anotherFloat)</code> int <code>compareTo(Integer anotherInteger)</code> int <code>compareTo(Long anotherLong)</code> int <code>compareTo(Short anotherShort)</code>	It compares this Number object to the argument.
boolean <code>equals(Object obj)</code>	Used to determine whether this number object is equal to the argument. This method returns true if the argument is not null and it is an object of the same type and with the same numeric value.

### Conversion Methods

Each Number class contains some methods used for converting numbers to and from strings and for converting between number systems. Table 10.2 lists the methods in the Integer class for conversion. It is to note that other Number subclasses also have similar classes.

**Table 10.2 : Conversion Methods of Integer Class**

Method	Description
static Integer <code>decode(String s)</code>	Decodes a string into an integer. This method can accept string representations of decimal, octal, or hexadecimal numbers as input.
static int <code>parseInt(String s)</code>	It returns an integer

## Number, Character And Strings

static int parseInt(String s, int radix)	It returns an integer value, of the given a string representation of decimal, binary, octal, or hexadecimal (radix equals 10, 2, 8, or 16, respectively) numbers as input.
String toString()	Returns a String object representing the value of this Integer.
static String toString(int i)	Returns a String object representing the specified integer.
static Integer valueOf(String s)	It returns an Integer object which is holding the value of the specified string parameter (value).
static Integer valueOf(int i)	It returns an Integer object which is holding the value of the specified primitive.
static Integer valueOf(String s, int radix)	It returns an Integer object which is holding the integer value of the specified string representation, parsed with the value of radix as arguments.

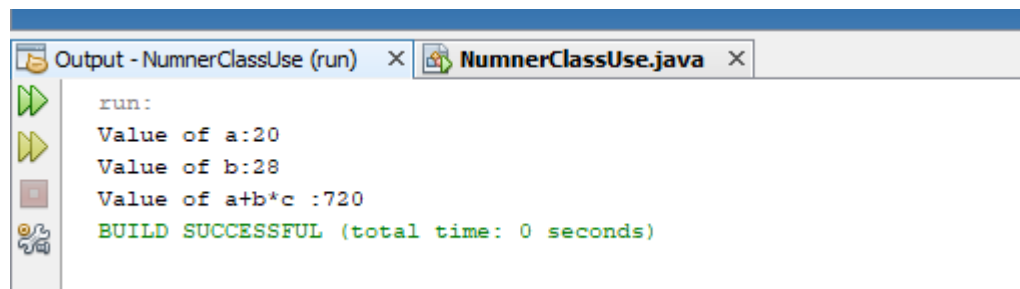
Now let us see use of some methods given above in the Java program.

### Example Program:

```
package numnerclassuse;
import static java.lang.Integer.parseInt;
import static oracle.jrockit.jfr.events.Bits.intValue;

public class NumnerClassUse
{
    public static void main(String[] args)
    {
        Integer I = 20; // I is object of Integer type
        Integer J = 25;
        String S = "28";
        int a, b, c, k ;
        a = intValue(I);
        b= parseInt(S); // getting integer value from String object
        c = intValue(J);
        System.out.println("Value of a:"+a);
        System.out.println("Value of b:"+b);
        k = a+b*c;
        System.out.println("Value of a+b*c :"+k);
    }
}
```

### Output



```
Output - NumnerClassUse (run) x NumnerClassUse.java x
run:
Value of a:20
Value of b:28
Value of a+b*c :720
BUILD SUCCESSFUL (total time: 0 seconds)
```

## Formatting Numeric for Print Output

In the earlier program where you used *println*, you have seen that the use of the *println* methods for printing strings to standard output (*System.out*) where all numbers are converted to strings. Using *println* you can print out an arbitrary mixture of strings and numbers. To add more control over your print-output options, the Java programming language provides some other methods, especially when numbers are included.

### The *printf* and *format* Methods

The *java.io* package includes *PrintStream* class. This class has two formatting methods which you can use to replace *println* method. These methods are *format* and *printf*, and are equivalent to one another.

The *System.out* that you have been using is a *PrintStream* object, therefore you can invoke *PrintStream* methods on *System.out*. Now you can try *format* or *printf* method anywhere in your code where you have previously been using *println*.

The syntax for *format* method and *print* methods of *java.io.PrintStream* is the same:  
`public PrintStream format(String format, Object... args)`

In the above method the argument *format* is a string. This is used to specify the formatting to be used, and the second argument *args* is a list of the variables which are to be printed using that formatting. Here the notation *Object... args* is called *varargs*, that indicates that the number of arguments may vary. The *format* string contains the plain text as well as the *format specifiers*, which contains characters that format the arguments of *Object... args*.

Let us see an example:

```
System.out.format("The value of " + "the integer variable is " +
    "%d, while the value of the " + "float variable is %f, " +
    "and the string is %s", intVar, floatVar, stringVar);
```

Example code:

```
int var1= 520;
```

```
System.out.format("The value of var1 is: %d%n", var1);
```

Where *%d* specifies that the single variable is a decimal integer. The *%n* is a platform-independent newline character. The output is:

```
The value of var1 is: 520
```

The *format* method has the following syntax:

```
public PrintStream format(Locale l, String format, Object... args)
```

*Locale* is used for localization. If *Locale l* is null then no localization is applied. For example, to print numbers in the German system (in which in the English representation of floating point numbers a comma is used in place of the decimal place), you may use:

```
System.out.format(Locale.GERMAN, "The value of the float variable is: %f, and the
" +
    "value of the integer variable is: %d %n", floatVar, intVar);
```

The following table 10.3 has a list of some converters and flags that are used in the program given below

**Table 10.3: Converters and Flags**

Converter	Flag	Explanation
d		A decimal integer.
f		A float.
n		A new line character
	08	Eight characters in width, with leading zeroes as necessary.
	+	Includes sign, whether positive or negative
	,	Includes locale-specific grouping characters.
	.3	Three places after decimal point.
	10.4	Ten characters in width, right justified, with four places after decimal point.

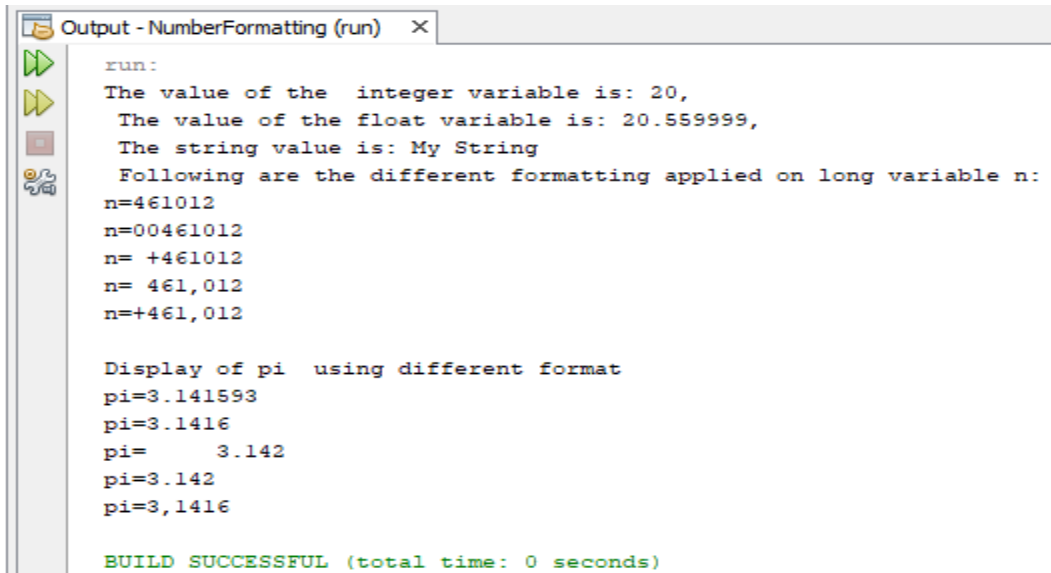
**Example Pogram:**

```

package numberformatting;
import java.util.Calendar;
import java.util.Locale;
public class NumberFormatting
{
    public static void main(String[] args)
    {
        long n = 461012;
        double pi = Math.PI;
        int intV= 20;
        float floatV = 20.56f;
        String stringV = "My String";
        System.out.format("The value of " + "the integer variable is: " + "%d,%n The
value of the " +
        "float variable is: %f,%n " + "The string value is: %s %n", intV, floatV, stringV);
        System.out.println(" Following are the different formatting applied on long
variable n:");
        System.out.format("n=%d%n", n);
        System.out.format("n=%08d%n", n);
        System.out.format("n=%+8d%n", n);
        System.out.format("n=%,8d%n", n);
        System.out.format("n=%+,8d%n%n", n);
        System.out.println("Display of pi using different format");
        System.out.format("pi=%f%n", pi);
        System.out.format("pi=%.4f%n", pi);
        System.out.format("pi=%10.3f%n", pi);
        System.out.format("pi=%-10.3f%n", pi);
        System.out.format(Locale.GERMAN, "pi=%-10.4f%n%n", pi);
    }
}

```

**Output:**



```

Output - NumberFormatting (run) x
run:
The value of the integer variable is: 20,
The value of the float variable is: 20.559999,
The string value is: My String
Following are the different formatting applied on long variable n:
n=461012
n=00461012
n= +461012
n= 461,012
n=+461,012

Display of pi using different format
pi=3.141593
pi=3.1416
pi= 3.142
pi=3.142
pi=3,1416

BUILD SUCCESSFUL (total time: 0 seconds)

```

## 10.3 CHARACTER CLASS AND ITS METHODS

In problem-solving, we often need to deal with variables (objects) of character and string type. In Java the following classes are provided to deal with characters and strings.

**Character Class:** Objects of this class hold a single character value only. Some methods are defined which can manipulate or check single-character data.

**String Class:** Objects of the String class are used to deal with the strings. The objects created using this class cannot be changed (immutable). This class provides many methods to perform different activities on string objects.

**StringBuilder Class:** Objects of the StringBuilder class are like String class objects, except that they can be modified. Internally, objects of the StringBuilder class are treated like variable-length arrays of characters. At any point, you may change the length and content of the objects through the invocation of the StringBuilder class methods.

**StringBuffer Class:** You can create string objects using this class, which is expected to be manipulated during the process.

### Character Class

An object of the Character class can contain only a single character value. While programming, if you need to use a single character value, most of the time you have to use the primitive char type. For example:

```

char ch = 'A';
// an array of chars
char[] charArray = { 'A', 'B', 'C', 'D', 'E' };

```

You sometimes need to use a Character object instead of a primitive char variable. For example, when you are required to pass a character value in a method while it changes the value passed as an argument or when you need to place a character value in a Java-defined data structure, where the requirement is to work on objects. For example, Vector is one of the data structures that requires objects. You have already seen Java; we have the *wrapper* class that “wraps” the char in a Character object. As

the Character class is immutable, i.e. once an object is created, it cannot be changed. You can create a Character object using the Character class constructor:

```
Character ch = new Character('a');
```

The ch is a character object created using Character class. Also, under some circumstances, the Java compiler creates a Character object for you. In Java when you pass a primitive char into a method as an argument which expects an object as an argument, then the Java compiler will automatically convert the char to a Character object for you. The following table 10.4 shows some of the commonly used Character class methods.

**Table 10.4: Some Methods of Character Class**

Method	Description
boolean isLetter(char ch) boolean isDigit(char ch)	This method tells whether the given char value is a letter or a digit.
boolean isWhitespace(char ch)	This method tells whether the specified char value is white space or not.
boolean isUpperCase(char ch) boolean isLowerCase(char ch)	This method tells whether the given char value is uppercase or lowercase.
char toUpperCase(char ch) char toLowerCase(char ch)	It returns the uppercase or lowercase form of the given char value.
toString(char ch)	It returns a String object representing the specified character value( i.e. a one-character string).
boolean compateTo( Character)	This method is used for comparing the values held by two character objects.
boolean equals( Character)	This method is used for comparing the values held by two character objects.
int compareTo (Character)	This method compare the value of the object on which this method is called with the value of the object passed as argument to the method.

Now let us see an example in which Character objects are created and used. In the following program some character objects are created, some Character class methods are used and results are displayed.

**Program**

```
package characterclassexample;
public class CharacterClassExample
{
public static void main(String args[])
{
Character C1 = new Character('J');
Character C2 = new Character('a');
Character C3 = new Character('v');
Character C4 = new Character('a');
int diff;
diff = C1.compareTo(C2);
if (diff == 0)
System.out.println(C1+" is equal to "+ C2);
else
System.out.println(C1+" is not equal to "+ C2);
System.out.println("Value of C1 is: " + C1);
System.out.println("Value of C2 is: " + C2);
System.out.println("Value of C3 is: " + C3);
```

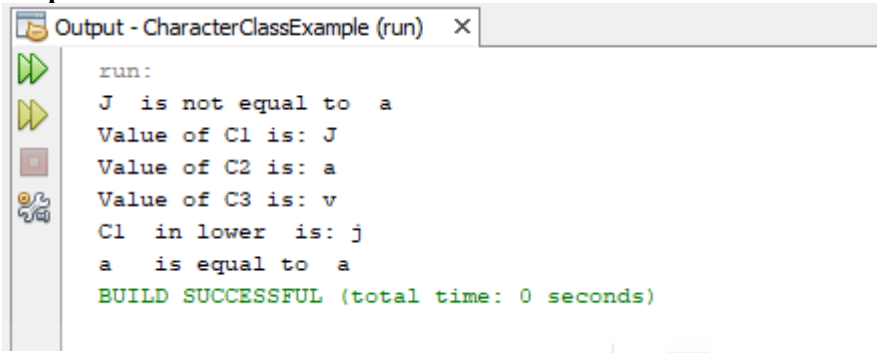


```

System.out.println("C1 in lower is: " + Character.toLowerCase(C1));
diff = C2.compareTo(C4);
if (diff == 0)
System.out.println(C2+" is equal to "+C4);
else
System.out.println(C1+" is not equal to "+C2);
}
}

```

**Output:**



**Check Your Progress – 1**

1) What is wrapper class? When is Number class used?

.....

.....

.....

.....

2) Write a program which displays the case (upper or lower) of a character object.

.....

.....

.....

.....

3) Explain the use of equals () method of Character class with the help of code statements.

.....

.....

.....

.....

---

## 10.4 STRING CLASS AND ITS METHODS

---

We use Strings widely in Java programming. Strings are a sequence of characters. In Java String, class is there to create and manipulate strings. In Java programming we may create objects of String class and use them. One important point to remember

here is that the strings value of the String class objects will not change because String is immutable.

For example, if you create the following string object. The string contained by the object MyString given below cannot be changed.  
String MyString = "This Java String cannot be changed!"

### Creating Objects of String Class

Following is the direct way to create a string:

```
String MyStringObj = "Welcome to Java String!";
```

Here " Welcome to Java String!" is a *string literal*. It is a series of characters in enclosed in double quotes (" "). Whenever Java compiler encounters a string literal in a program code it creates a String object with its value. In the above code, it is, Welcome to Java String!.

The String class has many constructors. Some of them are given below.

**public String():** This constructor creates a String object which represents an empty character sequence.

**public String(String value):** This constructor is used for creating String objects that represent the same sequence of characters as passed in the argument.

**public String(char value[]):** This constructor you may use to create a new String object which is contained in the character array argument.

**public String(byte bytes[], int offset, int length):** This constructor is used to create a String object by converting the given sub-array of bytes using the platform's default character encoding.

**public String(byte bytes[]):** This constructor is used to create a String object by converting the given array of bytes using the platform's default character encoding.

As String class have many important methods for:  
Examining individual characters of the strings,  
Comparing strings,  
Searching strings,  
Extracting sub-strings, and  
Creating a copy of a string

One important point to remember is that if you create a String object, and if it has the same value as another String object, in that case, Java will point both the object's references to the same memory location.

### Creating Strings Objects

As you create any other object, you can create objects of String class using the new keyword and a constructor. The String class has many constructors which can be used to provide the initial value of the string using different sources, such as an array of characters:

```
char[] MyString = { 'B', 'o', 'o', 'k' };  
String Str1 = new String(MyString);  
System.out.println(Str1);
```

The last line of above code snippet will display: Book.

**Table 10.5 : Some Search Methods in the String Class**

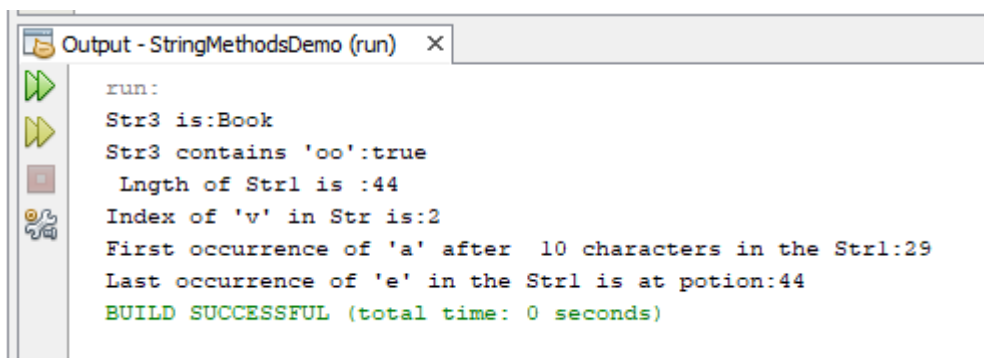
Method	Description
int indexOf(int ch) int lastIndexOf(int ch)	These methods returns the index of the first/last occurrence of the given character.
int indexOf(int ch, int fromIndex) int lastIndexOf(int ch, int fromIndex)	These methods returns the index of the first/last occurrence of the given character, searching forward (backward) from the specified index.
int indexOf(String str) int lastIndexOf(String str)	These methods returns the index of the first/ last occurrence of the given substring.
int indexOf(String str, int fromIndex) int lastIndexOf(String str, int fromIndex)	These methods return the index of the first/last occurrence of the specified substring, searching forward/backward from the specified index.
boolean contains(CharSequence s)	This method returns true if the string contains the specified character sequence.

The program given below demonstrates the use of some index methods.

**Program**

```
package stringmethodsdemo;
public class StringMethodsDemo
{
    public static void main(String[] args)
    {
        String Str1 = "Java is Object Oriented Programming Language";
        char[] Str2 = {'B', 'o', 'o', 'k'};
        String Str3 = new String(Str2);
        System.out.println("Str3 is:"+Str3);
        System.out.println("Str3 contains 'oo':" +Str3.contains("oo"));
        System.out.println(" Lngth of Str1 is :"+ Str1.length());
        System.out.println("Index of 'v' in Str is:"+Str1.indexOf('v'));
        // Print first occurrence of character v in string Str1
        System.out.println("First occurrence of 'a' after 10 characters in the Str1:" +
        Str1.indexOf('a',9));
        // Print first occurrence of character a in string Str1 after first 10 characters
        System.out.println("Last occurrence of 'e' in the Str1 is at potion:" +
        (Str1.lastIndexOf('e')+1));
        // Print Last occurrence of character e in string Str1
    }
}
```

**Output:**



### String class Method length ( )

This method is used to get the length of the string object. If you execute the following lines of code, you will find the length of Str1:

```
String Str1 = "My First String";  
int len = Str1.length();
```

### String Class Method charAt(int index)

This method return character at index i of the given String.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
S	H	R	I	K	A	N	T		M	I	S	H	R	A							

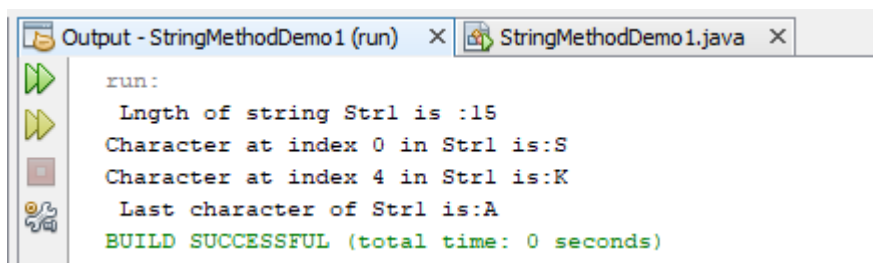
```
charAt (0): S                      charAt (9) : M  
charAt (length () -1): ?
```

Following program demonstrate use of lenth ( ) and charAt ( int index) methods of String class.

#### Program

```
package stringmethoddemo1;  
public class StringMethodDemo1  
{  
    public static void main(String[] args)  
    {  
        String Str1 = "SHRIKANT MISHRA";  
        System.out.println(" Lngth of string Str1 is :"+ Str1.length());  
        System.out.println("Character at index 0 in Str1 is:"+Str1.charAt(0));  
        System.out.println("Character at index 4 in Str1 is:"+Str1.charAt(4));  
        System.out.println(" Last character of Str1 is:"+Str1.charAt(Str1.length()-1));  
    }  
}
```

#### Output:



### Conversion Between Numbers and Strings

#### Converting Strings to Numbers

We need some time to convert numeric data in a string object to a numeric value. This conversion is done using some pre-defined methods valueOf. The Number subclasses that wrap primitive numeric types ( Byte, Integer, Double, Float, Long, and Short) each subclass provide method *valueOf* ( ), which converts a string to an object of that

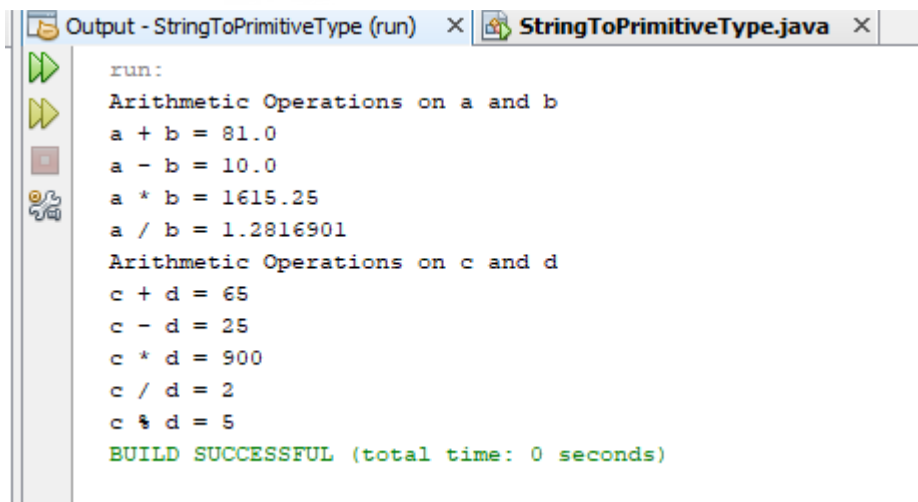
type. The following program converts the given strings to numbers and subsequently arithmetic operations are performed on the values.

Now let us see the example which uses the `valueOf ( )` method to convert a string to a primitive value.

**Program:**

```
package stringtoprimitivetype;
public class StringToPrimitiveType
{
    public static void main(String[] args)
    {
        String s1 = "45.5";
        String s2 = "35.5";
        String s3 = "45";
        String s4 = "20";
        // convert strings to float value
        float a = Float.valueOf(s1); //convert to float
        float b = Float.valueOf(s2); //convert to float
        //some arithmetic operations on a and b
        System.out.println("Arithmetic Operations on a and b");
        System.out.println("a + b = " + (a + b));
        System.out.println("a - b = " + (a - b));
        System.out.println("a * b = " + (a * b));
        System.out.println("a / b = " + (a / b));
        // convert strings to int value
        int c = Integer.valueOf(s3); // convert to integer
        int d = Integer.valueOf(s4); //convert to integer
        System.out.println("Arithmetic Operations on c and d");
        System.out.println("c + d = " + (c + d));
        System.out.println("c - d = " + (c - d));
        System.out.println("c * d = " + (c * d));
        System.out.println("c / d = " + (c / d));
        System.out.println("c % d = " + (c % d));
    }
}
```

**Output:**



```
Output - StringToPrimitiveType (run) × StringToPrimitiveType.java ×
run:
Arithmetic Operations on a and b
a + b = 81.0
a - b = 10.0
a * b = 1615.25
a / b = 1.2816901
Arithmetic Operations on c and d
c + d = 65
c - d = 25
c * d = 900
c / d = 2
c % d = 5
BUILD SUCCESSFUL (total time: 0 seconds)
```

**valueOf() Method for Converting Numbers to Strings:**

As a programmer many times you need to convert a number to a string so that you may operate on the value in its string form. In the table 10.6 you can see some methods of String class which are used for converting Numbers to String.

**Table 10.6: Methods for Converting Numbers to Strings**

Method	Description
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 subarray of the character array argument

In the following program, you can see that integer and float data are converted into a string type. Also, the operator '+' is used for concatenating two strings.

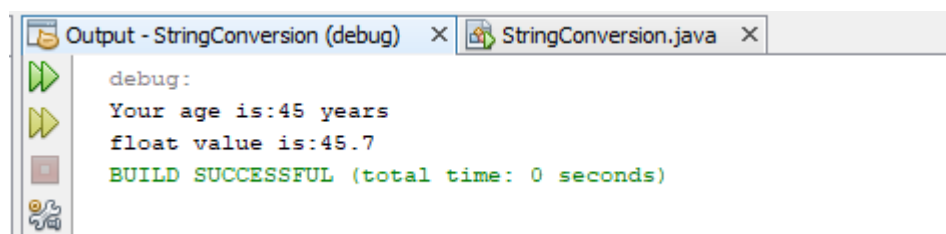
**Program**

```

package stringconversion;
public class StringConversion
{
public static void main(String[] args)
{
String s1 = "Your age is:";
String s2 = new String();
String s3 = "float value is:";
int age = 45;
float f=45.7f;
s2 = String.valueOf(age);
s1 = s1+s2+ " years";
System.out.println(s1);
s2 = String.valueOf(f);
System.out.println(s3+s2);
}
}

```

**Output:**

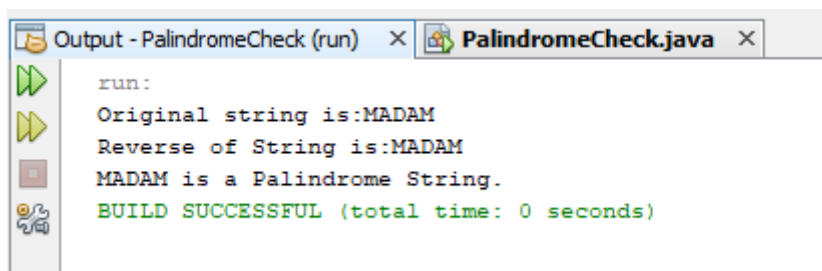


A *palindrome* is a word or sentence that is symmetric in nature, and it is spelt the same from both forward and backward, ignoring case and punctuation. Now let us see a Java program that checks whether a given String is a *palindrome* or not.

### Program

```
package palindromecheck;
public class PalindromeCheck
{
    public static void main(String[] args)
    {
        String MyPal = "MADAM";
        int len = MyPal.length();
        char[] tempCharArray = new char[len];
        char[] ChArray = new char[len];
        System.out.println("Original string is:"+ MyPal);
        // in this loop we put original string in an array of chars
        for (int i = 0; i < len; i++)
        {
            tempCharArray[i] = MyPal.charAt(i);
        }
        // loop to reverse array of chars
        for (int j = 0; j < len; j++)
        {
            ChArray[j] = tempCharArray[len - 1 - j];
        }
        String reversePal = new String(ChArray);
        System.out.println("Reverse of String is:"+ reversePal);
        if (MyPal.toLowerCase().equals(reversePal.toLowerCase()))
        {
            System.out.println(MyPal + " is a Palindrome String.");
        }
        else
        {
            System.out.println(MyPal + " is not a Palindrome String.");
        }
    }
}
```

### Output:



```
Output - PalindromeCheck (run) x PalindromeCheck.java x
run:
Original string is:MADAM
Reverse of String is:MADAM
MADAM is a Palindrome String.
BUILD SUCCESSFUL (total time: 0 seconds)
```

**Table 10.7: String Class Comparisons Methods**

Method	Description
public boolean equals(Object str)	Return true when 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 startsWith (String prefix, int toffset)	Return true if prefix occurs at index toffset.
public boolean regionMatches(int toffset, String other, int offset, int len)	Return true if both the string have exactly the same symbols in the given region.
public boolean startsWith(String prefix)	Return true if string starts with prefix.
public boolean endsWith(String suffix)	Return true if string ends with suffix.

**Table 10.8: Methods for Modifying Strings**

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

The following program shows how to use the String class methods for comparisons and modifying strings.

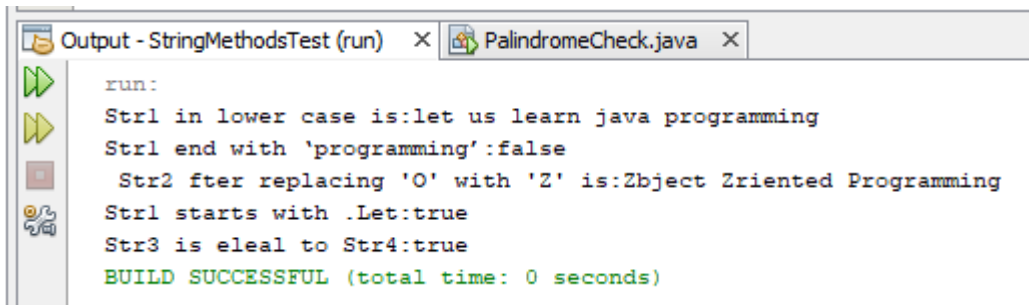
**Program**

```

package stringmethodstest;
public class StringMethodsTest
{
    public static void main(String[] args)
    {
        String Str1 = "Let us learn java programming";
        String Str2 = "Object Oriented Programming";
        String Str3 = "JAVA";
        String Str4 = "java";
        String Str5;
        char a = 'O';
        char b = 'Z';
        System.out.println("Str1 in lower case is:"+Str1.toLowerCase());
        System.out.println("Str1 end with 'programming':" + Str1.endsWith("Programming"));
        Str5= Str2.replace(a,b);
        System.out.println(" Str2 fter replacing 'O' with 'Z' is:"+Str5);
        System.out.println("Str1 starts with .Let:"+Str1.startsWith("Let"));
        System.out.println("Str3 is equal to Str4:"+Str3.equalsIgnoreCase(Str4));
    }
}

```



**Output:**


```

run:
Str1 in lower case is:let us learn java programming
Str1 end with 'programming':false
Str2 fter replacing 'O' with 'Z' is:Zbject Zriented Programming
Str1 starts with .Let:true
Str3 is elead to Str4:true
BUILD SUCCESSFUL (total time: 0 seconds)

```

**Some More String Class Methods**

String class proves some methods for getting substring, comparison, and string modifying methods. These methods are useful in handling string objects, such as finding a character's location in a string, getting substrings, concatenation of strings, comparing strings, and matching string reasons, etc.

**Table 10.9: Substring Mettthods**

Methods	Description
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 (int beginIndex, int endIndex)	Return substring from index beginIndex to the endIndex of string
public String concat(String str)	Return a concatenated string formed by the append/joining of two strings.
public char[] toCharArray()	Return character array of string

**Explanation of String Class concat Method**

This method of the String class is used for concatenating two strings. Syntax of using this method is:

```
string1.concat(string2);
```

The above code will return a new string which is the combination of string1 with string2 added to it at the end.

You can also use the **concat() method** with string literals, as in:

```
"This is ".concat(" Java Programming Course");
```

Also, strings are commonly concatenated with the '+' operator. You have widely used '+' operator in println statements. For example, the following code is an example of string concatenation:

```
String string1 = "This is my String";
System.out.println("string1 is:" + string1);
```

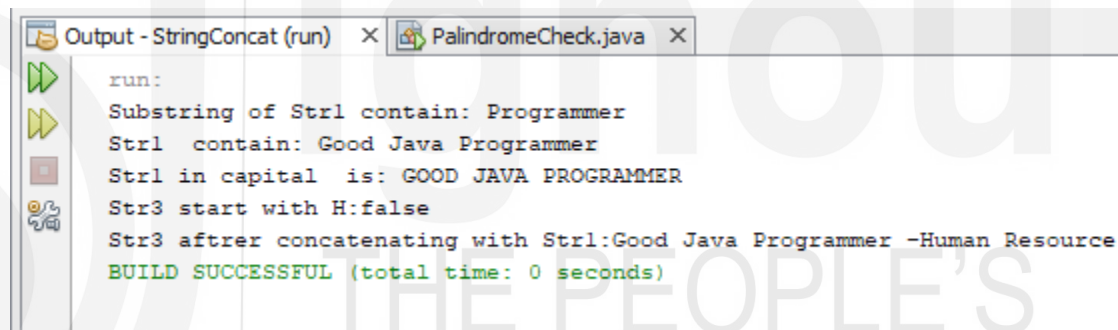
The above code will print: string1 is: This is My String.

The following program demonstrates the use of substring methods and string concatenation method, and substring method.

**Program:**

```
package stringconcat;
public class StringConcat
{
public static void main(String[] args)
{
String Str1 = "Good Java Programmer ";
String Str2 ;
String Str3 = "-Human Resource";
int index = Str1.indexOf('P');
Str2 =Str1.substring(index);
System.out.println("Substring of Str1 contain: " + Str2);
System.out.println("Str1 contain: " + Str1);
System.out.println("Str1 in capital is: " + Str1.toUpperCase());
System.out.println("Str3 start with H:"+ Str3.startsWith("H"));
Str2 = Str1.concat(Str3);
System.out.println("Str3 after concatenating with Str1:"+Str2);
}
}
```

**Output:**



---

## 10.5 STRINGBUILDER CLASS AND ITS METHODS

---

StringBuilder is another class in Java which is used for creating string objects. StringBuilder class objects are like String class objects, except that they can be modified. StringBulder objects are internally treated like variable-length arrays in Java. You can change the length and content of the sequence of StringBuilder objects at any point in time using StringBuilder class methods. The StringBuilder class is faster, as it performs no synchronization and it should generally be used in preference to String class. Concatenation of StringBuilder objects is more efficient compared to String objects. StringBuilder class supports all of the same operations as of the String class.

### StringBuilder Class Constructors

StringBuilder(): This constructor is used to create an empty string builder with a capacity of 16.

StringBuilder(CharSequence cs): This constructor is used to create a string builder containing the same characters as the specified CharSequence, plus an extra 16 empty

elements which are trailing the CharSequence. The **CharSequence** is a readable sequence of char values.

**StringBuilder(int initCapacity):** This constructor is used to create an empty string builder with the specified initial capacity.

**StringBuilder(String s):** This constructor is used to create a string builder, which is initialized by the specified string, plus an extra 16 empty elements which are trailing in the string.

Some methods of the StringBuilder class are

### length and capacity Methods of StringBuilder

The StringBuilder class have a `length()` method which returns the length of the string builder object. Also, every string builder has a *capacity*. The capacity is the number of character spaces that have been allocated to an string builder object and it is returned by the `capacity()` method. Capacity of string builder is always greater than or equal to the length (usually greater than) of object. The capacity of string builder object automatically expand when there is a need to accommodate additions to the string builder object.

index	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
value	S	T	R	I	N	G										

=16 `length() = 6` `capacity() = 16`

### Methods of StringBulder class

**void setLength(int newLength):** This method is used for setting the length of the character sequence. If `newLength` is less than `length()` of string builder object then the last characters which are remaining after `newLength` coverage in the character sequence are truncated. If `newLength` is greater than `length()`, then null characters are appended at the end of the character sequence.

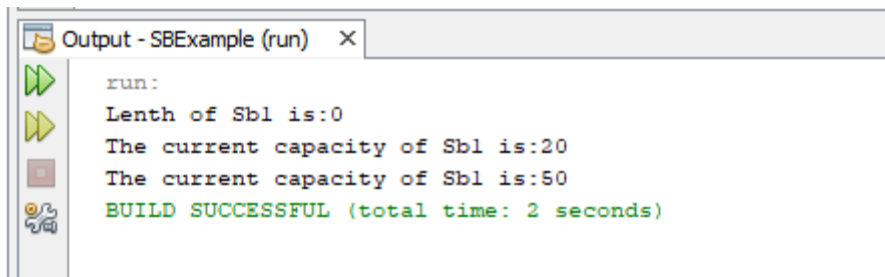
**void ensureCapacity(int minCapacity):** This method ensures that the capacity of the object is at least equal to the capacity passed as argument i.e. `minCapacity`.

### Program

```
package sbexample;
public class SBExample
{
    public static void main(String[] args)
    {
        StringBuilder Sb1 = new StringBuilder(20);
        int len, cap ;
        len = Sb1.length();
        System.out.println("Lenth of Sb1 is:"+len);
        cap = Sb1.capacity();
        System.out.println("The current capacity of Sb1 is:"+ cap);
        Sb1.ensureCapacity(50);
        cap = Sb1.capacity();
        System.out.println("The current capacity of Sb1 is:"+ cap);
    }
}
```

}

**Output:**



**StringBuilder Operations**

Some operations on a StringBuilder that are not available in String are the append() and insert() methods. These methods are overloaded so that they can accept data of any type. Each method first converts the argument passed to it to a string, subsequently they append or insert the characters of that string to the character sequence in the string builder.

When we use the append method, it always adds the characters at the end of the existing character sequence. If we use the insert method, we can add the characters at a specified point in the object.

**Table 10.10 : Some More StringBuilder Methods**

Method	Description
StringBuilder append(boolean b) StringBuilder append(char c) StringBuilder append(char[] str) StringBuilder append(char[] str, int offset, int len) StringBuilder append(double d) StringBuilder append(float f) StringBuilder append(int i) StringBuilder append(long lng) StringBuilder append(Object obj) StringBuilder append(String s)	These methods are used to append the argument to the string builder object on which it is called. The argument value passed is converted to a string before the append operation takes place.
StringBuilder delete(int start, int end)	This method deletes the subsequence from the start point to end-1 (inclusive) in the StringBuilder's object.
StringBuilder deleteCharAt(int index)	This method deletes the character located at index.
StringBuilder insert(int offset, boolean b) StringBuilder insert(int offset, char c) StringBuilder insert(int offset, char[] str) StringBuilder insert(int index, char[] str, int offset, int len) StringBuilder insert(int offset, double d) StringBuilder insert(int offset, float f) StringBuilder insert(int offset, int i) StringBuilder insert(int offset, long lng) StringBuilder insert(int offset, Object obj) StringBuilder insert(int offset, String s)	These methods are used for Inserting the second argument into the string builder. In the methods, the first integer argument indicates the index before which the data passed as the second argument is to be inserted. Remember that the data passed as a second argument is converted to a string before the insert operation takes place.
StringBuilder replace(int start, int end,	This method replaces the specified

String s) void setCharAt(int index, char c)	character(s) in this string builder.
StringBuilder reverse()	This method is used to reverse the sequence of characters in this string builder object.

When we use `append()`, `insert()`, or `setLength()` methods, they increase the length of the character sequence in the object of the string builder. This results to increase in length () , and it would be greater than the current capacity of the string builder object. Therefore when such an operation is processed, the capacity is automatically increased.

The program given below demonstrates the use of the `append` and `reverse` methods of `StringBuilder`.

**Program:**

```
package sbreversetest;
public class SBReverseTest
{
    public static void main(String[] args)
    {
        // creates empty builder, capacity 16
        StringBuilder Sb1 = new StringBuilder();
        System.out.println("Sb1 capacity is:"+Sb1.capacity());
        // adds ... character string at beginning
        Sb1.append("Welcome to Java Programming");
        System.out.println("Sb1 capacity after append operation is:"+Sb1.capacity());
        System.out.println("Sb1 contains:"+Sb1);
        System.out.println("Length of Sb1 is:"+Sb1.length());
        System.out.println("Reverse of Sb1 is:"+Sb1.reverse());
    }
}
```

**Output:**

```
run:
Sb1 capacity is:16
Sb1 capacity after append operation is:34
Sb1 contains:Welcome to Java Programming
Length of Sb1 is:27
Reverse of Sb1 is:gnimmargorP avaJ ot emocleW
BUILD SUCCESSFUL (total time: 0 seconds)
```

Also, you can use any `String` class method on a `StringBuilder` object. To use this facility, first, you have to convert the `StringBuilder` object to a `String` object with the `toString()` method of the `StringBuilder` class. After that, using object using the `StringBuilder(String str)` constructor, you can convert the `String` object back into a `StringBuilder`.

Now, the following program that we have implemented in section "Strings". This program is about reversing a palindrome. Here you can see that the program written here is more efficient when we have used a `StringBuilder` instead of a `String`.

**Program using String class:**

## Number, Character And Strings

```
package palindromecheck;
public class PalindromeCheck
{
    public static void main(String[] args)
    {
        String MyPal = "MADAM";
        int len = MyPal.length();
        char[] tempCharArray = new char[len];
        char[] ChArray = new char[len];
        System.out.println("Original string is:"+ MyPal);
        // in this loop we put original string in an array of chars
        for (int i = 0; i < len; i++)
        {
            tempCharArray[i] = MyPal.charAt(i);
        }
        // loop to reverse array of chars
        for (int j = 0; j < len; j++)
        {
            ChArray[j] = tempCharArray[len - 1 - j];
        }
        String reversePal = new String(ChArray);
        System.out.println("Reverse of String is:"+ reversePal);
        if (MyPal.toLowerCase().equals(reversePal.toLowerCase()))
        {
            System.out.println(MyPal + " is a Palindrome String.");
        }
        else
        {
            System.out.println(MyPal + " is not a Palindrome String.");
        }
    }
}
```

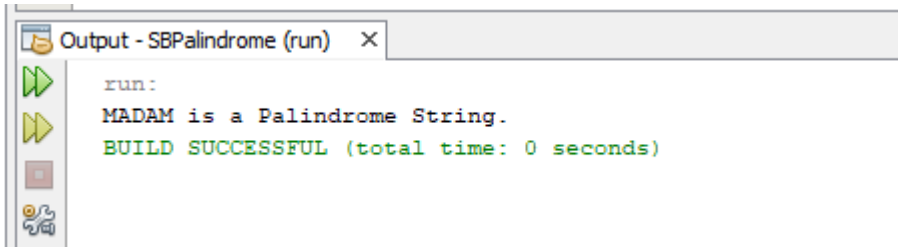
The program of palindrome checking is written using StringBuilder class below. You can see that the below-written program is more simple and small compared to the above program. It is possible because, in a string builder, there is a method named reverse(), which is used to make the code simpler and easier to read:

### Program:

```
package sbpalindrome;
public class SBPalindrome
{
    public static void main(String[] args)
    {
        String MyPal = "MADAM";
        StringBuilder Sb = new StringBuilder(MyPal);
        StringBuilder S3;
        // reverse Sb
        S3= Sb.reverse();
        if (Sb.equals(S3))
        {
            System.out.println(MyPal + " is a Palindrome String.");
        }
        else
        {
            System.out.println(MyPal + " is not a Palindrome String.");
        }
    }
}
```

```
}
}
```

**Output:**



**Check Your Progress – 2**

1) Write a java program which displays digits after dot(.) of a double data type variable.

.....

.....

.....

2) What will be the output of the following program?

```
package strtest;
public class StrTest
{
public static void main(String[] args)
{
String Str = new String("Sring Handling in Java");
System.out.println("Length of Str is :"+Str.length());
int i = Str.lastIndexOf('a');
int j = Str.indexOf('a');
System.out.println("Difference between first and last occurrence of 'a' in MyStr is
:"+ (i-j));
}
}
```

.....

.....

.....

3) Write a java program using StringBuilder which append Str2 to Str1 where Str1 is “Java is Simple” and Str2 is “Programming Language”. Also reverse the resultant string after apped operation.

.....

.....

.....

---

## 10.6 STRINGBUFFER CLASS AND ITS METHODS

---

The String class objects are of fixed length and cannot be modified. If there is a need of strings to be modified, then you have another option of using StringBuffer class object. Now StringBuffer class is almost obsolete. Almost all the features provided in StringBuffer class are supported by StringBuilder class which you have learned in the previous section. Now let us have a look at StringBuffer class in detail. String buffers class is generally used for constructing string objects dynamically when you need a string which may change. The StringBuffer class and the StringBuilder class are the same, except that StringBuffer class is thread-safe because its methods are synchronized, which means StringBuffer objects are safe for use by multiple threads. Major operations on a StringBuffer are the append and insert methods. These methods are overloaded so that they can accept data of any type.

### StringBuffer Class Constructors

**StringBuffer():** Used to define StringBuffer object with no characters in it, and it has the initial capacity of 16 characters.

**StringBuffer (int size):** It is used to construct StringBuffer object with no characters in it and with an initial capacity specified by the size argument.

**StringBuffer (String str):** It is used to construct StringBuffer object with the given sequence of characters as the string argument.

### Some Methods of String Buffer Class

StringBuffer provides the functionality of String class methods plus some more methods which are used :

- to modify strings by inserting substrings,
- for deleting contents of string,
- to append string, and
- for altering size of string dynamically.

**public int length():** This method returns the length of the string buffer object.

**public int capacity():** This method returns the total allocated capacity to the string buffer.

Every string buffer object has a capacity. Until the length of the character sequence contained in the object is less than this capacity, there is no need to allocate a new internal buffer array for the object. If the internal buffer overflows (increases more than capacity), it is automatically made larger.

**public void ensureCapacity (int MinCapacity):** This method ensures the capacity of the buffer at least equal to the specified MinCapacity.

When the current capacity of the string buffer object ( on which **ensureCapacity** method is called) is less than the argument value, then a new internal buffer is allocated which has greater capacity . The new capacity of the object is the larger than:

- i). The MinCapacity argument
- ii). Twice the old capacity, plus 2 .
- iii) If you pass a non-positive value as the MinCapacity then this method takes no action.



`public void setLength(int newLength)` : This method is used for setting the length of the string buffer object. If the new length is less than the current length of the object then, the string content is truncated so that it only contains exactly the same number of characters as given by the `newLength` argument. In case the `newLength` argument is greater than or equal to the current length of the object, there is no truncation of characters; rather object is appended with sufficient null characters so that its length increases to the `newLength` argument.

`public void setCharAt(int index, char ch)`: This method is used to place given character 'ch' at the position given by `index` in the object. This operation result into altering the character sequence because of addition of new character in object. After this operation the existing characters at `index` and characters after that in the object are shifted right by one position.

`public StringBuffer append(String newStr)` This method appends the `newStr` to the `StringBuffer` object. When you use this operation it result into increasing the length of the object.

`public StringBuffer append(int i)`: This method can be used to append the string representation of the `int` argument to the string buffer. Internally the argument passed is converted to a string then appended to the string buffer.

`public StringBuffer insert (int offset,char[] str)`: Using this method you can insert the string representation of the character array argument into the string buffer object . The position of insertion is the value indicated by `offset`. Using this operation increases the length of the `StringBuffer` object by the length of the argument passed to `append`.

`public StringBuffer insert (int offset, String str)`: Using this method you can insert a given string in the string buffer object at position indicated by `offset`. Its original contents of the object move up its characters after the `offset` position.

`public StringBuffer reverse()`: This method reverses the string contained in the `StringBuffer` object.

`public StringBuffer deleteCharAt(int index)` : This method removes the character located at the `index` position in the string buffer object.

`public StringBuffer replace (int start, int end, String Str)`: This method is used to replace the characters of the string buffer object from `start` index point to the `end` index point by the argument `Str`.

`public StringBuffer delete (int start, int end)`: This method deletes characters ( total characters delete:  $end - start + 1$  ) from location `start` and up to including location `end`.

Following program use of some of the methods like `capacity`, `append`, `replace`, `delete`, and `reverse` etc. of `StringBuffer` class. In the program given below `capacity` of the `StringBuffer` object `Sb1` never changes, while the length of the `Sb1` increased after inserting elements in it.

#### Program:

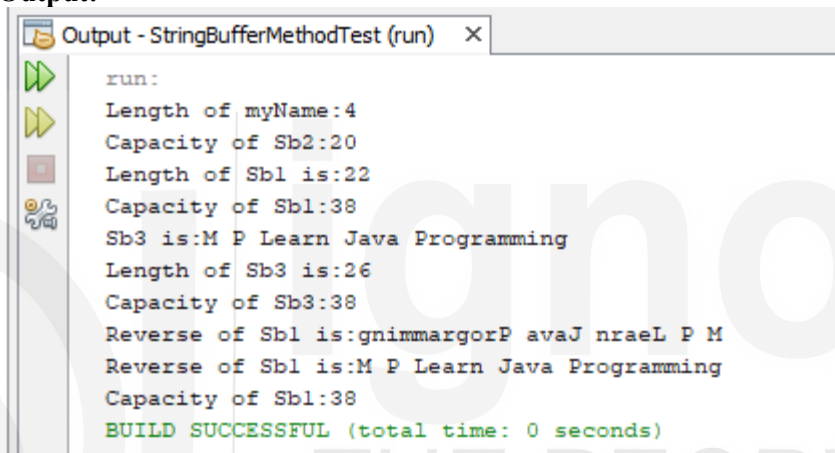
```
package stringbuffermethodtest;

public class StringBufferMethodTest
{
    public static void main ( String args[])
    {
        StringBuffer Sb1 = new StringBuffer("Learn Java Programming");
        String myName = new String("M P ");
    }
}
```

## Number, Character And Strings

```
StringBuffer Sb2 = new StringBuffer(myName);
StringBuffer Sb3;
System.out.println("Length of myName:"+Sb2.length());
System.out.println("Capacity of Sb2:"+Sb2.capacity());
System.out.println("Length of Sb1 is:"+Sb1.length());
System.out.println("Capacity of Sb1:"+Sb1.capacity());
Sb3 = Sb1.insert(0,myName);
System.out.println("Sb3 is:"+Sb3);
System.out.println("Length of Sb3 is:"+Sb3.length());
System.out.println("Capacity of Sb3:"+Sb3.capacity());
System.out.println("Reverse of Sb1 is:"+Sb1.reverse());
//reverse back to original string
System.out.println("Reverse of Sb1 is:"+Sb1.reverse());
System.out.println("Capacity of Sb1:"+Sb1.capacity());
}
}
```

### Output:



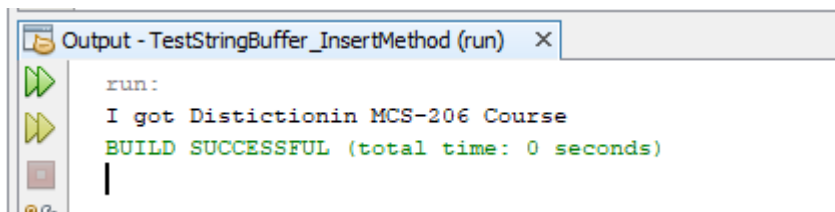
```
Output - StringBufferMethodTest (run) x
run:
Length of myName:4
Capacity of Sb2:20
Length of Sb1 is:22
Capacity of Sb1:38
Sb3 is:M P Learn Java Programming
Length of Sb3 is:26
Capacity of Sb3:38
Reverse of Sb1 is:gnimmargorP avaJ nraeL P M
Reverse of Sb1 is:M P Learn Java Programming
Capacity of Sb1:38
BUILD SUCCESSFUL (total time: 0 seconds)
```

Now let us see one more example program in which you can insert data into the middle of a StringBuffer object using insert methods.

### Program:

```
package teststringbuffer_insertmethod;
public class TestStringBuffer_InsertMethod
{
public static void main ( String args[])
{
StringBuffer MyBuffer = new StringBuffer("I got in MCS-206 Course");
MyBuffer.insert(6,"Distiction");
System.out.println(MyBuffer.toString());
}
}
```

### Output:



```
Output - TestStringBuffer_InsertMethod (run) x
run:
I got Distictionin MCS-206 Course
BUILD SUCCESSFUL (total time: 0 seconds)
```

## 10.7 AUTOBOXING AND UNBOXING

**Autoboxing:** Automatic conversion made by the Java compiler between the primitive types and their corresponding object wrapper classes is called *Autoboxing*. For example, it converts an int primitive type to an Integer object ,or a double primitive type to a Double, and so on. When this conversion is done in the other way by the Java compiler, it is known as *unboxing*.

Consider the following code:

```
List<Integer> li = new ArrayList<>();
for (int i = 1; i < 20; i ++)
    li.add(i);
```

In the above code, the int values of i( primitive type value) are added in the List object li, rather than Integer objects ( as required by List), the code compiles. As li is a List of Integer objects, not a List of int values, still the Java compiler does not issue a compile-time error because it creates an Integer object from the primitive int value i and adds the object to li. Actually, the compiler converts the above code to the following at runtime:

```
List<Integer> li = new ArrayList<>();
for (int i = 1; i < 20; i++)
    li.add(Integer.valueOf(i));
```

Therefore here, when a primitive value (an int, for example) is converted into an object of the corresponding wrapper class (Integer), it is called autoboxing.

The process of autoboxing is applied by the Java compiler when a primitive value is:

- Passed in a method as a parameter and that parameter expects an object of the corresponding wrapper class.
- Assigned to a variable of the corresponding wrapper class.

### Unboxing

In Java unboxing is a process through which an object of a wrapper type, let us say Integer, is converted to its corresponding primitive type (int) value. The unboxing is applied by the Java compiler when an object of a wrapper class:

- Is passed as an argument to a method where a value of the corresponding primitive type is expected.
- Is assigned to a variable of the corresponding primitive type.

Following is a Java program to demonstrate how the unboxing works:

### Program:

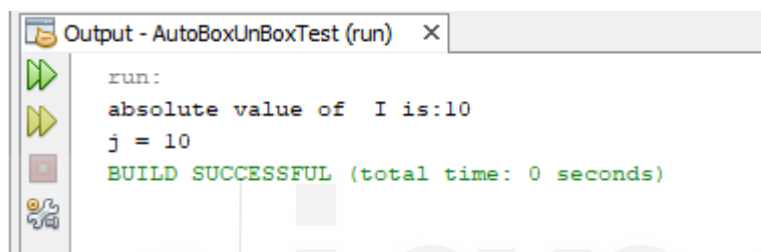
```
package autoboxunboxtest;
import java.util.ArrayList;
import java.util.List;
public class AutoBoxUnBoxTest
{
    public static void main(String[] args)
    {
```

```

Integer I = new Integer(10);
// Unboxing through method invocation
int iVal = I.intValue();
System.out.println("absolute value of I is:" + iVal);
List<Integer> li = new ArrayList<>();
// Autoboxing in add( ) method
li.add(iVal);
// Unboxing through assignment
int j=li.get(0);
System.out.println("j = " + j);
}
}

```

**Output:**



As a Java programmer, you should avoid using autoboxing and unboxing for scientific computing, or in other applications which are having performance-sensitive numerical code. Both autoboxing and unboxing help Java developers in writing cleaner code. Also, it makes code easier to read as an Integer is not a substitute for a primitive int value. In fact when autoboxing and unboxing processes are performed, they only blur the difference between primitive types and reference types, but they are unable to totally eliminate it.

The table given below shows the primitive types and their corresponding wrapper classes. These are used by the Java compiler for autoboxing and unboxing:

**Table 10.11 : Primitive Type and Corresponding Wrapper Classes**

Primitive type	Wrapper class
boolean	Boolean
byte	Byte
char	Character
float	Float
int	Integer
long	Long
short	Short
double	Double

**☞ Check Your Progress – 3**

- 1) What is the initial capacity of the following StringBuilder object Sb1? Also, find substring(6, 12) and charAt(9) in StringBbuilder Sb1 given below.  
 StringBuilder Sb1 = new StringBuilder("Now StringBuilder is used in place of StringBuffer");

.....  
 .....  
 .....

2) When should a StringBuffer object be preferred over a String object?

.....  
 .....  
 .....

3) Write a Java program which displays the length and initials of the given name (string) of a person using the StringBuffer class.

.....  
 .....  
 .....

4) What will be the output of the following program?

**Program:**

```
package arraylistsum;
import java.util.ArrayList;
public class ArrayListSum
{
    public static void main(String[] args)
    {
        ArrayList<Integer> list = new ArrayList<>();
        //autoboxing
        list.add(1);
        list.add(2);
        list.add(3);
        list.add(4);
        System.out.println("ArrayList: " + list);
        // unboxing
        int sum =0;
        for(Integer i:list)
            sum =sum+i;
        System.out.println("The sum of list elements is:"+sum);
    }
}
```

.....  
 .....  
 .....



---

## 10.8 SUMMARY

---

Dealing with characters and strings are among the routine job in programming. Java provides Character, String, StringBuilder and StringBuffer classes for handling characters and strings. Also, Java provides *wrapper* classes for each primitive data type. These classes are used to create object of that data type. Each primitive type in Java has a corresponding wrapper class. Java String class is used for creating the string objects which cannot be changed. This unit explained about wrapper classes, Number class and its conversion methods. Character class provides various methods like compareTo, equals, isUpperCase. This unit explains about use of Character class. Also, use of printf and format method are explained in this unit. Subsequently String class and some of its methods are explained. In general string classes is not used for dynamic Strings, for that purpose StringBuilder class and StringBuffer class and their methods are explained. Finally concept of autoboxing and unboxing are explained.

---

## 10.9 SOLUTIONS/ ANSWER TO CHECK YOUR PROGRESS

---

### ☞ Check Your Progress – 1

1) For all the primitives, there are classes that “wrap” the primitive in an object. The Number Class (an abstract class) and its subclasses are wrapper classes, and each wrapper class has **Object** as a superclass. Byte, Short, Integer, Long, Float and Double have **Number** as their direct superclass. For more details, refer to section 10.2 of this unit.

The following are three reasons to use a Number object rather than a primitive:

- i. When a method expects an object as an argument.
- ii. Using constants defined by the class, such as MIN\_VALUE, which lowers the bounds of the data type and MAX\_VALUE, which provide the upper bounds of the data type.
- iii. When there is a need to use class methods for converting values to primitive types and vice-versa. For example, when you need to convert to and from strings or to convert between number systems.

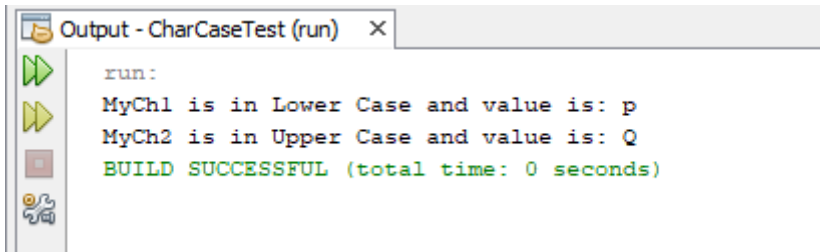
2) //Program to test whether the content of a Character object is Upper or Lower

```
package charcasetest;
public class CharCaseTest
{
    public static void main(String args[])
    {
        Character MyCh1 = new Character('p');
        Character MyCh2 = new Character('Q');
        //Test for MyCh1
        if (MyCh1.isUpperCase(MyCh1.charValue()))
            System.out.println("MyCh1 is in Upper Case and value is: "+MyCh1);
        else
            System.out.println("MyCh1 is in Lower Case and value is: "+MyCh1);
        // Test for MyCh2
        if (MyCh2.isUpperCase(MyCh2.charValue()))
            System.out.println("MyCh2 is in Upper Case and value is: "+MyCh2);
        else
            System.out.println("MyCh2 is in Lower Case and value is: "+MyCh2);
    }
}
```

```

    }
}

```

**Output:**


```

Output - CharCaseTest (run) x
run:
MyCh1 is in Lower Case and value is: p
MyCh2 is in Upper Case and value is: Q
BUILD SUCCESSFUL (total time: 0 seconds)

```

3) This instance method is used to compare the value held by the current object with the value held by another object. For example, let MyCh1 and MyCh2 be two objects of Character class then

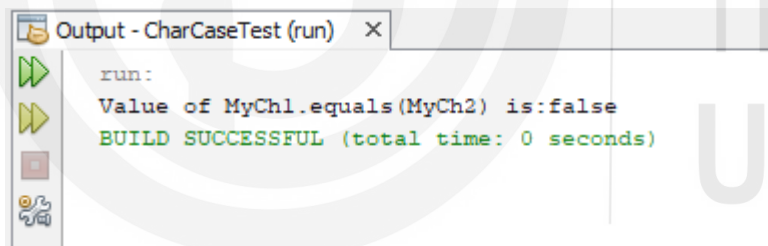
```
MyCh1.equals(MyCh2);
```

the method returns true if the values held by both objects are equal; otherwise false.

```

package charcasetest;
public class CharCaseTest
{
    public static void main(String args[])
    {
        Character MyCh1 = new Character('p');
        Character MyCh2 = new Character('Q');
        System.out.println("Value of MyCh1.equals(MyCh2)
is:"+MyCh1.equals(MyCh2));
    }
}

```

**Output:**


```

Output - CharCaseTest (run) x
run:
Value of MyCh1.equals(MyCh2) is:false
BUILD SUCCESSFUL (total time: 0 seconds)

```

**Check Your Progress – 2**

1)

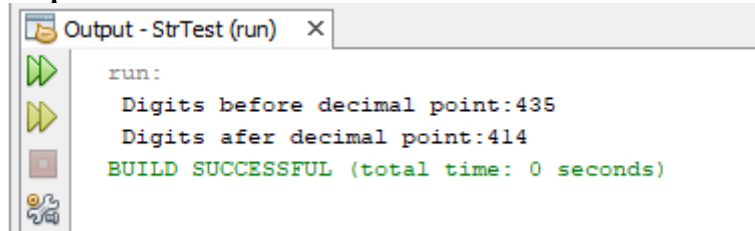
**Program**

```

package strtest;
public class StrTest
{
    public static void main(String[] args)
    {
        double d = 435.414;
        String Str = Double.toString(d);
        int dot = Str.indexOf('.');
        int len = Str.length();
        System.out.println(" Digits before decimal point:"+Str.substring(0,dot));
        System.out.println(" Digits afer decimal point:"+Str.substring(dot+1, len));
    }
}

```

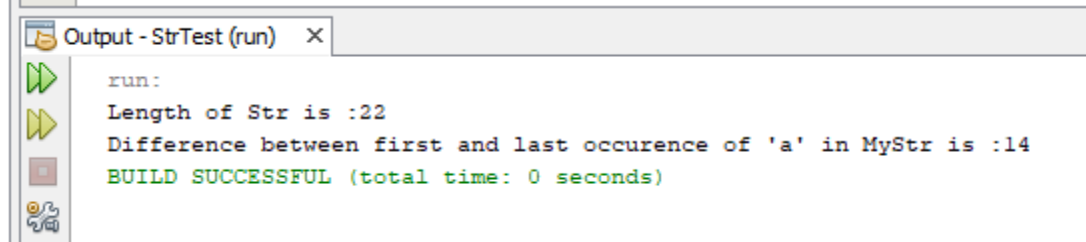
**Output:**



```
Output - StrTest (run) x
run:
  Digits before decimal point:435
  Digits afer decimal point:414
BUILD SUCCESSFUL (total time: 0 seconds)
```

2)

**Output:**



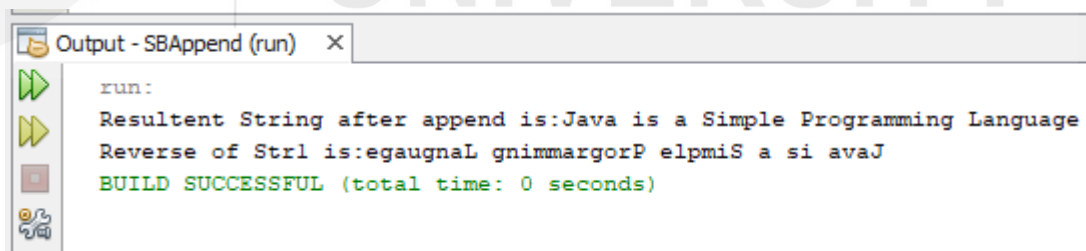
```
Output - StrTest (run) x
run:
  Length of Str is :22
  Difference between first and last occurrence of 'a' in MyStr is :14
BUILD SUCCESSFUL (total time: 0 seconds)
```

3)

**Program:**

```
package sbappend;
public class SBAppend
{
  public static void main(String[] args)
  {
    StringBuilder Str1 = new StringBuilder("Java is a Simple ");
    StringBuilder Str2 = new StringBuilder("Programming Language");
    Str1.append(Str2);
    System.out.println("Resultent String after append is:"+Str1);
    System.out.println("Reverse of Str1 is:"+Str1.reverse());
  }
}
```

**Output:**



```
Output - SBAppend (run) x
run:
  Resultent String after append is:Java is a Simple Programming Language
  Reverse of Str1 is:egaugnaL gnimmargorP elpmiS a si avaJ
BUILD SUCCESSFUL (total time: 0 seconds)
```

**Check Your Progress – 3**

1)

**Program:**

```
package stringbuildertest;
public class StringBuilderTest
{
  public static void main(String[] args)
  {
    StringBuilder Sb1 = new StringBuilder("Now StringBuilder is used in place of
StringBuffer");
    String Sb2;
```



```

System.out.println("Capacity of Sb1 is:"+ Sb1.capacity());
Sb2 = Sb1.substring(6, 12);
System.out.println("Substring Sb1.substring(6,12) of given string is:"+Sb2);
System.out.println("Character at index 9 in Sb1 is:"+ Sb1.charAt(9));
}
}

```

**Output:**

```

Output - StringBuilderTest (run) x
run:
Capacity of Sb1 is:66
Substring Sb1.substring(6,12) of given string is:ringBu
Character at index 9 in Sb1 is:g
BUILD SUCCESSFUL (total time: 0 seconds)

```

**2)**

The StringBuffer class object should be used when there is a need of modification (change may take place) in the content of the string. When we insert some content in an object of StringBuffer class, it increases the size of the object as per need. Therefore StringBuffer object should be given preference over String objects when you need:

- i. to insert a substring in a given string.
- ii. to reverse the content of a string object.

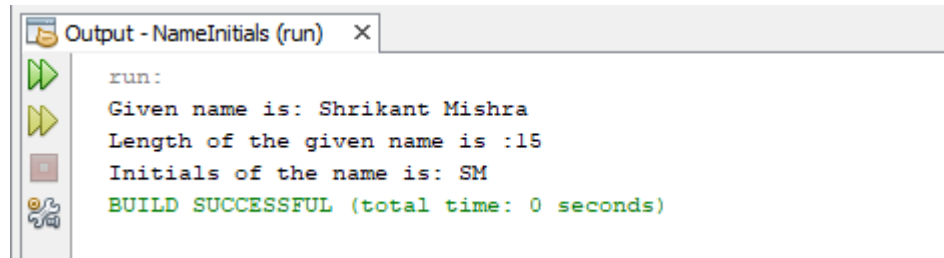
**3)****Program:**

```

package nameinitials;
public class NameInitials
{
    public static void main(String[] args)
    {
        String Str1 = "Shrikant Mishra";
        StringBuffer Str2 = new StringBuffer();
        // Find initial of the given name
        int len = Str1.length();
        for (int i = 0; i < len; i++)
        {
            if (Character.isUpperCase(Str1.charAt(i)))
            {
                Str2.append(Str1.charAt(i));
            }
        }
        System.out.println("Given name is: " + Str1);
        System.out.println("Length of the given name is :"+len);
        System.out.println("Initials of the name is: " + Str2);
    }
}

```

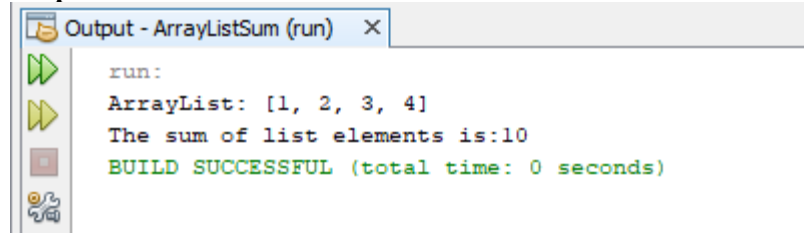
**Output:**



```
Output - NameInitials (run) X
run:
Given name is: Shrikant Mishra
Length of the given name is :15
Initials of the name is: SM
BUILD SUCCESSFUL (total time: 0 seconds)
```

4)

**Output:**



```
Output - ArrayListSum (run) X
run:
ArrayList: [1, 2, 3, 4]
The sum of list elements is:10
BUILD SUCCESSFUL (total time: 0 seconds)
```

---

## 10.10 REFERENCES/FURTHER READINGS

---

- MCS-024 Course of IGNOU BCA(Revised ) Programme, Block -3 Unit-3: Strings and Characters ( <https://egyankosh.ac.in/handle/123456789/11742>)
- Herbert Schildt, “Java The Complete Reference”, McGraw-Hill,2017.
- <https://docs.oracle.com/javase/tutorial/java/data/numberclasses.html>
- <https://docs.oracle.com/javase/tutorial/java/data/characters.html>
- <https://docs.oracle.com/javase/tutorial/java/data/buffers.html>
- <https://docs.oracle.com/javase/tutorial/java/data/buffers.html>
- <https://docs.oracle.com/javase/tutorial/java/data/autoboxing.html>
- <https://medium.com/@bpnorlander/java-understanding-primitive-types-and-wrapper-objects-a6798fb2afe9>