CHAPTER - 3

CLASS & OBJECTS

3. Class:

A class is a non-primitive datatype that contains member variables (data) and member functions (methods) that operates on the variables. A class is declared by the use of the class keyword. The general form of class definition is as follows.

Syntax: class class-name

```
{
        datatype instance-variable1;
        datatype instance-variable2;
        -
        -
        datatype instance-variablen;
        datatype methodname1(parameter-list)
        {
        // method body
        }
        -
        -
        datatype methodnamem(parameter-list)
        {
        // method body
        }
}
```

The data, or variables defined with in a class are called instance variables. The code is present with in methods. Collection of methods and variables defined with in a class are called members of the class.

Example: class box

```
{
double width, height, depth;
double volume( )
{
return width*height*depth;
}
}
```

3.1 Creating Objects

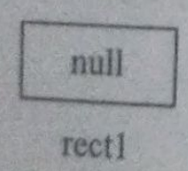
An Object in Java is a block of memory that contains space to store all the instance variables. Creating an object of a class is a two-step process.

1. Declare a variable of the class type. At this stage, the variable refers to an object only.
2. Assigning an actual, physical copy of the object and assign it to that variable by means of 'new' operator. The new operator dynamically allocates memory for an object and returns a reference to that object.

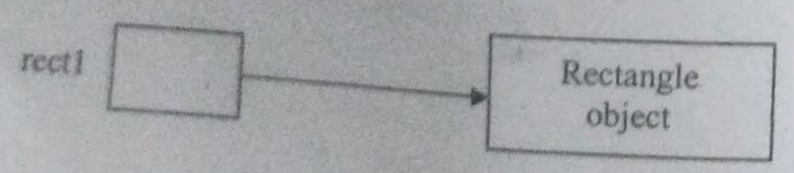
Example:

```
Rectangle rect1;            // reference
rect1 = new Rectangle();    // instantiate
        (or)
Rectangle rect1 = new Rectangle();
```

The first statement declares a variable to hold the object reference.



The second statement assigns the object reference and allocates memory

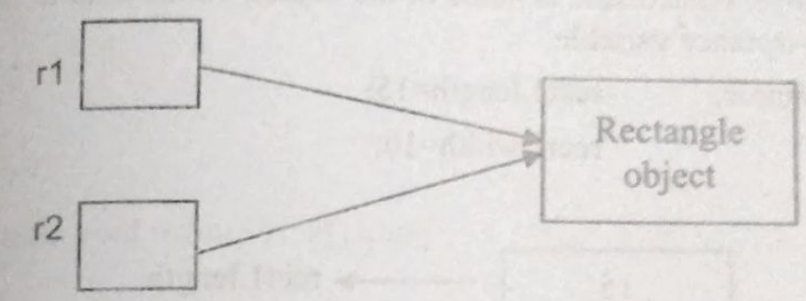


Note: Each object contains its own instance variables of its class. This means that any changes to the variables of one object have no effect on the variables of another. But the function name is name for both objects.

It is also possible to create two or more references to the same object.

Example:

```
Rectangle r1 = new Rectangle( );
Rectangle r2 = r1;
```



3.2. Methods

Methods allows the programmer to modularize a program. All variables declared in method definition are local variables and method parameters are also local variables. All methods must define inside a class definition. The general form of a method is as

Syntax:

```
type method-name (parameter-list)
{
        // body of the method
}
```

where,

- type specifies the type of data returned by the method. If the method does not return any value, its return type must be void
- The method-name is a valid identifier
- parameter-list is a sequence of type and identifier pairs separated by commas. The variable receives the value of the arguments passed to the method when it is called. If the method has no parameters, then the parameter list will be empty
- body of the method describes the operations to be performed on the data

Example:

```
int sum(int a, int b)
{
int c;
c=a+b;
return c;
}
```

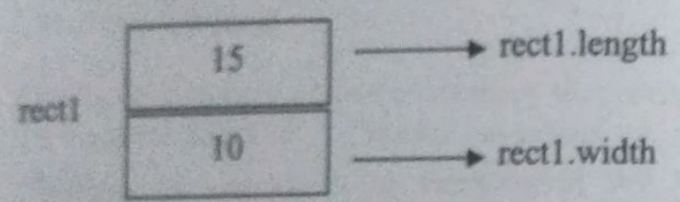
Accessing class members: All variables must be assigned values before they are used in our program. All class members are accessed by means of "dot" operator.

The general format of accessing instance variable of the class by dot operator is as

Syntax: objectname.variablename;

Where, objectname is name of the object, variablename is the name of the instance variable.

Example: rect1.length=15;
rect1.width=10;



The general format of accessing instance methods of the class by dot operator is as

Syntax: objectname.methodname(parameter-list);

Where, objectname is name of the object, methodname is the name of the instance method that calls with the object and paramerter-list is a list of actual values separated by commas.

Example:
```
void getdata(int x, int y)
{
length=x;
width=y;
}

rect1.getdata(10,20);
```

Example:

// Write a program to calculate the area of a rectangle.

```
class Rectangle
{
int length1,width1;
void getdata(int x,int y)
{
length1=x;
```



```
width1=y;
}
int calarea()
{
int area=length1*width1;
return area;
    }
}
class prg
{
    public static void main(String[] args)
    {
     int a,b;
     Rectangle rect1=new Rectangle();
     Rectangle rect2=new Rectangle();
     rect1.length1=15;
     rect1.width1=4;
     a=rect1.length1*rect1.width1;
     System.out.println("Area of Rectangle1="+a);
     rect2.getdata(2,3);
     b=rect2.calarea();
     System.out.println("Area of Rectangle2="+b);
    }
}
```

3.3. Method Overloading:

In Java it is possible to define two or more methods within the same class that share the same name as long as these methods have different sets of parameters (Based on the number of parameters, the types of the parameters and the order of the parameters). Then the methods are said to be overloaded, and the process is referred as Method Overloading.

When the overload method is called the Java compiler selects the proper method by examine the number, type and order of the

arguments in the call. Method Overloading is commonly used to create several methods with same name that perform fast accessing.

Note: Two methods differ in only by return type will result in a syntax error.

Example1:

```
class overload
{
    int square(int x)
    {
    return x*x;
    }
    double square(double x)
    {
    return x*x;
    }
}
class check
{
    public static void main(String[] args)
    {
     overload r1=new overload();
     int k=r1.square(3);
     double p=r1.square(4.0);
     System.out.println("k value="+k);
     System.out.println("p value="+p);
    }
}
```

Example2:

```
class overload1
{
    int sum(int x)
    {
```



```
    return x+x;
    }
    int sum(int x,int y,int z)
    {
    return x+y+z;
    }
}
class check1
{
    public static void main(String[] args)
    {
     overload1 r1=new overload1();
     int k=r1.sum(3,5,7);
     System.out.println("k value="+k);
    }
}
```

3.4 Constructors

A constructor is a special method of class, which is invoked automatically, whenever an object of a class is created. It has the same name as its class and resides similar to a method. A constructor has the following characteristics.

- It doesn't have any return type not even void
- The constructor can be declared as public
- It can be overloaded
- It is normally used to initialize the member of the object

Different types of Constructors:

1) **Default Constructor:** A constructor that accepts no parameters is called the default constructor. If no constructors are defined for a class, the Java system automatically generates the default constructor.

Example:

```
class Time1
{
    int hour,min,sec;
    public Time1()
    {
    hour=10;
    min=45;
    sec=23;
    }
    void printdata()
    {
    System.out.println(hour+"hours:"+min+"minutes:"+sec+"seconds");
    }
}
class const1
{
    public static void main(String[] args)
    {
     Time1 t=new Time1();
     t.printdata();
    }
}
```

O/P: 10hours:45minutes:23seconds

2) **Parameterized Constructor:** A constructor that takes arguments as parameters is called parameterized constructor. If any constructors are defined by a class with the parameters, Java will not create a default constructor for the class.

Example:

```
class Time1
{
    int hour,min,sec;
    public Time1(int h,int m,int s)
    {
    hour=h;
    min=m;
    sec=s;
    }
    void printdata()
    {
    System.out.println(hour+"hours:"+min+"minutes:"+sec+"seconds");
    }
}
class const2
{
    public static void main(String[] args)
    {
     Time1 t=new Time1(10,9,4);
     t.printdata();
    }
}
```

O/P: 10hours:9minutes:4seconds

3.5 Constructor Overloading:

An interesting feature of the constructor is that a class can have multiple constructors. This is called Constructor Overloading. All the constructors have the same name as corresponding to the class name that they are differ only interms of their signature as number of arguments or datatypes or order.

Example:

```
class Time1
{
    int hour,min,sec;
    public Time1()
    {
    hour=0;
    min=0;
    sec=0;
    }
    public Time1(int h)
    {
    hour=h;
    min=0;
    sec=0;
    }
    public Time1(int h, int m)
    {
    hour=h;
    min=m;
    sec=0;
    }
    public Time1(int h,int m,int s)
    {
    hour=h;
    min=m;
    sec=s;
    }
    void printdata()
    {
    System.out.println(hour+"hours:"+min+"minute:"+sec+"sec");
    }
}
```



```
class const3
{
    public static void main(String[] args)
    {
     Time1 t=new Time1();
     t.printdata();
     Time1 t1=new Time1(9);
     t1.printdata();
     Time1 t2=new Time1(7,8);
     t2.printdata();
     Time1 t3=new Time1(10,9,4);
     t3.printdata();
    }
}
```

3.6 Static Class Members

Generally each class contains instance variables and instance methods. Every time the class is instantiated, a new copy of each of them is created. They are accessed using the objects with the dot operator.

If we define a member that is common to all the objects and accessed without using a particular object i.e., the member belong to the class as a whole rather than the objects created from the class. Such facility can be possible by using "static" keyword.

The members declared with the static keyword are called static members.

Example:
```
static int x;
static int show(int x,int y);
```

The important points about static class members are

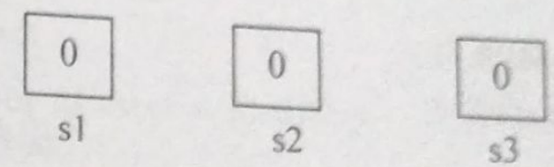
- Static members can be accessible without the help of objec s. It can be accessible by using the syntax: classname.members;
- A static method can access only static members (data + method)
- A static block will be executed first (before main). Static variables are initialized first
- A static member data store the same memory for all objects
- Static method cannot be referred to 'this' or 'super' in anyway.

Example 1:

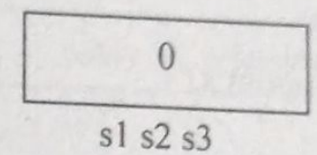
```
class static1
{
        static int c=0;
        void increment()
        {
        c++;
        }
        void display()
        {
        System.out.println("Value="+c);
        }
}
class Mstatic
{
        public static void main(String[] args)
        {
        static1 s=new static1();
        static1 s1=new static1();
        static1 s2=new static1();
        s.increment();
        s1.increment();
        s2.increment();
        s.display();
        s1.display();
        s2.display();
        }
}
```

O/P: value=3
value=3
value=3

without static keyword



with static keyword



Example 2:

```
class static1
{
        static int a=10;
        static int b=34;
        static void call()
        {
        System.out.println("Value1="+a);
        }
}
class Mstatic1
{
        public static void main(String[] args)
        {
        static1.call();
        System.out.println("Value2="+static1.b);
        }
}
```

O/P: Value1=10
Value2=34

Example 3:

```
class static2
{
    static int a=10,b;
    static void method(int x)
    {
    System.out.println("x="+x);
    System.out.println("a="+a);
    System.out.println("b="+b);
    }
    static
    {
    System.out.println("static block");
    b=a*4;
    }
}
class Mstatic2
{
    public static void main(String[] args)
    {
    static2.method(35);
    }
}
```

```
O/P:  static block
      x=35
      a=10
      b=40
```


3.7. Access Modifiers/Control

Java provides four types of Access Modifiers. They are public, protected, default and private.

Access Location	public	protected	friendly or default	private
Same class	Y	Y	Y	Y
Sub class in same package	Y	Y	Y	N
Other classes in same package	Y	Y	Y	Y
Sub classes in other packages	Y	Y	N	N
Non-sub classes in other packages	Y	N	N	N

For good programming technique place member data as private and member functions as public.

Example:

```
class modi
{
    private int k=6;
    void display()
    {
    System.out.println("k in display="+k);
    }
}
class ex
{
    public static void main(String[] args)
    {
    modi obj=new modi();
    obj.display();
    //System.out.println("k in main="+obj.k);
    //this statement provides error since private variable
    //can not be possible to access by other class
    }
}
```

3.8. this keyword

Java define "this" keyword, that can be used inside any method to refer the current object. "this" is always a reference to the object on which method was invoked. "this" reference is implicitly used to refer to both the instance variables and methods of current object.

Example 1:

```
class Box
{
    int len,dep,hei;
    Box(int len,int dep, int hei)
    {
    this.len=len;
    this.dep=dep;
    this.hei=hei;
    }
    int volume()
    {
    return len*dep*hei;
    }
}
class this1
{
    public static void main(String[] args)
    {
     Box obj=new Box(1,2,3);
     int x=obj.volume();
     System.out.println("volume="+x);
    }
}
```

O/P: volume=6

"this" reference is also used explicitly refer to the instance variables.

Example 2:

```
class Box
{
    int len=10;
    void method()
    {
    int len=40;
    System.out.println("local value="+len);
    System.out.println("Instance value="+this.len);
    }
}
class this2
{
    public static void main(String[] args)
    {
     Box obj=new Box();
     obj.method();
    }
}
```

O/P: local value=40
Instance value=10

3.9. Argument Passing:

Arguments passed to a sub-routine (method) is falls into two categories.
1. Call by value
2. Call by reference

1. ***Call by Value:*** Call by value copies the value of an argument into the formal parameter of the sub-routine. Therefore, if changes made to the parameter of the sub-routine have no effect on the argument used to call it.

Example:

```
class Test
{
    void meth(int i, int j)
    {
    i*=2;
    j/=2;
    }
}
```

```
class cbv
{
    public static void main(String[] args)
    {
    Test obj=new Test();
    int a=4,b=7;
    System.out.println("Before calling a="+a+" b="+b);
    obj.meth(a,b);
    System.out.println("After  calling a="+a+" b="+b);
    }
}
```

O/P: Before calling a=4 b=7
After calling a=4 b=7

2. ***Call by Reference:*** In this method, a reference to an argument is passed to the parameter. Inside the parameter, this reference is used to access the actual argument specified in the call. Therefore, any changes made to the parameter will effect the argument used to call it. This one is also referred as passing an object as parameter to a method.

Example:

```
class Test
{
    int a,b;
    Test(int i,int j)
    {
    a=i;
    b=j;
    }
    void meth(Test ox)
    {
    ox.a*=2;
    ox.b/=2;
    }
}
```



```
class cbr
{
    public static void main(String[] args)
    {
    Test obj=new Test(4,5);
    System.out.println("Before calling a="+obj.a+" b="+obj.b);
    obj.meth(obj);
    System.out.println("After  calling a="+obj.a+" b="+obj.b);
    }
}
```

O/P: Before calling a=4 b=5
After calling a=8 b=2

3.10. Recursion

A recursive method is a method that calls itself either directly or indirectly through another method. A method call itself is termed as Recursion.

Example: fact(n) = 1 ; if n=0
n*fact(n-1) ; n ≥ 1

// Write a program to print the factorial of a given number using recursion

```
class a
{
    double fact(double x)
    {
    if(x==0)
    return 1;
    else
    return x*fact(x-1);
    }
}
```

```
class recur
{
    public static void main(String[] args)
    {
    a obj=new a();
    int p=Integer.parseInt(args[0]);
    double t=obj.fact(p);
    System.out.println("facatorial="+t);
    }
}
```

// Write a program to print the first n Fibonacci sequence numbers using recursion
(Lab Program: 2)

```
class a
{
    int f(int x)
    {
    if(x==0||x==1)
    return x;
    else
    return (f(x-1)+f(x-2));
    }
}
class fibo
{
    public static void main(String[] args)
    {
    a obj=new a();
    int p=Integer.parseInt(args[0]);
```



```
    for(int i=1;i<=p;i++)
    {
    int t=obj.f(i);
    System.out.println("    "+t);
    }
    }
}
```

// Write a program to print the sum of 'n' numbers using the recursive formula

$$\text{sum}(n) = 0 \quad ; \text{if } n = 0$$
$$n + \text{sum}(n-1) \quad ; \text{if } n \geq 1$$

3.11 Random Number Generation

The elements of change can be introduced through the random method. Consider the following statement

double value = Math.random( );

The random( ) method generates a double value from 0.0 to 1.0(not included). The range of values produced directly by random is often different than what is needed in a specific application.

Example: A program that stimulates rolling six-sided die would required random integers in the range (1,6)

The general formula for this type of problems using random method is a+(int)(Math.random( )*b). Where 'a' is a shifting value that is equal to the fist number in the range of integers and 'b' is the scaling factor that is equal to the width of range of the integer.

Example:

```
class random
{
    public static void main(String[] args)
    {
     for(int i=1;i<=20;i++)
     {
     int s=1+(int)(Math.random()*6);
     System.out.print(s);
```

```
        if(i%5==0)
        System.out.print("\n");
        }
    }
}
```

3.12. Recursion Vs Iteration

Recursion	Iteration
1. Recursion uses a selection strucutre like if, if-else etc.,	1. Iteration uses a repetitive structure like while, do-while etc.,
2. Recursion terminates when base case is recognized	2. iteration terminates when the loop continuation condition fails
3. It can occur infinitely	3. It can occur infinitely
4. Extra memory is required	4. Extra memory is not required

3.13. Nested Class and Inner Classes

It is possible to define a class within another class such classes are known as nested classes. A nested class has access to all of the variables and methods of its outer class. But the outer class doesn't access the variables and methods of nested class. Since the scope of a nested class is bounded by the scope of its enclosing class.

There are two types of nested classes: static and non-static

A static nested class is a class started with 'static' modifier. Due to the static it must access the members of its enclosing class through an object. Direct access cannot be possible.

A non-static nested class is a class as ordinary representation of class. The most important type of the non-static class is "Inner class". The inner class has access to all of the variables and methods of its outer class. Thus, inner class scope is fully within the scope of its outer class.

Example:

```
class outer
{
    int out_x=34;
    void test()
    {
    inner in=new inner();
    in.display();
    }
    class inner
    {
    void display()
    {
    System.out.println("value="+out_x);
    }
    }
}
class test
{
    public static void main(String[] args)
    {
    outer y=new outer();
    y.test();
    }
}
```

3.14. Strings

A string is a collection of characters. Java implements strings as object type of String. Java provides three types of string handling classes.

1. String Class
2. StringBuffer Class
3. StringTokenizer Class

String and StringBuffer classes are defined in java.lang pacakage. StringTokenizer are defined in java.util package.

3.14.1. String Class:

A string is an object of class String. String literals or constants written as a sequence of characters in double quotes. A string may be assigned in declaration to string reference as

String color = "blue"

It initializes string reference color to refer the anonymous string object blue.

Note: 1. Once we create a string object, we cannot change the contents of string instance.

2. A variable declared as a string reference can be changed at any time.

String Constructors: String class provides several types of constructors. They are

a) String( ): It creates an empty string with no characters in it.

b) String(String): It takes one String as a parameter.

c) String(StringBuffer): Allocates a new string that contains the sequence of characters currently contained in the string buffer argument.

d) String(byte[] bytes): Construct a new String by converting the specified array of bytes using the platform's default character encoding.

e) String(bytes byte[], int startindex, int numberofcharacters): Allocates new string positioned with the startindex by counting the number of charactes.

f) String(char chars[]): Creates a string initialized by an array of characters.

g) String(char chars[], int startindex, int numberofcharacters): Creates a string initialized by an array of characters with the startindex by counting the number of charactes.

Methods:

1. **int length( ):** It returns the number of characters in a given string.

Example:

```
String s1=new String("hello");
System.out.println(s1.length());
class len
{
    public static void main(String[] args)
    {
    String s1=new String("hello");
    System.out.println(s1.length());
    }
}
```


2. **char charAt(int where):** It returns the character at the specified location. "where" must be a nonnegative specified location within the string. The first element of a string is considered to be at position "zero".

Example:

```
class pos
{
    public static void main(String[] args)
    {
    String s1=new String("hello java");
    char c=s1.charAt(7);
    System.out.println("character="+c);
    }
}
```

3. **void getChars(int sourcestart, int sourceend, char target[], int targetstart):** It extracts more than one character at a time and copy the characters of the string into a character array.

The first argument is the starting index from which characters are copied in the string. The second argument is the index that is one part of the last character to be copied from the string. The third argument is the array into which the characters are copied. The last argument is the starting index where the copied characters are placed in the character array.

Example:

```
class get
{
    public static void main(String[] args)
    {
    String s1="java programming language";
    char c[]=new char[10];
    s1.getChars(3,8,c,0);
    for(int i=0;i<(8-3);i++)
    System.out.print(c[i]);
    }
}
```

O/P: a prog

4. **boolean equals( ):** It compares the two strings and returns true if the strings contain the same characters in the same order, and false otherwise. The comparison is case-sensitive.

Syntax: boolean equals(Object string);

Example:

```
class equal
{
    public static void main(String[] args)
    {
        String s1=new String("india");
        String s2=s1;
        if(s1.equals(s2))
        System.out.println("Strings are Equal");
        else
        System.out.println("Strings are not Equal");
    }
}
```

5. **boolean equalsIgnoreCase( ):** It compares the two strings and returns true if the strings contain the same characters in the same order, and false otherwise by ignoring the case characters.

Syntax: boolean equalsIgnoreCase(String str);

Example:

```
class equall
{
    public static void main(String[] args)
    {
        String s1=new String("india");
        String s2=new String("InDia");
```



```
        if(s1.equalsIgnoreCase(s2))
        System.out.println("Strings are Equal");
        else
        System.out.println("Strings are not Equal");
    }
}
```

6. **int compareTo(String str):** It returns '0' if the two strings are equal. 'Negative number' if the invoking string is less than str and 'postivite number' if the invoking string is greater than str that is passed as an argument. The comparision is case-sensitive.

Example:

```
class comp
{
    public static void main(String[] args)
    {
        String s1=new String("hello");
        String s2=new String("india");
        if((s1.compareTo(s2))>0)
        System.out.println("S1 is greater than s2");
        else if((s1.compareTo(s2))<0)
        System.out.println("S2 is greater than s1");
        else
        System.out.println("Two stings are equal");
    }
}
```

O/P: S2 is greater than s1

If we want to ignore case differences when comparing two strings, we use the syntax of the form: int compareToIgnoreCase(String str)

This method was added by Java 2.

7. **regionMatches( ):** It compares a specific region inside a string with another specific region in another string. It follows two methods.

   1. boolean regionMatches(int startIndex, String str2, int str2StartIndex, int numchars)
   2. boolean regionMatches(boolean ignoreCase, int startIndex, String str2, int str2StartIndex, int numchars)

The first argument is the starting index of the string that invokes the method. The second argument is a comparison string. The third argument is starting index in the comparison string. The last argument is the number of characters to compare between the two strings. IgnoreCase is true the method ignores the case of the objects being compared.

Example 1:

```
class rmatch1
{
    public static void main(String[] args)
    {
      String s1=new String("happy birthday");
      String s2=new String("Happy birthday");
      if(s1.regionMatches(0,s2,0,5))
      System.out.println("Match");
      else
      System.out.println("Not Match");
    }
}
```

Example 2:

```
class rmatch1
{
    public static void main(String[] args)
    {
      String s1=new String("happy birthday");
      String s2=new String("Happy birthday");
      if(s1.regionMatches(true,0,s2,0,5))
      System.out.println("Match");
      else
      System.out.println("Not Match");
    }
}
```


8. **int indexOf(int ch):**

   int indexOf(String str):

   int indexOf(char ch): It search for the first occurrence of a character. If the character is found the index of that characters in the string is returned. Otherwise, it returns '-1'.

Example:

```
class index
{
    public static void main(String[] args)
    {
    String letter=new String("Java programming language");
    System.out.println(letter.indexOf('v'));                //O/P: 2
    System.out.println(letter.indexOf("programming"));      //O/P: 5
    System.out.println(letter.indexOf('A'));                //O/P: -1
    }
}
```

9. **int indexOf(int ch, int startIndex):**

   int indexOf(String ch, int startIndex):

   int indexOf(char ch, int startIndex): It searches the occurrence of character from the startIndex argument.

Example:

```
class indexof
{
    public static void main(String[] args)
    {
    String letter=new String("Java programming language");
    System.out.println(letter.indexOf('m',2));
    //O/P:11
    }
}
```

10. **int lastIndexOf(int ch):**

int lastIndexOf(char ch):

int lastIndexOf(String ch): It searches the last occurrence of the character. The search is performed from the end of the string towards the beginning of the string. If the character is found the index of that character in the string is return. Otherwise, it returns '-1'.

Example:

```
class lindexof
{
    public static void main(String[] args)
    {
    String letter=new String("Java programming language");
    System.out.println(letter.lastIndexOf('n'));          //O/P: 19
    System.out.println(letter.lastIndexOf("language"));   //O/P: 17
    }
}
```

11. **int lastIndexOf(int ch, int startIndex):**

int lastIndexOf(String ch, int startIndex):

int lastIndexOf(char ch, int startIndex): It searches the last occurrence of the character starting with the startIndex to search the string from ending to the beginning.

Example:

```
class lindexof
{
    public static void main(String[] args)
    {
    String letter=new String("Java programming language");
    System.out.println(letter.lastIndexOf('n',11));       //O/P: -1
    }
}
```


12. **String substring(int startIndex):** It returns a copy of the sub string that begins that startIndex and return to the end of the invoking string.

Example:

```
class sub
{
    public static void main(String[] args)
    {
        String s=new String("Java programming language");
        System.out.println(s.substring(8));
    }
}
```

O/P: gramming language

13. **String substring(int startIndex, int endIndex):** It returns all the characters from the beginning index upto but not including the ending index.

Example:

```
class s13
{
    public static void main(String[] args)
    {
    String s=new String("program");
    System.out.println(s.substring(2,5));      //O/P: ogr
    }
}
```

14. **boolean startsWith(String str):**

boolean endsWith(String str): The startsWith( ) method determines whether a given string begins with a specified string or not. Conversely, endsWith( ) method determines whether string ends with a specified string or not.

Example:

```
class s14
{
    public static void main(String[] args)
    {
    String s=new String("Java is a program");
    if(s.startsWith("is"))
    System.out.println("Match");
```

```
        else
        System.out.println("Not Match");
        }
}
//O/P:Not Match
```

15. **String concat(String str):** It concatenates two string objects.

Example:

```
class s15
{
        public static void main(String[] args)
        {
        String s=new String("Java ");
        String s1=s.concat("program");
        System.out.println(s1);
        }
}
```

O/P: Java program

16. **String replace(char original, char replacement):** It replace all characters of one character in the invoking string with another character.

Example:

```
class s16
{
        public static void main(String[] args)
        {
        String s="Hello";
        System.out.println(s.replace('l','w'));      //O/P:  Hewwo
        }
}
```


17. **String toLowerCase():** It converts all uppercase characters into lowercase characters of a string.

Example:

```
class s17
{
        public static void main(String[] args)
        {
        String s="HELLO";
        System.out.println(s.toLowerCase());          //O/P:  hello
        }
}
```

18. **String toUpperCase():** It converts all lowercase characters into uppercase characters of a string.

Example:

```
class s17
{
        public static void main(String[] args)
        {
        String s="hello";
        System.out.println(s.toUpperCase());     //O/P:  HELLO
        }
}
```

19. **String trim( ):** It returns a copy of the invoking string from which any leading and lining white space has been removed.

Example:

```
class s18
{
        public static void main(String[] args)
        {
        String s="   hello";
        System.out.println(s.trim());
        }
}
```

20. **char[] toCharArray():** It returns a character array containing a copy of the characters in a string.

Example:

```
class s20
{
    public static void main(String[] args)
    {
    String s="hello";
    char ch[]=s.toCharArray();
    for(int i=0;i<ch.length;i++)
    System.out.println(ch[i]);
    }
}
```

21. **String toString():** It converts all objects into strings objects.

Example:

```
class s21
{
    public static void main(String[] args)
    {
    Integer i=new Integer(10);
    System.out.println(i.toString());          //O/P: 10
    }
}
```

LabProgram 5: Write a program to sort the list of names in ascending order.

```
class lab5
{
    public static void main(String[] args)
    {
    String ch[]={"india","hai","andhra"};
    String temp;
    for(int i=0;i<ch.length-1;i++)
    {
    for (int j=i+1;j<ch.length;j++)
```



```
    {
    if((ch[i].compareTo(ch[j]))>0)
    {
    temp=ch[i];
    ch[i]=ch[j];
    ch[j]=temp;
    }
    }
    }
    for(int i=0;i<ch.length;i++)
    System.out.println(ch[i]);
    }
}
```

3.14.2. StringBuffer Class:

Once a string object is created then its contents cannot be changed. But the StringBuffer class provides the feature of creating and manipulating dynamic string information i.e., mutable strings. Every StringBuffer class is capable for storing a number of characters specified by its capacity. If the capacity of StringBuffer exceeded the capacity is automatically expanded to accommodate the additional characters.

StringBuffer Constructors:

a) StringBuffer(): It is a default StringBuffer Constructor to create StringBuffer with no characters init and an initial capacity of 16 characters.

b) StringBuffer(String str): It takes a string argument to create a StringBuffer containing the characters of the string argument. The initial capacity is the number of arguments in the string argument + 16.

c) StringBuffer(int size): It takes an integer argument and creates a StringBuffer with no characters init. The initial capacity is specified by the integer.

Example:

```
class sb
{
    public static void main(String[] args)
    {
    StringBuffer s1,s2,s3;
    s1=new StringBuffer();
    s2=new StringBuffer("hello");
    s3=new StringBuffer(45);
    System.out.println(s1.capacity()+s2.capacity()+s3.capacity());   //O/P:
    82
    }
}
```

Methods:

1. **int length():** It returns the number of characters in a StringBuffer.

Example:

```
class sb1
{
    public static void main(String[] args)
    {
    StringBuffer s=new StringBuffer("hello");
    System.out.println(s.length());          //O/P:  5
    }
}
```

2. **int capacity():** It returns the number of characters that can be stored in a StringBuffer without allocating more memory.

Example:

```
class sb2
{
    public static void main(String[] args)
    {
    StringBuffer s=new StringBuffer("hello");
    System.out.println(s.capacity());        //O/P:  21
    }
}
```


3. **void ensureCapacity():** It is provided to allow the programmer to guarantee that a StirngBuffer has a minimum capacity where capacity specifies the size of the buffer.

Example:

```
class sb3
{
    public static void main(String[] args)
    {
    StringBuffer s=new StringBuffer("hello");
    s.ensureCapacity(75);
    System.out.println(s.capacity());        //O/P:  75
    }
}
```

4. **void setLength(int len):** It increase or decrease the length of StringBuffer.

Example:

```
class sb4
{
    public static void main(String[] args)
    {
    StringBuffer s=new StringBuffer("abcdefghijkl");
    s.setLength(5);
    System.out.println(s);                   //O/P:  abcde
    }
}
```

5. **char charAt(int where):** It returns a character at the specified where position.

Example:

```
class sb5
{
    public static void main(String[] args)
    {
    StringBuffer s=new StringBuffer("abcdefghijkl");
```

```
        System.out.println(s.charAt(7));
        //O/P:      h
        }
}
```

6. **void setCharAt(int where, char ch):** It takes an integer and a character argument and sets the character at the specified position of the character argument.

Example:

```
class sb6
{
    public static void main(String[] args)
    {
    StringBuffer s=new StringBuffer("abcdef");      //O/P: Hbcdef
    s.setCharAt(0,'H');
    System.out.println(s);
    }
}
```

7. **void getChars(int Sourcestart, int Sourceend, char Target[], int Targetstart):** It copies the sub string of a StringBuffer into character array. Sourcestart specifies the starting index from which characters should be copied in StringBuffer. Sourceend specifies the index one past the last character to be copied from the StringBuffer. Target specifies the character array into which the characters are to be copied. Targetstart specifies the starting location of the character array from where the first character should be placed.

Example:

```
class sb7
{
    public static void main(String[] args)
    {
    StringBuffer s=new StringBuffer("abc def");
    char ch[]=new char[s.length()];
    s.getChars(3,s.length(),ch,0);
    for(int i=0;i<ch.length;i++)
    System.out.println(ch[i]);
    //O/P:def
    }
}
```


8. **StringBuffer reverse():** It returns the reverse contents of the StringBuffer.

Example:

```
class sb8
{
    public static void main(String[] args)
    {
    StringBuffer s=new StringBuffer("abcdef");
    System.out.println(s.reverse());
    }
}
```

3.14.3. StringTokenizer class:

In programming languages, statements are divided into individual pieces like identifiers, keywords, and operators etc., called Tokens. Java provides StringTokenizer class that breaks a string into its component tokens. This class is available in java.util package. Tokens are separator from one another by delimiters (special operators), white space characters such as blank, tab, new line and carriage return.

StringTokenizer Constructors:

a) StirngTokenizer(Stirng str)

b) StirngTokenizer(Stirng str, String delimiters)

c) StirngTokenizer(Stirng str, String delimiters, boolean delimiterAsToken)

In all versions str is a string that is tokenized.

In the first version, the default delimiters are used.

In the second version and third versions delimiters in string that specifies the delimiters.

In the third version if "delimiterAsToken" is true then the delimiters also returned as tokens when the string is tokenized otherwise, the delimiters are not returned. Delimiters are not returned as tokens by the first two forms.

Methods:

1. **int countTokens():** Using the current set of delimiters the method determines the number of tokens left to be tokenized and return the result.

2. **boolean hasMoreTokens():** It returns true if one or more tokens remain in the string and returns false if there are none.

3. **String nextToken():** It returns the next token as string.

Example:

```
import java.util.*;
class st
{
    public static void main(String[] args)
    {
    String s=new String("abc def gh");
    StringTokenizer t=new StringTokenizer(s);
    System.out.println(t.countTokens());
    while(t.hasMoreTokens())
    System.out.println(t.nextToken());
    }
}
```

O/P: 3

abc

def

gh

LabProgram 7: Write a program that reads a line of integers, then displays each integers and the sum of all the integers

```
import javax.swing.JOptionPane;
import java.util.*;
class lab7
{
    public static void main(String[] args)
    {
    //String s=new String("4$8$9$67");
    //StringTokenizer t=new StringTokenizer(s,"$");
    String s;
```



```
    s=JOptionPane.showInputDialog("Enter the string");
    StringTokenizer t=new StringTokenizer(s,"$");
    int a,sum=0;
    System.out.println(t.countTokens());
    while(t.hasMoreTokens())
    {
    a=Integer.parseInt(t.nextToken());
    System.out.println(a);
    sum=sum+a;
    }
    System.out.println("Total="+sum);
    }
}
```

LabProgram 4: Write a program to check whether a given string is palindrome or not.

```
import javax.swing.JOptionPane;
class lab4
{
    public static void main(String[] args)
    {
        String s;
        s=JOptionPane.showInputDialog("Enter a String");
        char ch[]=s.toCharArray();
        int i,j,count=0;
        for(i=0,j=ch.length-1;i<j;i++,j--)
        {
        if(ch[i]!=ch[j])
        {
        System.out.println("Not polindrome");
        count=1;
        break;
        }
        }
        if(count==0)
        System.out.println("Polindrome");
    }
}
```

3.15. Garbage Collection

Objects are dynamically allocated by using the new operator such objects are destroyed automatically and their memory released for later reallocation in Java. The technique that accomplishes this is called "Garbage Collection". Garbage collection works when no references to an object exist, the object is assumed to be no longer needed, and the memory occupied by the object can be reclaimed. Java run-time implementations will take various approaches to garbage collection.

finalize( ) method:

The garbage collector of an object called this method when the garbage collection determines that those are no more references to the object. It is also placed some actions inside the finalized method. The following points are related to finalize( ) method.

1. Every class has Object class as a super class
2. Every class in Java can have a finalize( ) method that returns resources to the system
3. A class should have only one finalize( ) method that takes no argument. Method finalize( ) is originally defined in class Object

The finalize( ) method has the following syntax

Syntax:
```
protected void finalize( )
{
        // finalization code
}
```

Example:
```
class Garbage extends Object
{
    int x,y;
    void setdata(int a,int b)
    {
    x=a;
    y=b;
    }
    void printdata()
    {
    System.out.println("x="+x+"  y="+y);
```



```
    }
    protected void finalize()
    {
    System.out.println("finalizer");
    }
}
class Mg
{
    public static void main(String[] args)
    {
        Garbage obj=new Garbage();
        Garbage obj1=new Garbage();
        obj.setdata(10,20);
        obj.printdata();
        obj1=null;
        System.gc();        // method to call garbage collector
        //obj1.setdata(2,3); //error
        obj.printdata();
    }
}
```

O/P: x=10 y=20
finalizer
x=10 y=20

3.16. EXAMPLE PROGRAMS

EXAMPLE :: 1

```
/* A program on instance variables.
  Call this file Demo1.java
*/
class B {
 double a;
 double b;
 double c;
}

// This class declares an object of type B
class Demo1 {
```

```
 public static void main(String args[]) {
  B bb = new B ();
  double v;

  // assign values to b instance variables
  bb.a= 10;
  bb.b = 20;
  bb.c = 15;

  // compute
  v=  bb.a *  bb.b *  bb.c;
  System.out.println("v value" + v);
 }
}
```

OUTPUT

```
C:\ >javac Demo1.java
C:\ >java Demo1

V value is 3000.0
```

EXAMPLE :: 2

```
// This program declares two B objects.
class B {
 double a;
 double b;
 double c;
}
class Demo2 {
 public static void main(String args[]) {
    B bb1 = new B ();

    B bb2 = new B ();
 double v;
```



```
// assign values to bb1's instance variables
  bb1.a= 10;
bb1.b = 20;
bb1.c = 15;
/* assign different values to bb2      instance variables */
bb2.a= 1;
bb2.b = 2;
bb2.c = 1;
// compute v firstbb1
v=  bb1.a *  bb1.b *  bb1.c;

System.out.println("V is " + v);

// compute volume of second bb2
  v=  bb2.a *  bb2.b *  bb2.c;

System.out.println("V is " + v);
 }
}
/* Here, B  uses a constructor to initialize */
class B {
 double a;
 double b;
 double c;

 // This is the constructor for B.
 B() {
  System.out.println("Constructing B");
  a = 10;
  b = 10;
  c = 10;
 }
```

```
  // compute and return v
  double v() {
   return a *b *c;
  }
}.

class Demo6 {
 public static void main(String args[]) {
  // declare, allocate, and initialize Box objects
B bb1 = new B ();
     B bb2 = new B ();

  double v;

  // get v of first B
  v = bb1.v();
  System.out.println("V is " + v);

  // get v of second B
  v = bb2.v();
  System.out.println("V is " + v);
 }
}
```

OUTPUT

```
C:\ >javac Demo2.java
C:\ >java Demo2
V value is 3000.0
V is 3.0
```


EXAMPLE ::3

```
// This program includes a method inside the B class.
class B {
 double a;
 double b;
 double c;

 // display v of a B
 void v () {
  System.out.print("V is ");
  System.out.println(a* b * c);
 }
}

class Demo3 {
 public static void main(String args[]) {
     B bb1 = new B ();
     B bb2 = new B ();

  // assign values to bb1s instance variables
    bb1.a= 10;
  bb1.b = 20;
  bb1.c = 10;

  /* assign different values to bb2
    instance variables */
    bb2.a= 1;
  bb2.b = 2;
  bb2.c = 1;

  // display vof first B
  bb1.v();

  // display v of second B
  bb2.v();
 }
}
```

OUTPUT

```
C:\ >javac Demo3.java
C:\ >java Demo3
V is 2000.0
V is 2.0
```

EXAMPLE :: 4

```
// Now, v () returns the v value of a B.

class B {
 double a;
 double b;
 double c;

 // compute and return v
 double v () {
  return a * b * c;
 }
}

class Demo4 {
 public static void main(String args[]) {
     B bb1 = new B ();
     B bb2 = new B ();

 double v;

 // assign values to bb1's instance variables
    bb1.a= 10;
 bb1.b = 20;
 bb1.c = 10;

 /* assign different values to bb2
```



```
   instance variables */
       bb2.a= 1;
   bb2.b = 2;
   bb2.c = 3;

   // get v of first B
   v = bb1.v();
   System.out.println("V is " + v);

   // get v of second B
   v = bb2.v();
   System.out.println("V is " + v);
  }
}
```

OUTPUT

```
C:\ >javac Demo4.java
C:\ >java Demo4
V is 2000.o
V.is 6.0
```

EXAMPLE :: 5

```
// This program uses a parameterized method.

class B {
  double a;
  double b;
  double c;

  // compute and return v
  double v () {
   return a * b * c;
  }
```

```
  // sets dimensions of B
  Void D(double w, double h, double d) {
   a = w;
   b = h;
   c = d;
  }
}

class Demo5 {
 public static void main(String args[]) {
  B bb1 = new B ();
     B bb2 = new B ();

  double v;

  // initialize each B
  bb1.D(10, 20, 15);
  bb2.D(3, 6, 9);

  // get v of first B
  v = bb1.v();
  System.out.println("V is " + v);

  // get v of second B
  v = bb2.v();
  System.out.println("V is " + v);
 }
}
```

OUTPUT

```
C:\ >javac Demo5.java
C:\ >java Demo5

V is 30000.0
V is 162.0
```


EXAMPLE :: 6

```
/* Here, B uses a constructor to initialize */
class B {
 double a;
 double b;
 double c;

 // This is the constructor for B.
 B() {
  System.out.println("Constructing B");
 a = 10;
 b = 10;
 c = 10;
 }

 // compute and return v
 double v() {
  return a *b *c;
 }
}

class Demo6 {
 public static void main(String args[]) {
  // declare, allocate, and initialize Box objects
 B bb1 = new B ();
     B bb2 = new B ();

  double v;

  // get v of first B
  v = bb1.v();
  System.out.println("V is " + v);

  // get v of second B
  v = bb2.v();
  System.out.println("V is " + v);
 }
}
```

OUTPUT

```
C:\ >javac Demo6.java
C:\ >java Demo6

Constructing B
Constructing B
V is 1000.0
V is 1000.0
```

EXAMPLE :: 7

```
/* Here, B uses a parameterized constructor to  initialize */
class B {
 double a;
 double b;
 double c;

 // This is the constructor for B.
 B(double w, double h, double d) {
  a = w;
  b = h;
  c = d;
 }

 // compute and return v
 double v() {
  return a * b * c;
 }
}
```



```
class Demo7 {
public static void main(String args[]) {
 // declare, allocate, and initialize B objects
 B bb1 = new B(10, 20, 15);
 B bb2 = new B(3, 6, 9);

 double v;

 // get v of first B
 v = bb1.v();
 System.out.println("Vis " + v);

 // get v of second B
 v = bb2.v();
 System.out.println("V is " + v);
 }
}
```

OUTPUT

```
C:\ >javac Demo7.java
C:\ >java Demo7

V is 3000.0
V is 162.0
```

EXAMPLE :: 8

```
// This class defines an integer stack that can hold 10 values.
class Stack {
 int stck[] = new int[10];
 int tos;

 // Initialize top-of-stack
 Stack() {
  tos = -1;
 }
```

```
  // Push an item onto the stack
  void push(int item) {
   if(tos==9)
    System.out.println("Stack is full.");
   else
    stck[++tos] = item;
  }

  // Pop an item from the stack
  int pop() {
   if(tos < 0) {
    System.out.println("Stack underflow.");
    return 0;
   }
   else
    return stck[tos--];
  }
}

class TestStackDemo {
  public static void main(String args[]) {
   Stack st1 = new Stack();
   Stack st2 = new Stack();

   // push some numbers onto the stack
   for(int i=0; i<10; i++) st1.push(i);
   for(int i=10; i<20; i++) st2.push(i);

   // pop those numbers off the stack
   System.out.println("Stack in mystack1:");
   for(int i=0; i<10; i++)
     System.out.println(st1.pop());
```



```
   System.out.println("Stack in mystack2:");
   for(int i=0; i<10; i++)
     System.out.println(st2.pop());
  }
}
```

OUTPUT

```
C:\ >javac TestStackDemo.java
C:\ >java TestStackDemo

Stack in my stack1:
9
8
7
6
5
4
3
2
1
0
Stack in my stack2:
19
18
17
16
15
14
13
12
11
10
```

EXAMPLE :: 9

```
/* This program demonstrates the difference between public and private */
class Access {
 int a; // default access
 public int b; // public access
 private int c; // private access

 // methods to access c
 void setc(int i) { // set c's value
  c = i;
 }
 int getc() { // get c's value
  return c;
 }
}

class AccessTestdemo {
 public static void main(String args[]) {
  Access ob = new Access();

  // These are OK, a and b may be accessed directly
  ob.a = 10;
  ob.b = 20;

  // This is not OK and will cause an error
// ob.c = 100; // Error!

  // You must access c through its methods
  ob.setc(100); // OK

  System.out.println("a, b, and c: " + ob.a + " " +
                ob.b + " " + ob.getc());
 }
}
```


OUTPUT

```
C:\ >javac AccessTestDemo.java
C:\ >java AccessTestDemo

a,b and c:10  20  100
```

EXAMPLE :: 10

```
// Objects are passed by reference.

class TestArg {
 int a, b;

 Test(int i, int j) {
  a = i;
  b = j;
 }

 // pass an object
 void meth(Test o) {
  o.a *= 2;
  o.b /= 2;
 }
}

class CallByRefDemo {
 public static void main(String args[]) {
  TestArg ob = new TestArg(10, 20);
    System.out.println("ob.a and ob.b before call: " +
                ob.a + " " + ob.b);
  ob.meth(ob);
  System.out.println("ob.a and ob.b after call: " +
                ob.a + " " + ob.b);
 }}
```

OUTPUT

```
C:\ >javac CallByRefDemo.java
C:\ >java CallByRefDemo

ob.a and ob.b beforecall: 10 20
Ob.a and ob.b aftercall: 20 10
```

EXAMPLE :: 11

```
// Simple Types are passed by value.
class CallByValue{
 void meth(int i, int j) {
  i *= 2;
  j /= 2;
 }
}

class CallByValueDemo {
 public static void main(String args[]) {
  CallByValue ob = new CallByValue ();
  int a = 150, b = 200;

  System.out.println("a and b before call: " +
                a + " " + b);

  ob.meth(a, b);
  System.out.println("a and b after call: " +
                a + " " + b);
 }
}
```

OUTPUT

```
C:\ >javac InnerClassDemo2.java
C:\ >java InnerclassDemo2
a and b before call:150 200
A and b after call:200 150
```


EXAMPLE :: 12

```
// Display all command line arguments.
class CommandLineDemo {
 public static void main(String args[]) {
  for(int i=0; i<args.length; i++)
   System.out.println("args[" + i + "]: " +
                 args[i]);
 }
}
```

OUTPUT

```
C:\ >javac CallByValueDemo.java
C:\ >java CallByValueDemo
```

EXAMPLE :: 13

```
// A simple example of recursion.
class Fact {
 // this is a recusive function
 int fact(int n) {
  int result;

  if(n==1) return 1;
  result = fact(n-1) * n;
  return result;
 }
}

class RecursionDemo {
 public static void main(String args[]) {
  Fact f = new Fact();
```

```
    System.out.println("Factorial of 3 is " + f.fact(3));
    System.out.println("Factorial of 4 is " + f.fact(4));
    System.out.println("Factorial of 5 is " + f.fact(5));
  }
}
// Demonstrate method overloading.
class OverloadDemo {
  void test() {
    System.out.println("No parameters");
  }

  // Overload test for one integer parameter.
  void test(int a1) {
    System.out.println("a1: " + a1);
  }

  // Overload test for two integer parameters.
  void test(int a, int b) {
    System.out.println("a1 and b1: " + a 1+ " " + b1);
  }

  // overload test for a double parameter
  double test(double a1) {
    System.out.println("double a: " + a1);
    return a1*a1;
  }
}

class Overload {
  public static void main(String args[]) {
    OverloadDemo ob = new OverloadDemo();
    double result;

    // call all versions of test()
    ob.test();
```



```
    ob.test(10);
    ob.test(10, 20);
    result = ob.test(123.2);
    System.out.println("Result of ob.test(123.2): " + result);
  }
}
```

OUTPUT

```
C:\ >javac RecursionDemo.java
C:\ >java RecursionDemo

Factorial of 3 is 6
Factorial of 4 is 24
Factorial of  5 is 120
```

EXAMPLE :: 14

```
// Demonstrate an inner class.
class Out {
  int outer_x = 10;

  void test() {
    Inn inner = new Inn();
    inner.display();
  }

  // this is an innner class
  class Inn {
    void display() {
      System.out.println("display: outer_x = " + outer_x);
    }
  }
}

class InnerClassDemo {
  public static void main(String args[]) {
```

```
    Out outer = new Out();
    outer.test();
  }
}
```

OUTPUT

```
C:\ >javac InnerClassDemo.java
C:\ >java InnerClassDemo

Display:outer_x=10
```

EXAMPLE :: 15

```
// This program will not compile.
class Out {
 int outer_x = 1000;

  void test() {
   Inner inner = new Inner();
   inner.display();
  }

  // this is an innner class
  class Inn {
   int y = 100; // y is local to Inner
   void display() {
    System.out.println("display: outer_x = " + outer_x);
   }
  }

 void showy() {
  System.out.println(y); // error, y not known here!
 }
}
```



```
class InnerClassDemo1 {
 public static void main(String args[]) {
  Out outer = new Out();
  outer.test();
 }
}
```

OUTPUT

```
C:\ >javac InnerClassDemo1.java
C:\ >java InnerclassDemo1

Display:outer_x=1000
Error,y not known here!
```

EXAMPLE :: 16

```
// Define an inner class within a for loop.
class Out {
 int outer_x = 10;

 void test() {
  for(int i=0; i<10; i++) {
   class Inn {
    void display() {
     System.out.println("display: outer_x = " + outer_x);
    }
   }
   Inn inner = new Inn();
   inner.display();
  }
 }
}

class InnerClassDemo2 {
 public static void main(String args[]) {
  Out outer = new Out();
  outer.test();
 }
}
```

OUTPUT

```
C:\ >javac InnerClassDemo2.java
C:\ >java InnerclassDemo2


Display:over_x=10
Display:over_x=10
Display:over_x=10
Display:over_x=10
Display:over_x=10
Display:over_x=10
Display:over_x=10
Display:over_x=10
Display:over_x=10
Display:over_x=10
```

EXAMPLE :: 17

```
// This program demonstrates the length array member.
class LengthDemo {
 public static void main(String args[]) {
  int a1[] = new int[10];
  int a2[] = {3, 5, 7, 1, 8, 99, 44, -10};
  int a3[] = {4, 3, 2, 1};

  System.out.println("length of a1 is " + a1.length);
  System.out.println("length of a2 is " + a2.length);
  System.out.println("length of a3 is " + a3.length);
 }
}
```

OUTPUT

```
C:\ >javac LengthDemo.java
C:\ >java LengthDemo
```


EXAMPLE :: 18

```
// Demonstrate method overloading.
class OverloadDemo {
 void test() {
  System.out.println("No parameters");
 }

 // Overload test for one integer parameter.
 void test(int a1) {
  System.out.println("a1: " + a1);
 }

 // Overload test for two integer parameters.
 void test(int a, int b) {
  System.out.println("a1 and b1: " + a 1+ " " + b1);
 }

 // overload test for a double parameter
 double test(double a1) {
  System.out.println("double a: " + a1);
  return a1*a1;
 }
}

class Overload {
 public static void main(String args[]) {
  OverloadDemo ob = new OverloadDemo();
  double result;

  // call all versions of test()
```

```
    ob.test();
    ob.test(10);
    ob.test(10, 20);
    result = ob.test(123.2);
    System.out.println("Result of ob.test(123.2): " + result);
  }
}
```

OUTPUT

```
C:\ >javac InnerClassDemo2.java
C:\ >java InnerclassDemo2
```

EXAMPLE :: 19

```
// Automatic type conversions apply to overloading.
class OverloadDemo {
  void test() {
    System.out.println("No parameters");
  }

  // Overload test for two integer parameters.
  void test(int a1, int b1) {
    System.out.println("a1 and b1: " + a1 + " " + b1);
  }

  // overload test for a double parameter and return type
  void test(double a1) {
    System.out.println("Inside test(double) a1: " + a1);
  }
}

class Overload1 {
  public static void main(String args[]) {
```



```
    OverloadDemo ob = new OverloadDemo();
    int i = 88;

    ob.test();
    ob.test(10, 20);

    ob.test(i); // this will invoke test(double)
    ob.test(123.2); // this will invoke test(double)
  }
}
```

OUTPUT

```
C:\ >javac InnerClassDemo2.java
C:\ >java InnerclassDemo2
```

EXAMPLE :: 20

```
/* Here, B defines three constructors to initialize

class B {
  double a;
  double b;
  double c;
  // constructor used when all dimensions specified
  Box(double w, double h, double d) {
    a = w;
    b = h;
    c = d;
  }

  // constructor used when no dimensions specified
  B() {
    a = -1; // use -1 to indicate
```

```
    b = -1; // an uninitialized
    c = -1; // box
  }

  // constructor used when cube is created
  B(double l) {
    a = b = c = l;
  }

  // compute and return volume
  double v() {
    return a * b * c;
  }
}

class OverloadConsDemo {
  public static void main(String args[]) {
    // create B using the various constructors
    B aa = new B(10, 20, 15);
    B bb = new B();
    B cc = new B(7);

    double v;

    // get v of first B
    v = aa.v();
    System.out.println("V of aa  is " + v);

    // get v of second B
    v = bb.v();
    System.out.println("V of bb is " + v);

    // get volume of cube
    v = cc.v();
    System.out.println("V of cc is " + v);
  }
}
```


EXAMPLES ON STRING HANDLING FUNCTIONS

EXAMPLE :: 1

```
// Demonstrate toUpperCase() and toLowerCase().

class ChangeCaseDemo {
 public static void main(String args[])
 {
  String s = "This is a vijay.";

  System.out.println("Original: " + s);

  String uppr = s.toUpperCase();
  String lwr = s.toLowerCase();

  System.out.println("Uppercase: " + uppr);
  System.out.println("Lowercase: " + lwr);
 }
}
```

OUTPUT

```
C:\ >javac ChangeCaseDemo.java
C:\ >java ChangeCaseDemo


Original:This is Vijay
Uppercase:THIS IS VIJAY
Lowercase:this is vijay
```

EXAMPLE :: 2

```
// Using concatenation to prevent long lines.
class ConCatDemo {
 public static void main(String args[]) {
  String longStr = "This is object oriented programming " +
```

```
    "through java " +
    "by v.vijaybhaskar" ;

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

OUTPUT

```
C:\ >javac ConCatDemo.java
C:\ >java ComCatDemo

This is object oriented program through java by V.Vijaybhaskar
```

EXAMPLE :: 3

```
// Demonstrate delete() and deleteCharAt()
class deleteDemo {
 public static void main(String args[]) {
  StringBuffer sb = new StringBuffer("This is oops through java.");

   sb.delete(4, 9);
   System.out.println("After delete: " + sb);

   sb.deleteCharAt(0);
   System.out.println("After deleteCharAt: " + sb);
  }
}
```

OUTPUT

```
C:\ >javac deleteDemo.java
C:\ >java deleteDemo

After delete:thisops through java.
After deletechartAt:hisops through java.
```


EXAMPLE :: 4

```
// Demonstrate equals() and equalsIgnoreCase().
class equalsDemo {
 public static void main(String args[]) {
  String s1 = "Vijay";
  String s2 = "Vijay";
  String s3 = "Good-bye";
  String s4 = "VIJAY";
  System.out.println(s1 + " equals " + s2 + " -> " +
              s1.equals(s2));
  System.out.println(s1 + " equals " + s3 + " -> " +
              s1.equals(s3));
  System.out.println(s1 + " equals " + s4 + " -> " +
              s1.equals(s4));
  System.out.println(s1 + " equalsIgnoreCase " + s4 + " -> " +
              s1.equalsIgnoreCase(s4));
 }
}
```

OUTPUT

```
C:\ >javac equalsDemo.java
C:\ >java equalsDemo

Vijay equals Vijay->true
Vijay equals Good-bye->false
Vijay equals VIJAY->false
Vijay equals ignorecase VIJAY->true
```

EXAMPLE :: 5

```
// equals() vs ==
class EqualsNotEqualTo {
 public static void main(String args[]) {
  String s1 = "vijay";
```

```
    String s2 = new String(s1);

    System.out.println(s1 + " equals " + s2 + " -> " +
                s1.equals(s2));
    System.out.println(s1 + " == " + s2 + " -> " + (s1 == s2));
  }
}
```

OUTPUT

```
C:\ >javac EqualIsNotEqualTo.java
C:\ >java EqualsNotEqualTo

Vijay equals vijay ->true
Vijay==vijay->false
```

EXAMPLE :: 6

```
class getCharsDemo {
  public static void main(String args[]) {
    String s = "this is vijaybhaskar.";
    int start = 10;
    int end = 14;
    char buf[] = new char[end - start];

    s.getChars(start, end, buf, 0);
    System.out.println(buf);
  }
}
```

OUTPUT

```
C:\ >javac getCharsDemo.java
C:\ >java getCharDemo
jayb
```


EXAMPLE :: 7

```
// Demonstrate indexOf() and lastIndexOf().
class indexOfDemo {
 public static void main(String args[]) {
  String s = "this is object oriented programming through java " +
          "by v.vijaybhaskar";

  System.out.println(s);
  System.out.println("indexOf(t) = " +
                s.indexOf('t'));
  System.out.println("lastIndexOf(t) = " +
                s.lastIndexOf('t'));
  System.out.println("indexOf(the) = " +
                s.indexOf("the"));
  System.out.println("lastIndexOf(the) = " +
                s.lastIndexOf("the"));
  System.out.println("indexOf(t, 10) = " +
                s.indexOf('t', 10));
  System.out.println("lastIndexOf(t, 60) = " +
                s.lastIndexOf('t', 60));
  System.out.println("indexOf(the, 10) = " +
                s.indexOf("the", 10));
  System.out.println("lastIndexOf(the, 60) = " +
                s.lastIndexOf("the", 60));
 }
}
```

OUTPUT

```
C:\ >javac indexOfDemo.java
C:\ >java indexOfDemo

This is object oriented program through java by V.VijayBhaskar
Index<t>=0
Lastindex<t>==36
Indexof<the>=-1
```

```
Last index of<the>=-1
Indexof<t,10>=13
Last index of<t,60>=36
Indexof<the,10>=-1;
Last index of <the,60>=-1
```

EXAMPLE :: 8

```
// Demonstrate insert().
class insertDemo {
 public static void main(String args[]) {
  StringBuffer sb = new StringBuffer("I like!");

  sb.insert(2, "love ");
  System.out.println(sb);
 }
}
```

OUTPUT

```
C:\ >javac insertDemo.java
C:\ >java insertDemo

I Love Like!
```

EXAMPLE :: 9

```
// Construct one String from another.
class MakeString {
 public static void main(String args[]) {
  char c[] = {'V', 'i', 'j', 'a','y'};
  String s1 = new String(c);
  String s2 = new String(s1);

  System.out.println(s1);
  System.out.println(s2);
 }
}
```


OUTPUT

```
C:\ >javac MakeString.java
C:\ >java MakeString

'V','I','J','A','Y'
```

EXAMPLE :: 10

```
// Demonstrate replace()
class replaceDemo {
 public static void main(String args[]) {
  StringBuffer sb = new StringBuffer("This is vijay.");

  sb.replace(5, 7, "was");
  System.out.println("After replace: " + sb);
 }
}
```

OUTPUT

```
C:\ >javac replaceDemo.java
C:\ >java replaceDemo

After replace:this was vijay
```

EXAMPLE :: 11

```
// Using reverse() to reverse a StringBuffer.
class ReverseDemo {
 public static void main(String args[]) {
  StringBuffer s = new StringBuffer("java");
  System.out.println(s);
  s.reverse();
  System.out.println(s);
 }
}
```

OUTPUT

```
C:\ >javac ReverseDemo.java
C:\ >java ReverseDemo

Java
avaj
```

EXAMPLE :: 12

```
// Demonstrate charAt() and setCharAt().
class setCharAtDemo {
 public static void main(String args[]) {
  StringBuffer sb = new StringBuffer("vijay");
  System.out.println("buffer before = " + sb);
  System.out.println("charAt(1) before = " + sb.charAt(1));
   sb.setCharAt(1, 'i');
   sb.setLength(2);
   System.out.println("buffer after = " + sb);
   System.out.println("charAt(1) after = " + sb.charAt(1));
  }
 }
```

OUTPUT

```
C:\ >javac setCharAtDemo.java
C:\ >java setCharAtDemo

Buffer before=vijay
charAt<1>before=i
buffer after=vi
charAt<1>after=i
```

EXAMPLE :: 13

```
// A bubble sort for Strings.
class SortString {
 static String arr[] = {
  "vijay", "isaac", "manasa", "joshna"
 };
```



```
 public static void main(String args[]) {
  for(int j = 0; j < arr.length; j++) {
   for(int i = j + 1; i < arr.length; i++) {
    if(arr[i].compareTo(arr[j]) < 0) {
     String t = arr[j];
     arr[j] = arr[i];
     arr[i] = t;
    }
   }
   System.out.println(arr[j]);
  }
 }
}
```

OUTPUT

```
C:\ >javac setCharAtDemo.java
C:\ >java setCharAtDemo

Isaac
Joshna
Manasa
vijay
```

EXAMPLE :: 14

```
// StringBuffer length vs. capacity.
class StringBufferDemo {
 public static void main(String args[]) {
  StringBuffer sb = new StringBuffer("vijay");

  System.out.println("buffer = " + sb);
  System.out.println("length = " + sb.length());
  System.out.println("capacity = " + sb.capacity());
 }
}
```

OUTPUT

```
C:\ >javac ReverseDemo.java
C:\ >java ReverseDemo

Buffer=vijay
Length==5
Capacity=21
```

EXAMPLE :: 16

```
// Construct string from subset of char array.
class SubStringConsDemo {
  public static void main(String args[]) {
    byte ascii[] = {65, 66, 67, 68, 69, 70 };

    String s1 = new String(ascii);
    System.out.println(s1);

    String s2 = new String(ascii, 2, 3);
    System.out.println(s2);
  }
```

OUTPUT

```
C:\ >javac SubStringConsDemo.java
C:\ >java SubStringConsDemo
ABCDEF
CDE
```