INTRODUCTION

Java was conceived by James Gosling, Patrick Naughton, Chris Warth, Ed Frank, and Mike Sheridan at Sun Microsystems, Inc. in 1991. This language was initially called “Oak” but was renamed “Java” in 1995.

The original impetus for Java was not the Internet! Instead, the primary motivation was the need for a platform-independent (that is, architecture-neutral) language that could be used to create software to be embedded in various consumer electronic devices, such as microwave ovens and remote controls. As you can probably guess, many different types of CPUs are used as controllers.

The trouble with C and C++ (and most other languages) is that they are designed to be compiled for a specific target. Although it is possible to compile a C++ program for just about any type of CPU, to do so requires a full C++ compiler targeted for that CPU. The problem is that compilers are expensive and time-consuming to create. An easier— and more cost-efficient—solution was needed.

In an attempt to find such a solution, Gosling and others began work on a portable, platform-independent language that could be used to produce code that would run on a variety of CPUs under differing environments. This effort ultimately led to the creation of Java.

The key that allows Java to solve both the security and the portability problems just described is that the output of a Java compiler is not executable code. Rather, it is bytecode. Bytecode is a highly optimized set of instructions designed to be executed by the Java run-time system, which is called the Java Virtual Machine (JVM). That is, in its standard form, the JVM is an interpreter for bytecode

FEATURES OF JAVA

- ✓ simple
- ✓ secure
- ✓ portable
- ✓ Object Oriented
- ✓ Robust
- ✓ Multi-Threaded
- ✓ Architecture Neutral
- ✓ Interpreted
- ✓ High Performance
- ✓ Distributed
- ✓ Dynamic

Simple

Java was designed to be easy for the professional programmer to learn and use effectively. Assuming that you have some programming experience, you will not find Java hard to master. If you already understand the basic concepts of object-oriented programming, learning Java will be even easier.

Security

When you use a Java-compatible Web browser, you can safely download Java applets without fear of viral infection or malicious intent. Java achieves this protection by confining a Java program to the Java execution environment and not allowing it access to other parts of the computer.

Portability

As discussed earlier, many types of computers and operating systems are in use throughout the world—and many are connected to the Internet. For programs to be dynamically downloaded to all the various types of platforms connected to the Internet, some means of generating portable executable code is needed. As you will soon see, the same mechanism that helps ensure security also helps create portability. Indeed, Java's solution to these two problems is both elegant and efficient.

Object-Oriented

Java was not designed to be source-code compatible with any other language. This allowed the Java team the freedom to design with a blank slate. One outcome of this was a clean, usable, pragmatic approach to objects.

Robust

The programs must execute reliably in a variety of systems without any flaws and it should allow us to add or remove modules to the existing project without affecting the performance of the project. Java creates such environment so Java is said to be robust.

Multithreaded

Java was designed to meet the real-world requirement of creating interactive, networked programs. To accomplish this, Java supports multithreaded programming, which allows you to write programs that do many things simultaneously.

Architecture-Neutral

A central issue for the Java designers was that of code longevity and portability. One of the main problems facing programmers is that no guarantee exists that if you write a program today, it will run tomorrow—even on the same machine. Operating system upgrades, processor upgrades, and changes in core system resources can all combine to make a program malfunction. The Java designers made several hard decisions in the Java language and the Java Virtual Machine in an attempt to alter this situation. Their goal was "write once; run anywhere, anytime, forever." To a great extent, this goal was accomplished.

Interpreted and High Performance

Java enables the creation of cross-platform programs by compiling into an intermediate representation called Java bytecode. This code can be interpreted on any system that provides a Java Virtual Machine.

Distributed

Java is designed for the distributed environment of the Internet, because it handles TCP/IP protocols. Java provides *Remote Method Invocation* (*RMI*) to implement this. This feature brings an unparalleled level of abstraction to client/ server programming.

Dynamic

Java programs carry with them substantial amounts of run-time type information that is used to verify and resolve accesses to objects at run time. This makes it possible to dynamically link code in a safe and expedient manner.

JAVA VIRTUAL MACHINE (JVM)

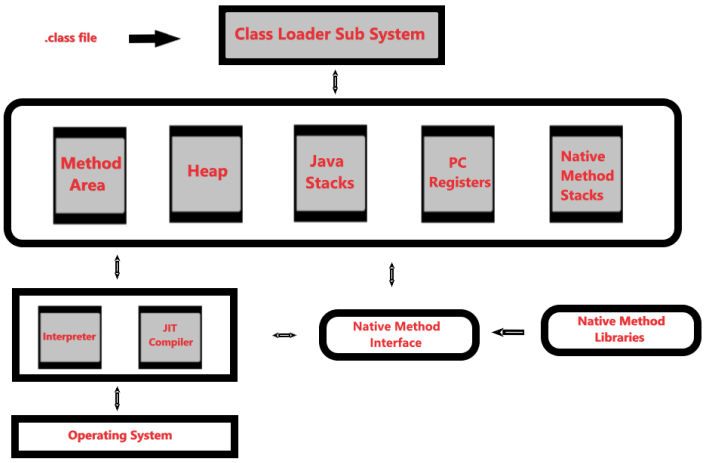


Fig: Java Virtual Machine

JVM is the heart of entire java execution process. It will take .class file and convert each byte code instruction into machine language instruction that can be executed by the processor.

After .java program is converted into .class file, it is loaded into jvm by using class loader subsystem. In this process it loads .class file into memory, checks for the errors, and then execute the program.

In this, memory is divided into 5 sub parts called runtime data areas. These areas are as follows.

Method Area : It stores the classes, variables, methods related code

Heap: In this area objects are created. Whenever jvm loads a class immediately a method & heap area are created on it

Java Stacks: Method code is stored on Method area. But while running a method it needs some memory to store the data and results. This memory is allotted on Java Stacks.

PC(Program Counter) Registers: It stores memory addresses of the instructions of the methods

Native Method Stacks: To store temporary results while executing native methods, this area will be used.

Execution Engine contains Interpreter and JIT Compiler which are responsible for converting the byte code instructions into machine code.

HOW TO WRITE COMMENTS IN JAVA

Generally we write the comments

- To describe about the purpose of a class / method / variable
- To temporarily suspend the execution of some lines in our code

Comments improve programs readability and understand ability.

We will write single line comments / multi line comments / documentation comments based on our need.

Single Line Comments : To comment a single line we will use this single line comment. It will be represented as follows.

```
// This is a single line comment
```

Multi Line Comments: To Comment multiple lines we will use /* and */ at starting and ending points.

```
/*  These are multiline Comments

        first statement

        second statement

*/
```

Documentation Comments:

We can write documentation comments which are useful in creating API. We can write documentation comments as /** and */ at starting and ending points.

```
/**

            This method returns two numbers addition as result

*/
```

HOW TO COMPILE & RUN OUR FIRST PROGRAM

Consider the below program

```
//this is my first java program
class FirstProgram
{
  public static void main(String ar[])
  {
                System.out.println("Welcome to Java class by Telugu Web Guru");
  }
}
```

In the above program please observe that all the statements are written within the class bounds only. Java is a purely object oriented language. So each and every line must be within the bounds of a class

One more important thing is each and every program's execution is starts with main method only and we must save the program with the class name only then only jvm can execute our program.

FirstProgram is just classname. You can give any name here. class is a keyword that is used to create a class.

In main method syntax each keyword's description is given below

public : jvm which stays external to our program need to enter into our class and run main method. So we must declare main method as public then only jvm can execute it.

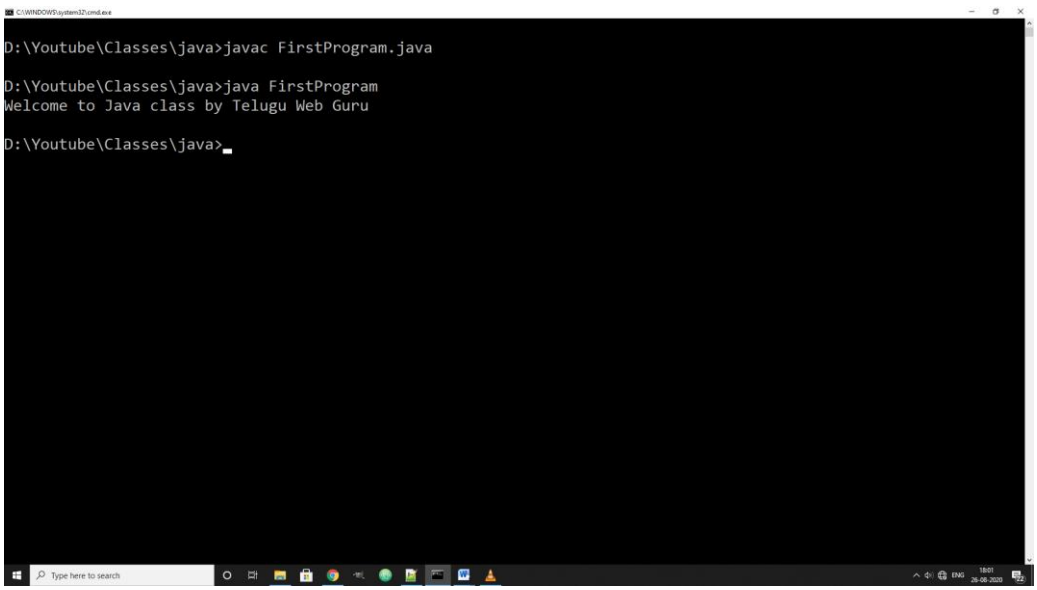
static : In java every class member (variables, methods) of a class must be accessed through objects. If you want to access a class member without creating object, then we must declare it as static.

void: Every method in java may or may not return a value. If it returns any value using return statement, then method must be declared with the corresponding datatype that matches to the value that it returns. If a method does not return any value then we must declare that method as void.

String ar[] : Observe the string array as parameter in main method. It is used to store user inputs if passed by user. Here ar is just array name we can give any name instead of it. But the mandatory thing here is it must be String array.

System.out.println : This is used to print any message to the screen as output. it is same as printf statement in C Language.

Output



As shown in above output screen every java program must be compiled before execute/run it.

To compile a java program javac command is used.

javac programname-with-extension

To Run / Execute Java Program java command is used.

java programname-without-extension

DATA TYPES

Java defines eight simple (or elemental) types of data: byte, short, int, long, char, float, double, and boolean.

These can be put in four groups.

- Integers : This group includes byte, short, int, and long, which are for whole-valued signed numbers.
- Floating point numbers : This group includes float and double, which represent numbers with fractional precision.
- Character : This group includes char, which represents symbols in a character set, like letters and numbers.
- Boolean : This group includes boolean, which is a special type for representing true/false values.

Integers

Java defines four integer types: byte, short, int, and long. All of these are signed, positive and negative values.

The width and ranges of these integer types vary widely, as shown in this table:

Name	Width	Range
byte	8	-128 to 127
short	16	-32,768 to 32,767
int	32	-2,147,483,648 to 2,147,483,647
long	64	-9,223,372,036,854,775,808 to 9,223,372,036,854,775,807

The smallest integer type is byte. width tells how much memory a variable of a datatype occupies. Range tells us the range of values that can be assigned to a variable. for example we can assign only -128 to 127 to a variable of type byte.

Example

```
byte a = 10;
int b = 257;
```

Floating-Point Types

Floating-point numbers, also known as real numbers, are used when evaluating expressions that require fractional precision. There are two kinds of floating-point types, float and double, which represent single- and double-precision numbers, respectively. Their width and ranges are shown here:

Name	Width in bits	Range
float	32	1.4e–045 to 3.4e+038
double	64	4.9e–324 to 1.8e+308

Single precision numbers represents only up to 7 bits (float) after decimal point whereas Double precision numbers can represent up to 15 bits after decimal point (double).

Example:

```
float a = 3.4f;   //floating values must be suffixed with the character f to indicate it is float
double b = 4.556;
```

Characters

In Java, the data type used to store characters is char. : char in Java is not the same as char in C or C++. In C/C++, char is an integer type that is 8 bits wide. This is not the case in Java. Instead, Java uses Unicode to represent characters. in Java char is a 16-bit type. The range of a char is 0 to 65,536.

Example:

```
char a = ' h ' ;
```

Booleans

Java has a simple type, called boolean, for logical values. It can have only one of two possible values, true or false.

Example:

```
boolean a = true;
```

Let's see more examples on data types with operators concept.

Variables:

Variable is the basic unit of storage in a Java program. These are memory location names which stores values. A variable is defined by the combination of an identifier, a type, and an optional initializer. In addition, all variables have a scope, which defines their visibility, and a lifetime.

A variable is declared with the following syntax.

```
datatype variable-name  = value
```

Example

```
int a = 10;
```

Here int is a datatype and a is an variable.

The Scope and Lifetime of Variables

A block defines a scope. Thus, each time you start a new block, you are creating a new scope. As you probably know from your previous programming experience, a scope determines what objects are visible to other parts of your program. It also determines the lifetime of those objects.

A variable declared inside a class, outside all the methods and blocks is an instance variable. The general scope of an instance class variable is throughout the class except in static methods. The lifetime of an instance variable is until the object stays in memory.

A variable declared in a class with static is a class variable. The general scope of a class variable is throughout the class. The lifetime of an instance variable is until end of the program or as long as the class is loaded in memory.

All other variables (which are not class/instance variables) are called local variables. The general scope of a local variable is within in its block in which it is declared. The lifetime of a local variable is until the control leaves the block.

OPERATORS IN JAVA

Operators are used to write programs that performs basic computations.

Arithmetic Operators

Arithmetic operators are used in mathematical expressions in the same way that they are used in algebra. The following table lists the arithmetic operators

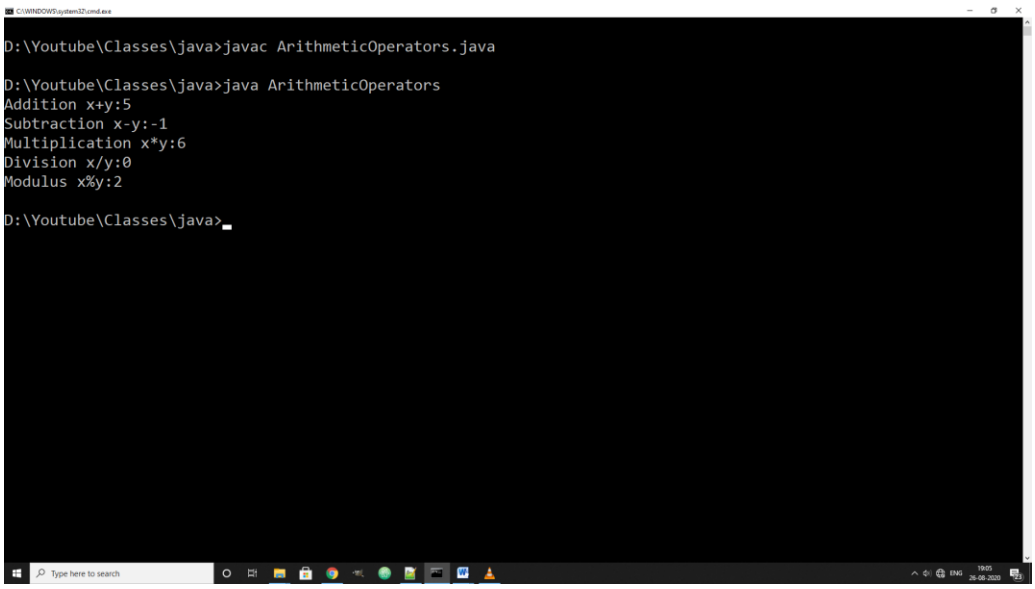
Operator	Result
+	Addition
-	Subtraction (also unary minus)
*	Multiplication
/	Division
%	Modulus
++	Increment
+=	Addition assignment
-=	Subtraction assignment
*=	Multiplication assignment
/=	Division assignment
%=	Modulus assignment
--	Decrement

The operands of the arithmetic operators must be of a numeric type.

Example

```
//program to demonstrate arithmetic operations in java

class ArithmeticOperators
{
  public static void main(String ar[])
  {
            byte x=2;
            byte y=3;

            System.out.println("Addition x+y:"+(x+y));
            System.out.println("Subtraction x-y:"+(x-y));
            System.out.println("Multiplication x*y:"+(x*y));
            System.out.println("Division x/y:"+(x/y));
            System.out.println("Modulus x%y:"+(x%y));


  }
}
```

Output



Unary Operators:

There are three unary operators : unary minus (-) , Increment Operator (++) , Decrement Operator (--).

Unary Minus (-)

It will convert the sign of the operand from positive to negative and vice versa.

Increment Operator (++)

Increment operator is used to increment the operand's value by one.

There are two types of Increment Operators :

Pre Increment: (++x)

It first increments the value and then the statement that contains pre increment will be executed.

Post Increment : (x++)

It first executes the statement and then it increments the value.

Decrement Operator (--)

Decrement operator is used to decrement the operand's value by one.

There are two types of Decrement Operators:

Pre Decrement: (--x)

It first decrements the value and then the statement that contains pre decrement will be executed.

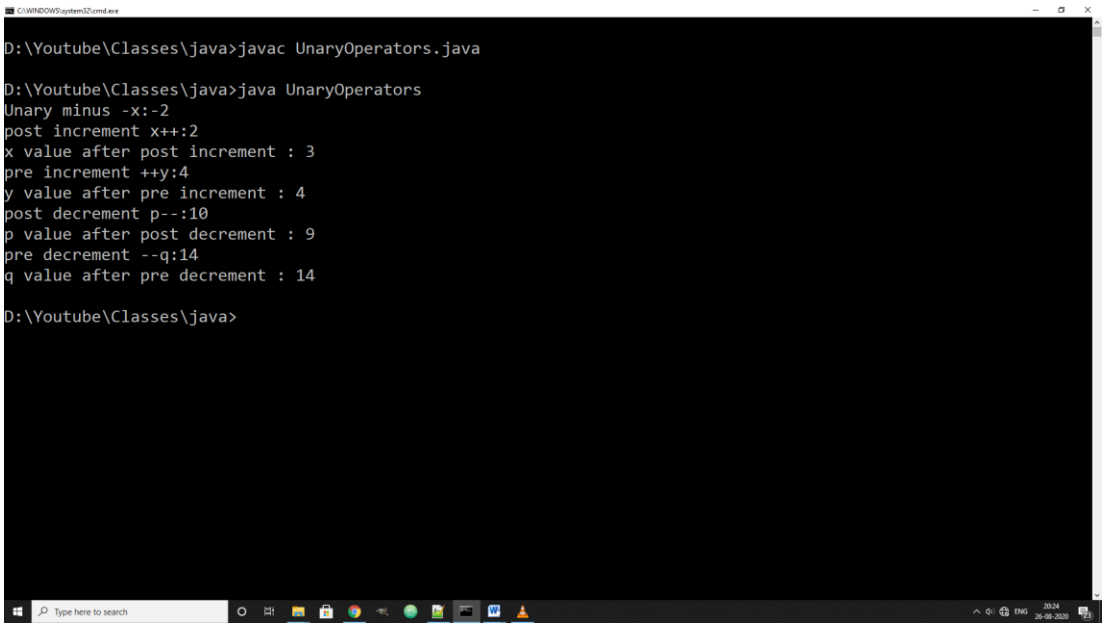
Post Decrement: (x--)

It first executes the statement and then it decrements the value.

Example Program:

```
//program to demonstrate unary operations in java

class UnaryOperators
{
 public static void main(String ar[])
 {
        byte x=2;
        byte y=3;
        byte p=10;
        byte q=15;

        System.out.println("Unary minus -x:"+(-x));

        System.out.println("post increment x++:"+(x++));
        System.out.println("x value after post increment : "+(x));

        System.out.println("pre increment ++y:"+(++y));
        System.out.println("y value after pre increment : "+(y));

        System.out.println("post decrement p--:"+(p--));
        System.out.println("p value after post decrement : "+(p));

        System.out.println("pre decrement --q:"+(--q));
        System.out.println("q value after pre decrement : "+(q));



 }
}
```

Output:



Assignment Operator (=) :

Assignment operator is used to assign values to the variables.

Example : int a = 2;

Relational Operators :

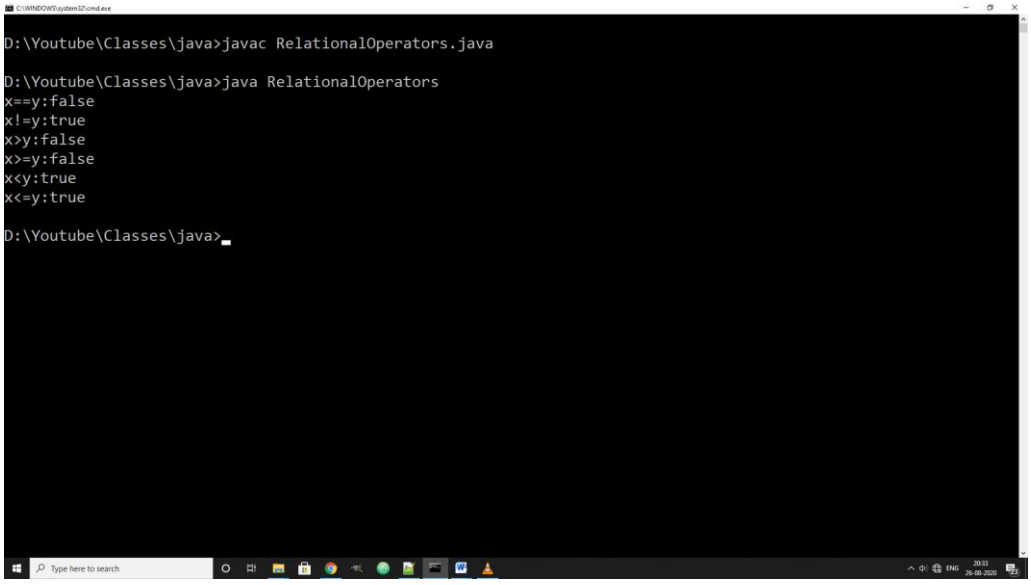
These are used to compare the values of two operands.

Operator	Meaning
==	Compares whether two operands are equal or not
!=	Compares whether two operands are not equal or not
>	Compares whether the first operand is greater than second operand or not
>=	Compares whether the first operand is greater than or equal to second one or not
<	Compares whether the first operand is less than second operand or not
<=	Compares whether the first operand is less than or equal to second operand or not

Example:

```
//program to demonstrate relational operations in java

class RelationalOperators
{
	public static void main(String ar[])
	{
		byte x=2;
		byte y=3;

		System.out.println("x==y:"+(x==y));
		System.out.println("x!=y:"+(x!=y));
		System.out.println("x>y:"+(x>y));
		System.out.println("x>=y:"+(x>=y));
		System.out.println("x<y:"+(x<y));
		System.out.println("x<=y:"+(x<=y));



	}
}
```

Output



Logical Operators :

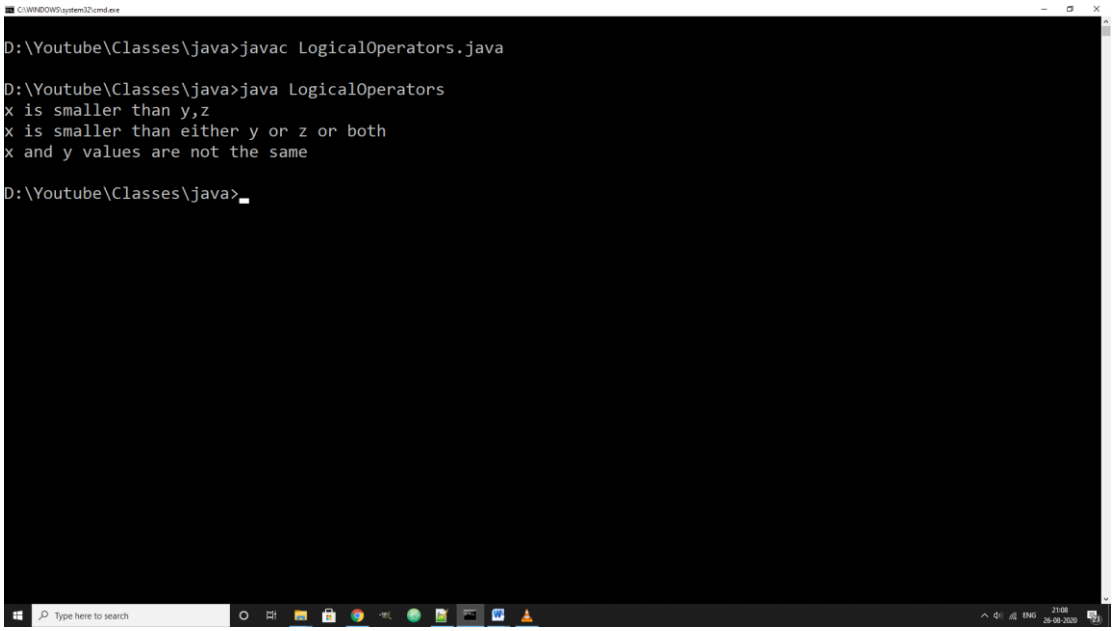
These are used to create compound statements by combining more than one condition

&&	-	Logical And	-	Returns true only if both conditions are true
||	-	Logical OR	-	Returns false only if both conditions are false
!	-	Logical Not	-	It makes any condition Negative (it checks for not true)

Example:

```
//program to demonstrate logical operations in java

class LogicalOperators
{
  public static void main(String ar[])
  {
            byte x=10;
            byte y=20;
            byte z=30;

            if(x<y && x<z)
                    System.out.println("x is smaller than y,z");


            if(x<y || x<z)
                     System.out.println("x is smaller than either y or z or both");

            if(!(x==y))
                     System.out.println("x and y values are not the same");


  }
}
```

Output



Boolean Operators :

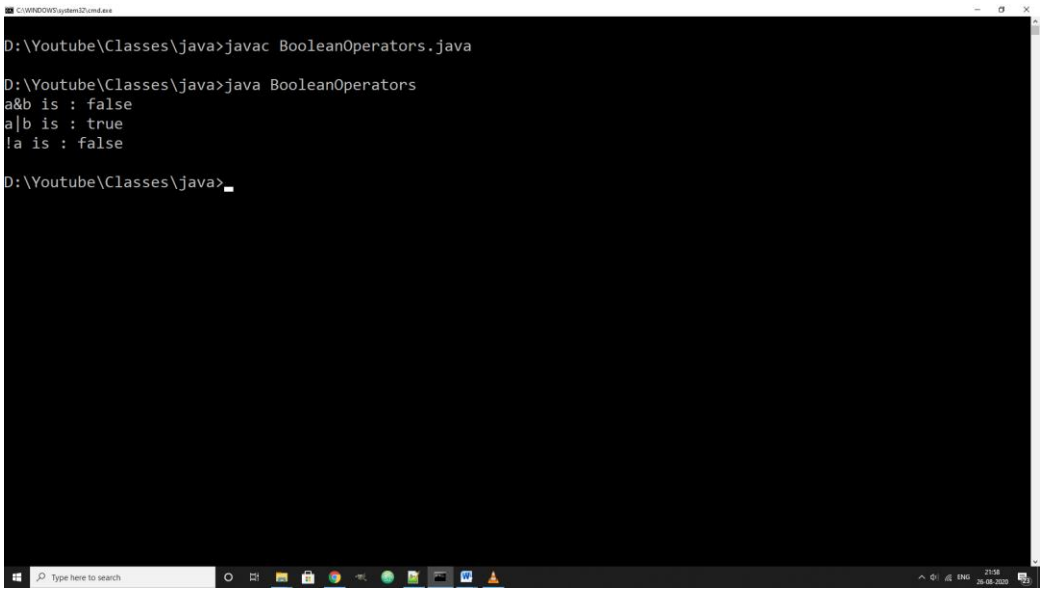
Boolean operators can be applied on boolean type operands only.

&	-	And	-	Returns true only if both conditions are true
|	-	OR	-	Returns false only if both conditions are false
!	-	Not	-	true becomes false and false becomes true

Example:

```
//program to demonstrate boolean operations in java

class BooleanOperators
{
	public static void main(String ar[])
	{
		boolean a=true;
		boolean b=false;

		System.out.println("a&b is : "+(a&b));
		System.out.println("a|b is : "+(a|b));
		System.out.println("!a is : "+(!a));

	}
}
```

Output:



Bitwise Operators

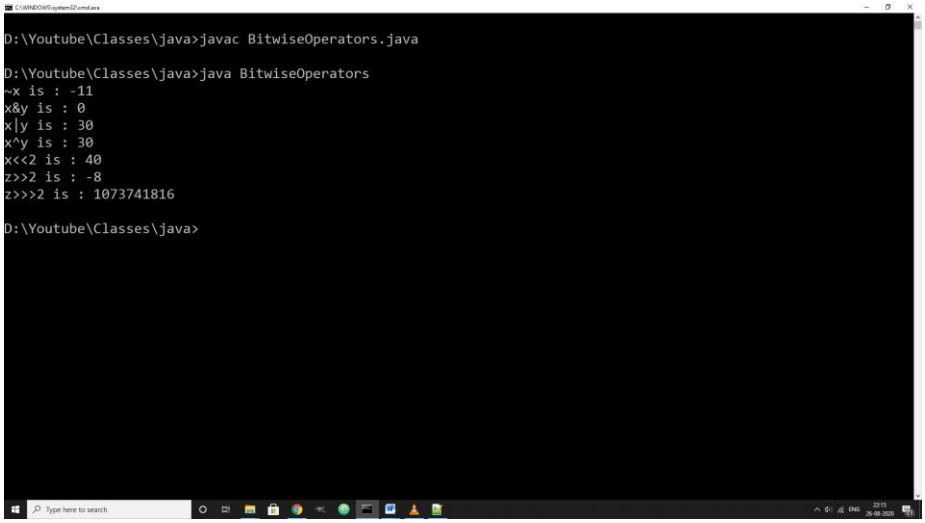
Bitwise operators are applied on numbers. While applying it internally converts the operands into 0&1 and then apply the corresponding operator on it

operator	meaning
~(Not)	takes binary representation and change each digit from 0 to 1 or 1 to 0
&(And)	AND operation on both the operands – if both are 1 then only it returns 1
|(Or)	OR operation on both the operands – if both are 0 then only result is 0
^(XOR)	XOR operation on both the operands (if both inputs are same then result is false)
<<(Shift Left)	shifts all the bits to left for specified number of positions
>>(Shift Right)	shifts all the bits to right and fill empty bit with the sign bit
>>>(Shift Right zero fill)	shifts all the bits to right and fill empty bit with zero

Example:

```
//program to demonstrate Bitwise operations in java

class BitwiseOperators
{
	public static void main(String ar[])
	{
		byte x=10;
		byte y=20;
		byte z=-30;

		System.out.println("~x is : "+(~x));
		System.out.println("x&y is : "+(x&y));
		System.out.println("x|y is : "+(x|y));
		System.out.println("x^y is : "+(x^y));
		System.out.println("x<<2 is : "+(x<<2));
		System.out.println("z>>2 is : "+(z>>2));
		System.out.println("z>>>2 is : "+(z>>>2));

	}
}
```

Output:



Conditional or Ternary Operator (?:)

Conditional operator is used to execute one of the statements based on true or false of a condition.

Syntax:

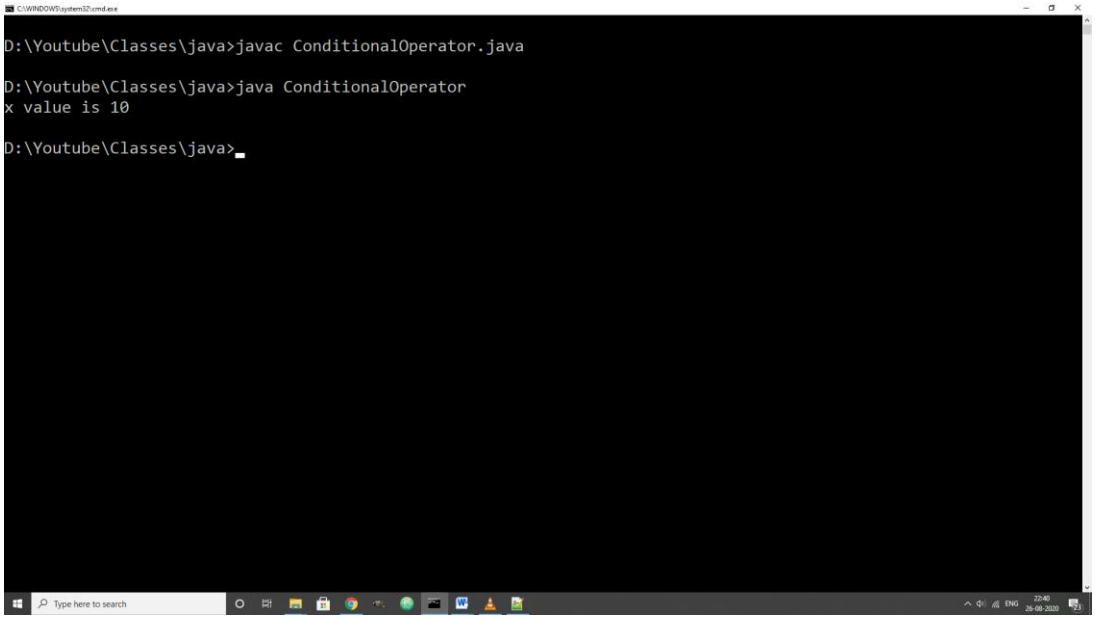
Condition ? statement1 : statement2

In above syntax If specified condition is true then statement1 will be executed else statement2 will be executed.

Example:

```
class ConditionalOperator
{
  public static void main(String ar[])
  {
        int x=10;

        System.out.println(x==10 ? "x value is 10" : "x value is not 10");
  }
}
```

Output



CONTROL STATEMENTS IN JAVA

There are three types of control statements

1) Selection Statements
2) Iteration Statements
3) Jump Statements

Selection Statements:

Java supports two selection statements: if and switch. These statements allow you to control the flow of your program's execution based upon conditions known only during run time.

if :

if condition syntax is as shown below

```
if (condition)
        statement1;
else
        statement2;
```

Nested ifs

A nested if is an if statement within another if. Nested ifs are very common in programming.

```
if(condition)
{
    if(condition)
        statement;
}
```

The if-else-if Ladder

We can write any number of if conditions and execute a block of statements based on met condition.

Syntax is as follows

```
if (condition1)
        statements;
 else if(condition2)
        statements;
        …
```

else

statements;

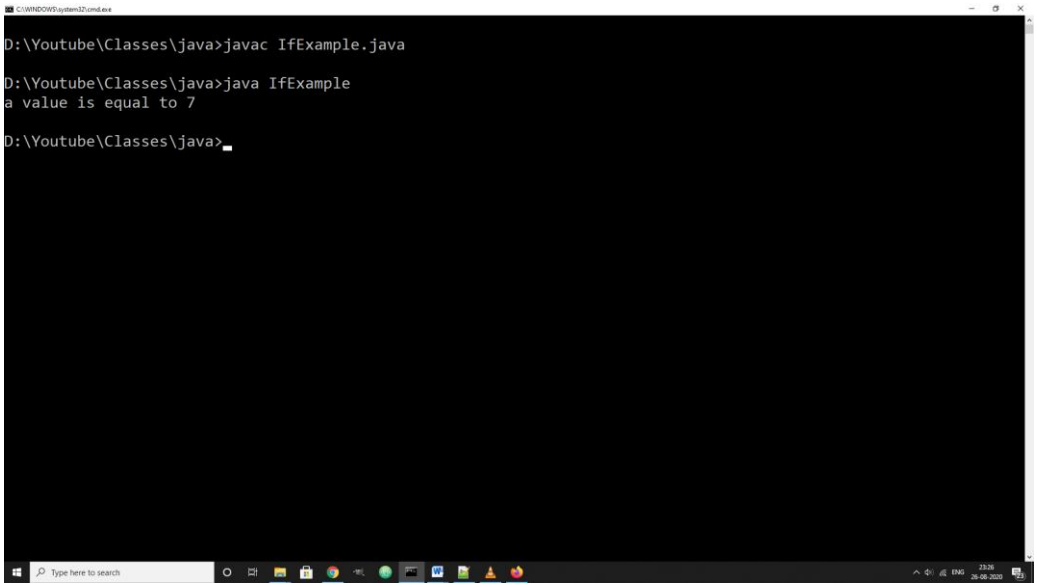
Block of statements whose condition becomes true will be executed. If all conditions become false then else part statements will be executed.

Example program is shown below

Example:

```
//program to demonstrate if statement
class IfExample
{
  public static void main(String ar[])
  {
            int a=7;

            if(a==7)
                  System.out.println("a value is equal to 7");
            else if(a<7)
                  System.out.println("a value is less than 7");
            else
                  System.out.println("a value is greater than 7");

  }
}
```

Output



Switch:

The switch statement is Java's multiway branch statement. It provides an easy way to dispatch execution to different parts of your code based on the value of an expression.

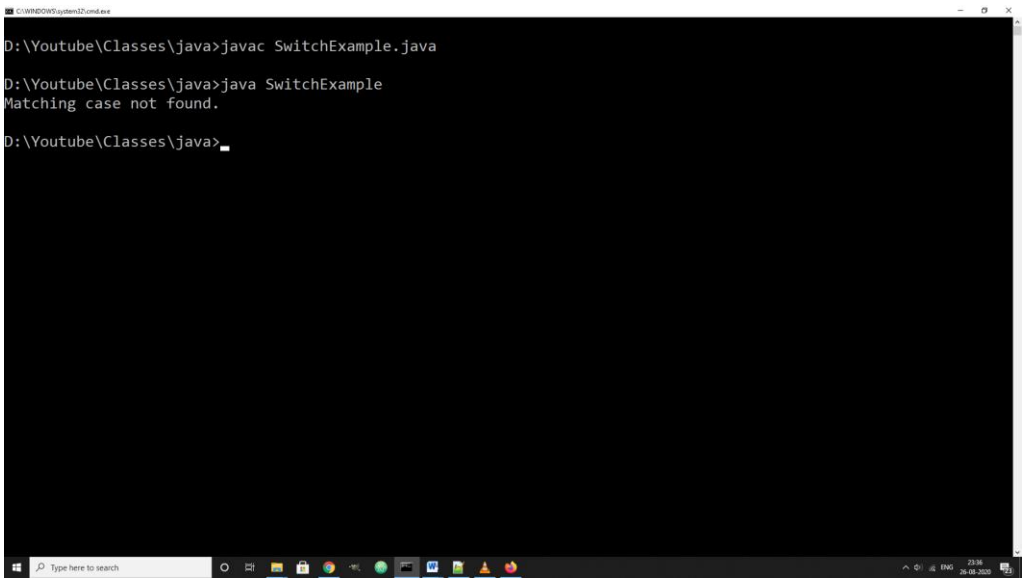
Syntax is :

```
switch (expression) {
case value1:
        // statement sequence
        break;
case value2:
        // statement sequence
        break;
.
.
.
case valueN:
        // statement sequence
        break;
default:
        // default statement sequence
}
```

In the above syntax which case matches with the expression the corresponding statements are executed. here break statement is mandatory to terminate from the switch block after executed set of statements under a case. If break statement is not used, then all the next cases will be executed instead of terminating from the switch block.

```
//program to demonstrate switch statement
class SwitchExample
{
  public static void main(String ar[])
  {
            int a=10;

            switch(a)
            {
                    case 0:
                            System.out.println("a value is equal to 0");
                            break;

                    case 5:
                            System.out.println("a value is equal to 5");
                            break;

                    case 7:
                            System.out.println("a value is equal to 7");
                            break;


                    default:
                    System.out.println("Matching case not found.");

            }
  }

}
```

Output



Iteration Statements or Loop Statements:

Java's iteration statements are for, while, and do-while. These statements create what we commonly call loops

while

The while loop is Java's most fundamental looping statement. It repeats a statement or block while its controlling expression is true.

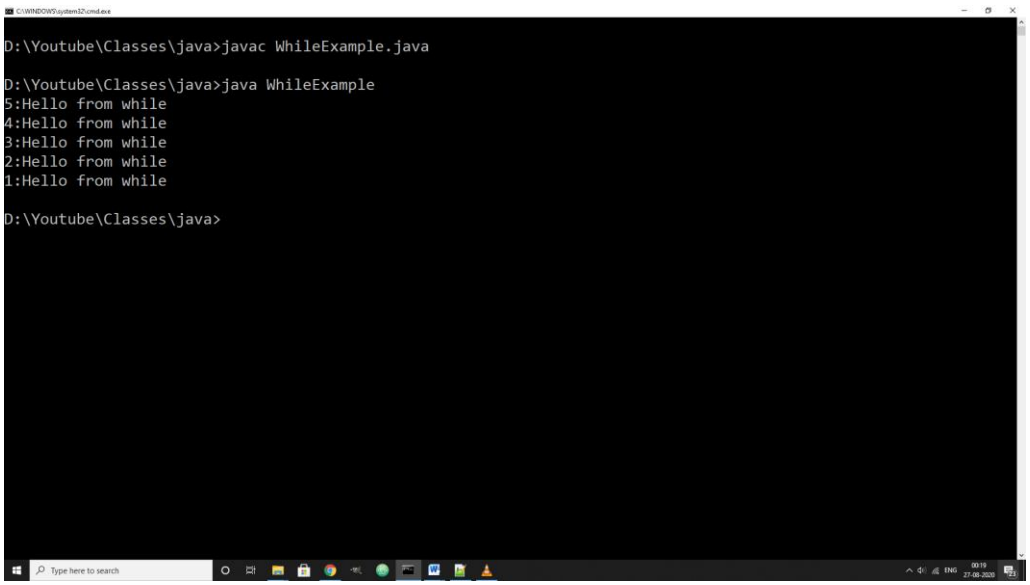
Here is its general form:

```
while(condition) {
        // body of loop
}
```

Example:

```
//program to demonstrate while loop
class WhileExample
{
        public static void main(String ar[])
        {
                int i=5;

                while(i>0)
                        {
                                System.out.println(i+":Hello from while");
                                i--;
                        }
        }
}
```

Output



do-while

The do-while loop always executes its body at least once, because its conditional expression is at the bottom of the loop.

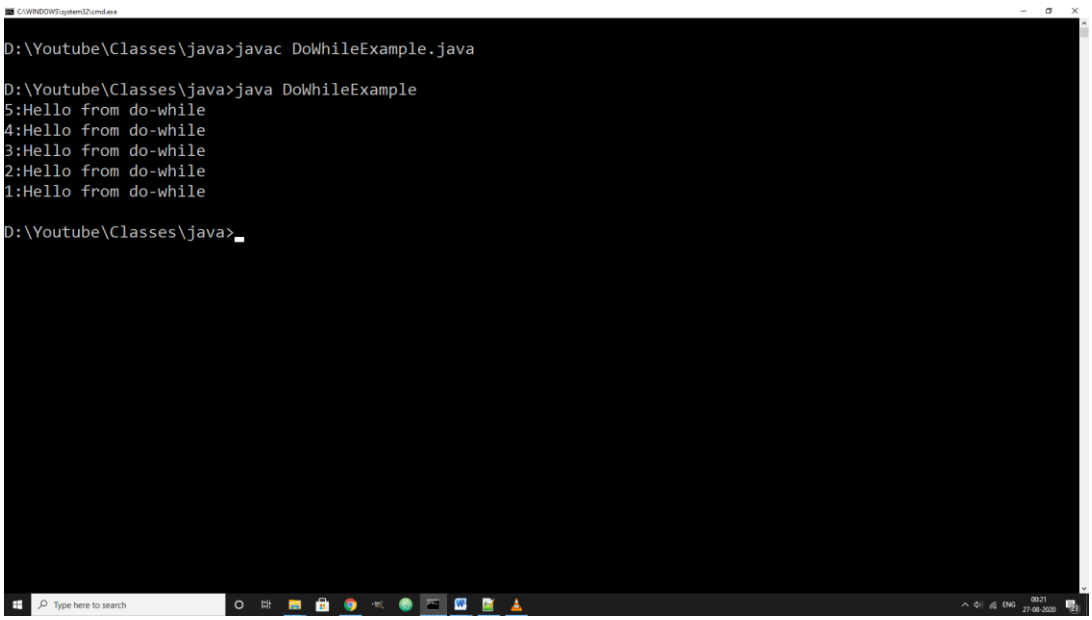
Its general form is

```
do {
        // body of loop
  } while (condition);
```

Example:

```
//program to demonstrate do-while loop
class DoWhileExample
{
        public static void main(String ar[])
        {
                int i=5;

                  do
                        {
                                System.out.println(i+":Hello from do-while");
                                i--;
                        }while(i>0);
        }
}
```

Output



for

Here is the general form of the for statement:

for(*initialization*; *condition*; *iteration*) {

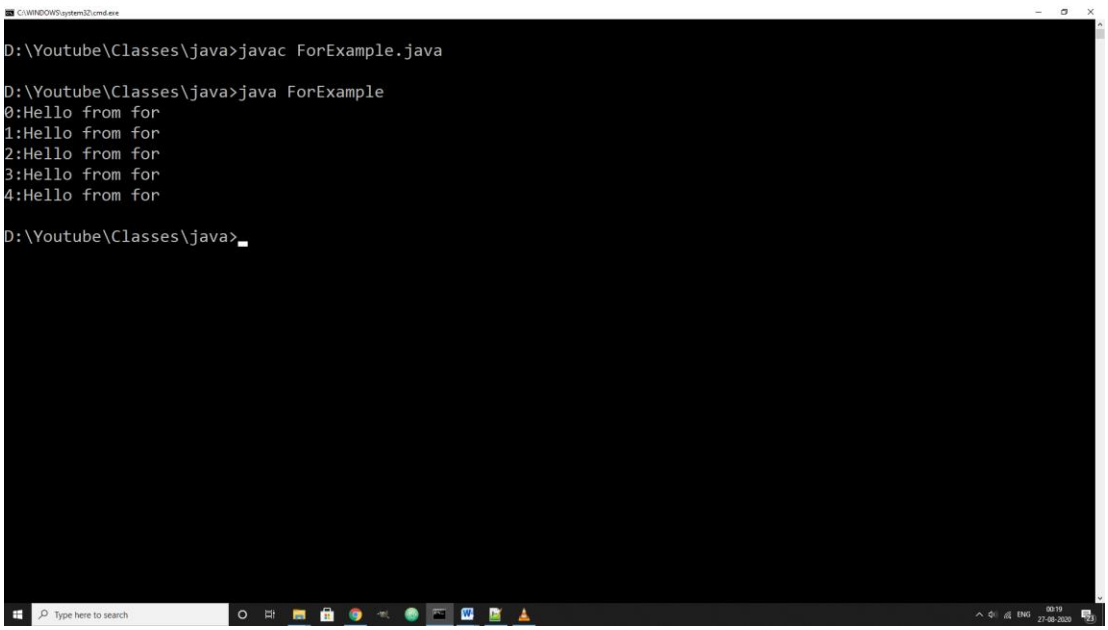
// body

}

Example:

```
//program to demonstrate for loop
class ForExample
{
	public static void main(String ar[])
	{
			for(int i=0;i<5;i++)
			{
				System.out.println(i+":Hello from for");
			}
	}
}
```

Output



Jump Statements

Java supports three jump statements: break, continue, and return. These statements transfer control to another part of your program.

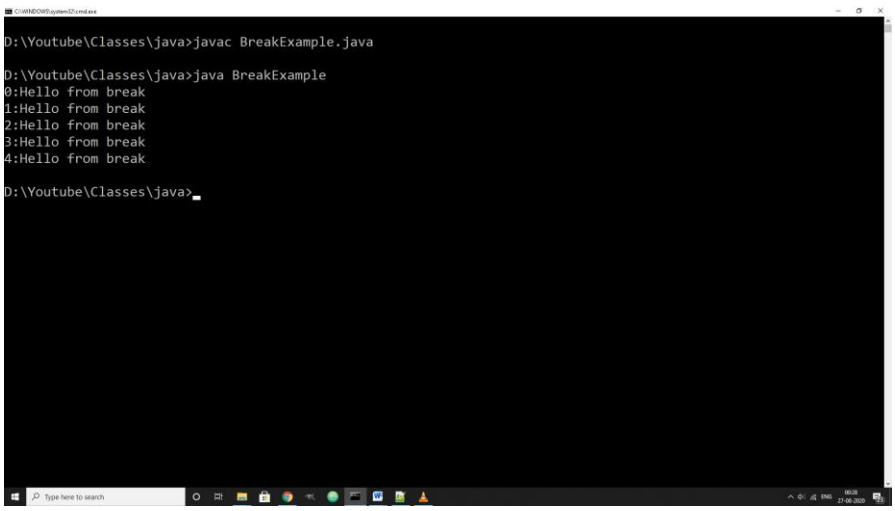
break:

In Java, the break statement has three uses. First, as you have seen, it terminates a statement sequence in a switch statement. Second, it can be used to exit a loop. Third, it can be used as a "civilized" form of goto.

Example

```
//program to demonstrate break statement
class BreakExample
{
	public static void main(String ar[])
	{
			for(int i=0;i<10;i++)
			{
				if(i==5)
					break;

				System.out.println(i+":Hello from break");
			}
	}
}
```

Output



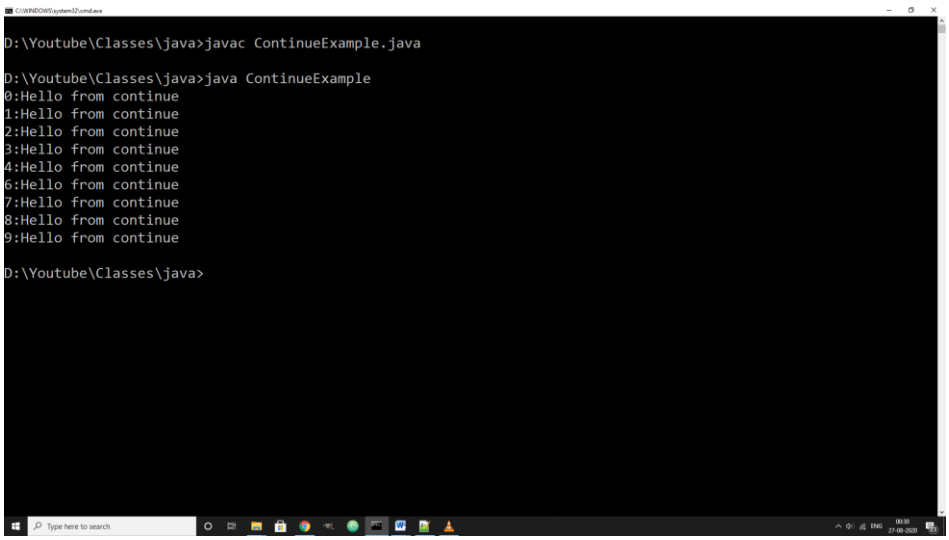
continue:

The continue statement just skips the iteration only and it continues from next iteration as shown in in following example.

Example

```
//program to demonstrate continue statement
class ContinueExample
{
	public static void main(String ar[])
	{
		for(int i=0;i<10;i++)
		{
			if(i==5)
				continue;

			System.out.println(i+":Hello from continue");
		}
	}
}
```

Output



INPUT & OUTPUT IN JAVA

How to Read input from the Keyboard?

To read input from keyboard or from a file we need a stream in java. A stream represents flow of data between one to another place. Basically there are two types of streams - input streams and output streams. Input streams are those streams which receive or read data from some other place. Output streams will send or write data.

Keyboard is represented by a field, called ' *in* ' in System class.

System.in	–	represents InputStream object that represents keyboard device
System.out	-	represents PrintStream object that represents monitor by default
System.err	-	also same as System.out but it is used to output error messages.

To use Input Output steams in our programs we need to import java.io package which provides us all the classes and interfaces related to input and output

```
import java.io.*;
```

To read input from keyboard we will use InputStreamReader which directly connects with keyboard (System.in)

```
InputStreamReader obj = new InputStreamReader(System.in)
```

To read input we need to use BufferedReader which connects with obj (InputStreamReader) which is just created.

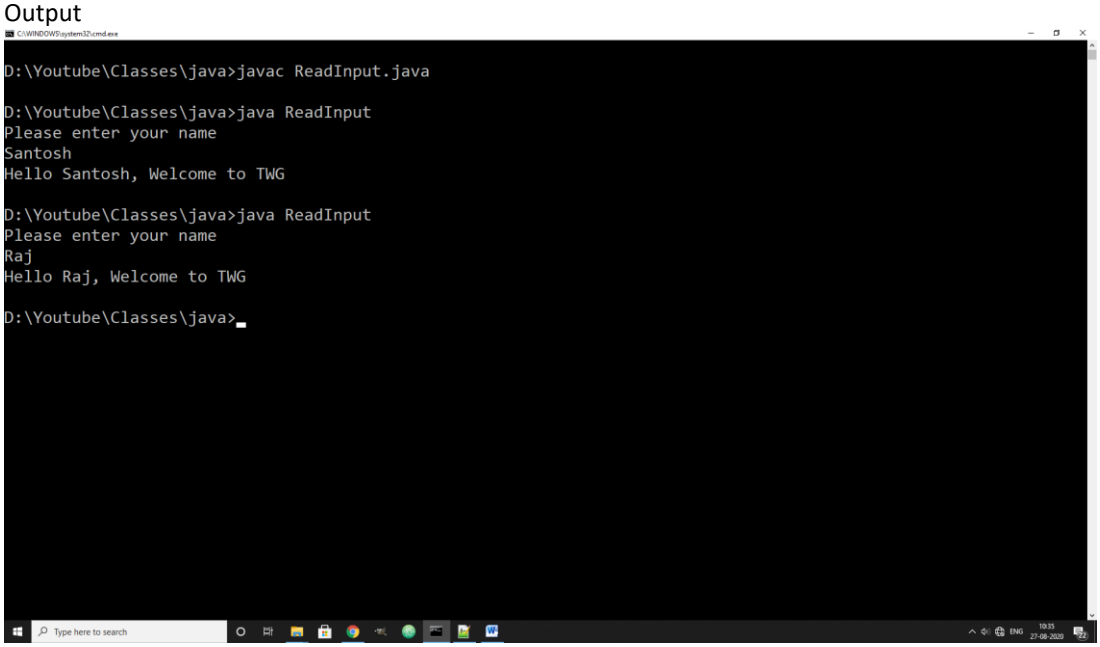
```
BufferedReader br = new BufferedReader(obj);
```

Now we can read characters or strings with the help of methods provided by Buffered Reader.

Example:

```
//program to demonstrate reading input from keyboard
import java.io.*;

class ReadInput
{
	public static void main(String ar[]) throws IOException
	{
		//connect input stream with the keyboard
		InputStreamReader obj=new InputStreamReader(System.in);

		//connect BufferedReader with the obj
		BufferedReader br = new BufferedReader(obj);

		//read input Now
		System.out.println("Please enter your name");
		String name = br.readLine();

		//display the string
		System.out.println("Hello "+name+", Welcome to TWG");
	}
}
```

Output



Alternatively we can use Scanner class also to read input from the keyword

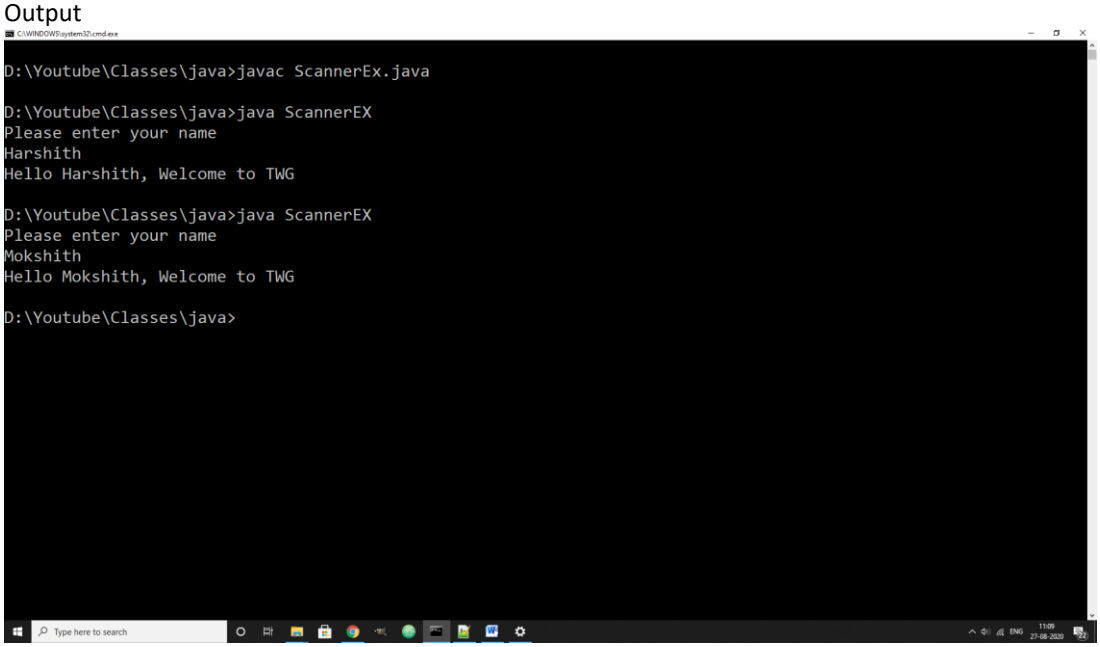
```
//program to demonstrate reading input from keyboard using Scanner
import java.util.*;

class ScannerEX
{
	public static void main(String ar[])
	{
		//connect input stream with the keyboard
		Scanner obj=new Scanner(System.in);


		//read input Now
		System.out.println("Please enter your name");
		String name = obj.next();

		//display the string
		System.out.println("Hello "+name+", Welcome to TWG");
	}
}
```

Here Scanner class provides next(), nextInt(), nextFloat() so on to read input from keyboard as String, int float etc.,

Scanner class is available in java.util package so we need to import java.util before using Scanner class.

Java outputs on to the screen by using System.out which is already discusses in our first program.

Output


D:\Youtube\Classes\java>javac ScannerEx.java
D:\Youtube\Classes\java>java ScannerEX
Please enter your name
Harshith
Hello Harshith, Welcome to TWG
D:\Youtube\Classes\java>java ScannerEX
Please enter your name
Mokshith
Hello Mokshith, Welcome to TWG
D:\Youtube\Classes\java>
Type here to search

ARRAYS IN JAVA

An Array is group of / collection of elements of similar data type.

There are two types of arrays.

1) Single Dimensional Arrays
2) Multi-Dimensional Arrays

Single Dimensional Array (1D Array) :

It represents a row or column of elements

We can create an 1-D Array in two steps using the following syntax.

```
Datatype arrayname[ ]
arrayname = new Datatype[size];
```

Example :

```
int marks[];
marks = new int[5];   // An 1D array is created which can store 5 elements of type int
```

We can achieve this in single statement also as follows.

```
Datatype arrayname[ ]=new Datatype[size];
```

Example:

```
int marks[ ] = new int[5];
```

The above statement saves lot of time of the developer because without arrays if we want to store five integer elements we need 5 integer variables. Instead of that with single statement (Array) we are able to store any number of values.

Now we can store elements into an array by using it's index which starts from 0.

```
marks[0]=10;
marks[1]=9;
marks[2]=7;
marks[3]=8;
marks[4]=9;
```

We have an alternative way where we can create array and save elements into the array by using single statement as follows.

```
int marks[ ] = {10,9,7,8,9}
```

We can insert [ ] with array name or datatype also as follows

```
int[  ] marks;
int marks[ ];
```

These both are valid statements.

we can access / print the elements from an array by using index.

for example : marks[4] = 10; // this will update the element at index 4 from previous value(9) to 10

How to print the elements in an array:

We can print the elements in an array with the help of a loop.

```
for(int i=0;i<marks.length;i++)
{
        System.out.println(marks[i]);
}
```

Here length is a method that is used to find out length of an array.

Example

```
//program to demonstrate single dimensional arrays
class ArrayEx
{
        public static void main(String ar[])
        {
                //creating an array
                int marks[] = new int[5];

                //inserting elements into array
                marks[0]=10;
                marks[1]=9;
                marks[2]=7;
                marks[3]=8;
                marks[4]=9;

                //printing elements from the array
                System.out.println("marks[3]:"+marks[3]);

                //printing all elements
                for(int i=0;i<marks.length;i++)
                {
                System.out.println(marks[i]);
                }


        }
}
```

Output

```
D:\Youtube\Classes\java>javac ArrayEx.java

D:\Youtube\Classes\java>java ArrayEx
marks[3]:8
10
9
7
8
9

D:\Youtube\Classes\java>
```

Two-Dimensional Array (2D)

2-d arrays represent rows and columns of data. i.e., it can store set of 1-D arrays in it.

For example, we can store marks of multiple students in various subjects using 2-d array.

How to create/declare 2-d array:

datatype arrayname[][] = new datatype[rowsize][colsize]

Example

int marks[][] = new int[3][5]

This example creates marks array which can store 3 rows and 5 columns of information. i.e., it can store 15 elements. It looks like below.

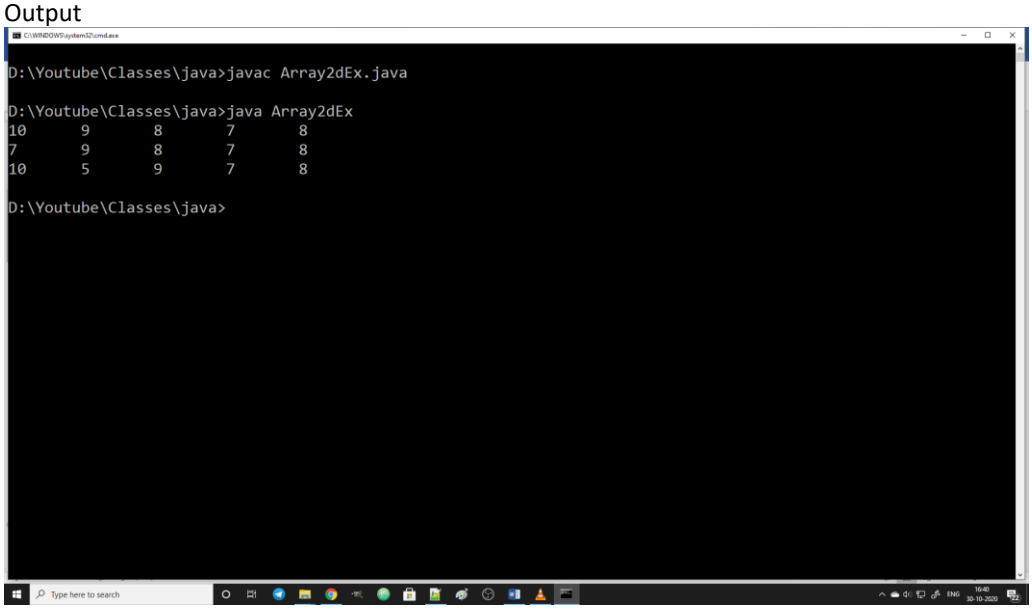
In above pic, elements (black color) are displayed according to their index positions (green color). Generally, for both rows and columns the index starts from 0. So index of the first row first column element is 00 and the index of first row second column element is 01 so on.

Once a 2-d array is created elements are stored and access by using their index values.

Example

```
//program to demonstrate 2-d arrays in java
class Array2dEx
{
	public static void main(String ar[])
	{
		//creating and inserting elements into an 2d array
		int marks[][] = {{10,9,8,7,8},{7,9,8,7,8},{10,5,9,7,8}};

	  for(int i=0;i<3;i++) //rows
		{
			for(int j=0;j<5;j++) //columns
		  {
			  System.out.print(marks[i][j]+"\t");
		  }

			System.out.println();

		}
	}
}
```

Output



Three Dimensional Array (3D) :

3-d arrays can store set of two-dimensional arrays.

From the above example, if we want to store all the students marks of all branches then we may need to maintain individual 2d arrays for each branch.

alternatively, we can use a 3-d array where it can store all the branches and all the students marks at a time in a single 3d array.

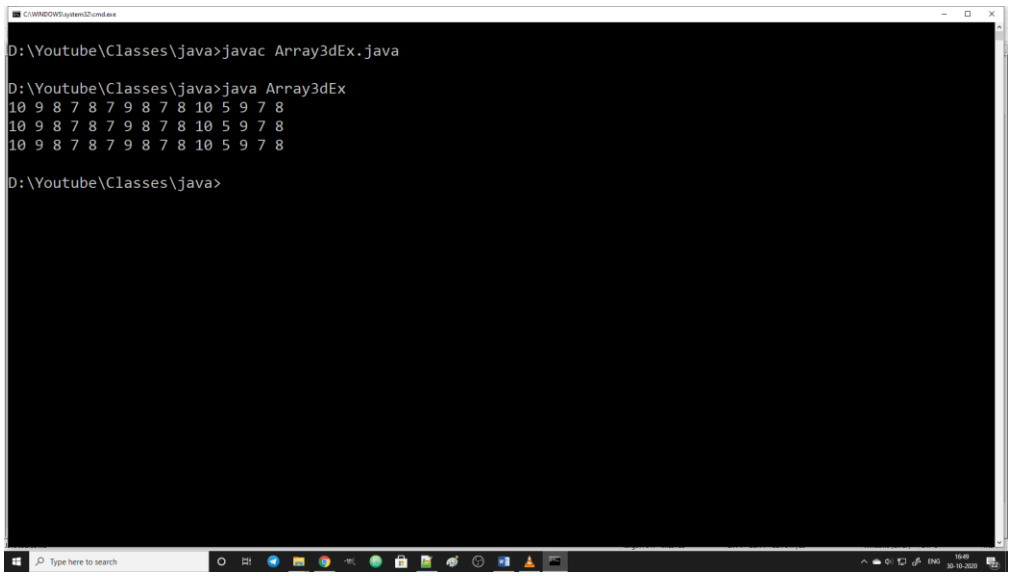
How to create a 3d array :

```
int campusmarks[][][] = new int[3][3][5];
```

Example

```
//program to demonstrate 3-d arrays in java
class Array3dEx
{
	public static void main(String ar[])
	{
		//creating and inserting elements into an 2d array
		int marks[][][] = {
			{{10,9,8,7,8},{7,9,8,7,8},{10,5,9,7,8}},
				{{10,9,8,7,8},{7,9,8,7,8},{10,5,9,7,8}},
				{{10,9,8,7,8},{7,9,8,7,8},{10,5,9,7,8}}


			};

	   for(int i=0;i<3;i++) //2-d array index
		{
			for(int j=0;j<3;j++) //row
		  {
			 for(int k=0;k<5;k++) //column
			 {
				System.out.print(marks[i][j][k]+" ");
			 }
		  }

			System.out.println();

		}
	}
}
```

Output



STRINGS IN JAVA

Collection of characters or group of characters is said to be a String.

String is a class which is available in java.lang package. In java all classes are also considered as data types. So in java, a string is a data type as well as a class.

How to create a String ?

We can create strings in three ways.

1) By declaring a string as a variable

```
String name = "Santosh";
```

2) By declaring string as an object of class String

```
String name = new String("Santosh");
```

3) Creating string from character array.

```
char c [] = {'s', 'a', 'n', 't', 'o', 's', 'h'};
String name = new String(c);
```

String Methods:

String concat(String s)

Concatenates the specified string to the end of this string.

Example:

```
String s = "Welcome to ";
String s1 = s.concat("Telugu Web Guru");
```

The above statement will concatenate the substring "Telugu Web Guru" to String s and result s1 becomes "Welcome to Telugu Web Guru"

int length()

Returns the length of this string.

Example:

```
String s = "Hello";
int l = s.length();
```

Above statement will returns length of the string and stores it in variable l. So now l contains the value 5.

char charAt(int i)

Returns the char value at the specified index i

Example

```
String s = "Welcome";
char c = s.charAt(2);
```

above method returns the character at index 2 in string s and result character (' l ') will be stored in char c.

int compareTo(String s)

Compares two strings lexicographically.

compareTo returns 0 if both strings are equal. It returns positive value if first string is greater than second string else it returns negative value.

int compareToIgnoreCase(String str)

Compares two strings lexicographically, ignoring case differences.

It is same as compareTo but it ignores the case while comparing the strings.

boolean equals(String s)

Compares this string to the specified object and returns true or false.

If we want to compare two strings for equality of its contents then we need to use equals method. If we use == it never compares the content of the strings rather it compares the references.

boolean equalsIgnoreCase(String anotherString)

Compares this String to another String, ignoring case considerations.

it is same as equals but it ignores the case while comparing.

boolean startsWith(String s)

Tests if this string starts with the specified prefix.

It returns true if string on which startsWith method running is started with the substring s or not. else it returns false.

boolean endsWith(String suffix)

Tests if this string ends with the specified suffix.

It returns true if string on which this method runs is ends with the substring else it returns false

int indexOf(String s)

Returns the index within this string of the first occurrence of the specified substring.

int lastIndexOf(String str)

Returns the index within this string of the last occurrence of the specified substring.

String replace(char c1,char c2)

Returns a string resulting from replacing all occurrences of oldChar in this string with newChar.

String substring(int beginIndex)

Returns a string that is a substring of this string.

String substring(int i1,int i2)

Returns a string that is a substring of this string.

String toLowerCase()

Converts all of the characters in this String to lower case using the rules of the default locale.

String toUpperCase()

Converts all of the characters in this String to upper case using the rules of the given Locale.

String trim()

Returns a string whose value is this string, with any leading and trailing whitespace removed.

String[] split(delimiter)

Splits this string around matches of the given regular expression.

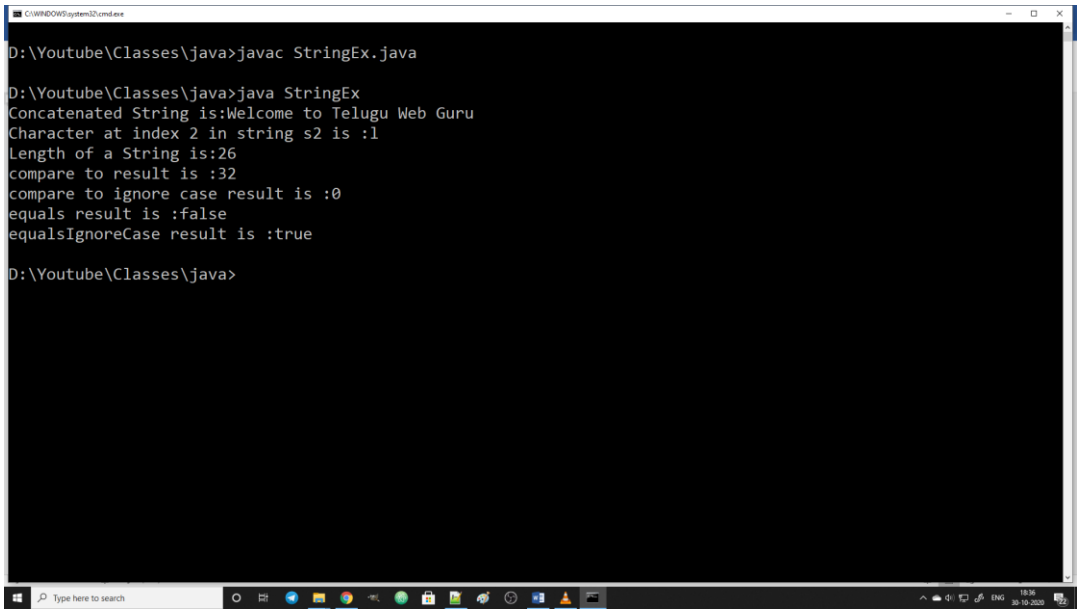
void getChars(int i1,int i2,char arr[],int i3)

Copies characters from this string into the destination character array.

Example 1 :

```
//program to demonstrate String methods
class StringEx{
  public static void main(String ar[])
  {
              //create a string
             String s = "Welcome to";
             String s1  = s.concat(" Telugu Web Guru");
             System.out.println("Concatenated String is:"+s1);
             String s2 = "Welcome to Telugu Web Guru";
             char c =  s2.charAt(2);
             System.out.println("Character at index 2 in string s2 is :"+c);
             int len = s2.length();
             System.out.println("Length of a String is:"+len);
             String s3 = "welcome to Telugu Web Guru";
             System.out.println("compare to result is :"+s3.compareTo(s2));
             System.out.println("compare to ignore case result is :"+s3.compareToIgnoreCase(s2));
             System.out.println("equals result is :"+s3.equals(s2));
             System.out.println("equalsIgnoreCase result is :"+s3.equalsIgnoreCase(s2));
  }

}
```

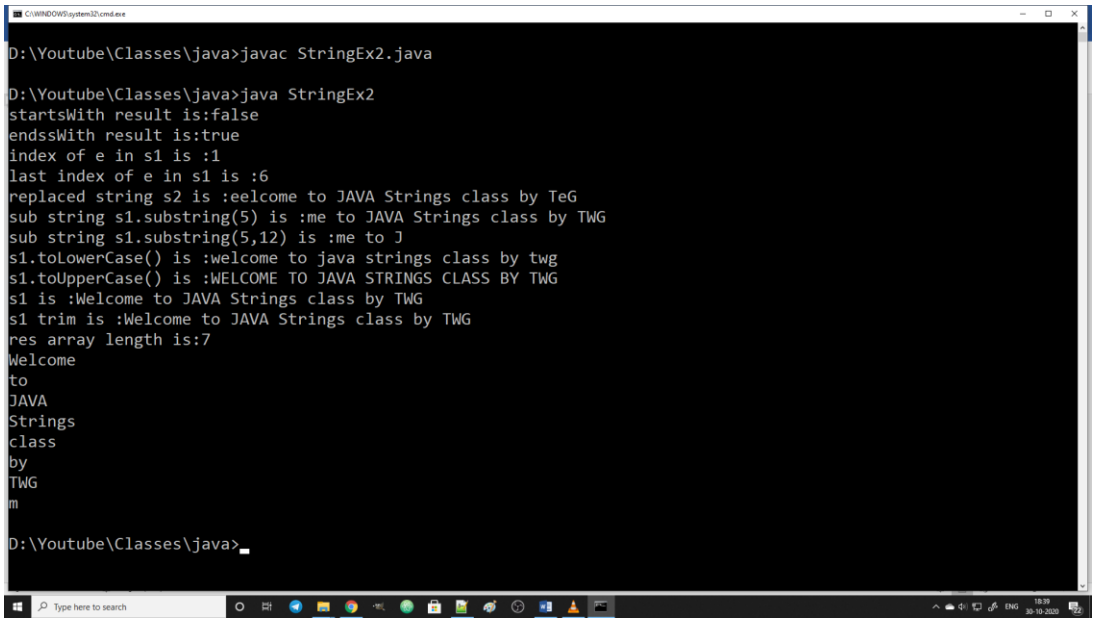
Output:



Example 2:

```
//program to demonstrate String methods
class StringEx2{
  public static void main (String ar[])
  {

            String s1 = "Welcome to JAVA Strings class by TWG";
            System.out.println("startsWith result is:"+s1.startsWith("wel"));
            System.out.println("endssWith result is:"+s1.endsWith("TWG"));

             System.out.println("index of e in s1 is :"+s1.indexOf("e"));
             System.out.println("last index of e in s1 is :"+s1.lastIndexOf("e"));
             String s2 = s1.replace('W','e');
             System.out.println("replaced string s2 is :"+s2);
             System.out.println("sub string s1.substring(5) is :"+s1.substring(5));
             System.out.println("sub string s1.substring(5,12) is :"+s1.substring(5,12));
             System.out.println("s1.toLowerCase() is :"+s1.toLowerCase());
             System.out.println("s1.toUpperCase() is :"+s1.toUpperCase());
             System.out.println("s1 is :"+s1);
             System.out.println("s1 trim is :"+s1.trim());
             String res[]    =    s1.split(" ");
             System.out.println("res array length is:"+res.length);

             for(int i=0;i<res.length;i++)
                        System.out.println(res[i]);

             char a[] = new char[100];
             s1.getChars(5,12,a,1);
             System.out.println(a[1]);

  }
}
```

Output


D:\Youtube\Classes\java>javac StringEx2.java
D:\Youtube\Classes\java>java StringEx2
startsWith result is:false
endssWith result is:true
index of e in s1 is :1
last index of e in s1 is :6
replaced string s2 is :eelcome to JAVA Strings class by TeG
sub string s1.substring(5) is :me to JAVA Strings class by TWG
sub string s1.substring(5,12) is :me to J
s1.toLowerCase() is :welcome to java strings class by twg
s1.toUpperCase() is :WELCOME TO JAVA STRINGS CLASS BY TWG
s1 is :Welcome to JAVA Strings class by TWG
s1 trim is :Welcome to JAVA Strings class by TWG
res array length is:7
Welcome
to
JAVA
Strings
class
by
TWG
m
D:\Youtube\Classes\java>

STRINGBUFFER IN JAVA

Generally, Strings are immutable. So, their contents cannot be modified. To overcome this problem StringBuffer is introduced.

Creating StringBuffer:

There are two ways to create StringBuffer

1) create an object by using new operator and pass the string

```
StringBuffer obj = new StringBuffer("Welcome to TeluguWebGuru");
```

2) First create object and allocate the memory and then store the string into it. In this case object is created with the capacity of 16 characters.

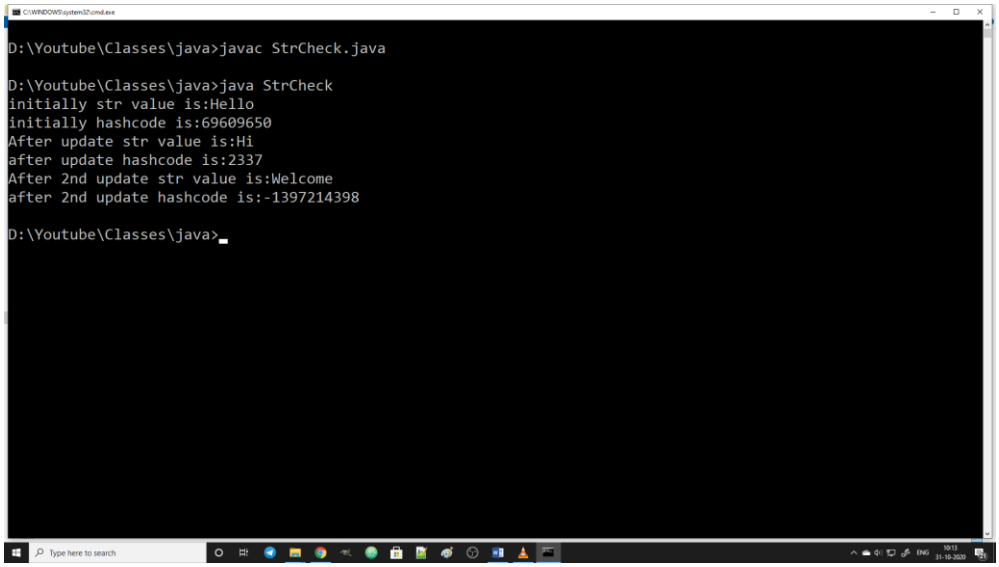
```
StringBuffer obj = new StringBuffer();
```

Example

```
//program to check whether a string is immutable or not

class StrCheck
{
		public static void main(String ar[])
		{
			String str = new String("Hello");

			System.out.println("initially str value is:"+str);
			System.out.println("initially hashcode is:"+str.hashCode());

			str = "Hi";

			System.out.println("After update str value is:"+str);
			System.out.println("after update hashcode is:"+str.hashCode());

			str = "Welcome";

			System.out.println("After 2nd update str value is:"+str);
			System.out.println("after 2nd update hashcode is:"+str.hashCode());

		}

}
```

In above example, observe that whenever string contents are changed, new memory location is allocated and pointed by the string object.

Output



Methods in StringBuffer :

1) StringBuffer append(String value)

   Appends the specified string to this character sequence.

2) StringBuffer insert(index,value)

   Inserts the string into this character sequence.

3) StringBuffer delete(int beginindex, index endindex)

   Removes the characters in a substring of this sequence.

4) String reverse()

   Causes this character sequence to be replaced by the reverse of the sequence.

5) String toString()

   Returns a string representing the data in this sequence.

6) int length()

   Returns the length (character count).

7) int indexOf(String str)

   Returns the index within this string of the first occurrence of the specified substring.

8) int lastIndexOf(String str)

   Returns the index within this string of the rightmost occurrence of the specified substring.

9) StringBuffer replace(int begin, int end, String str)

Replaces the characters in a substring of this sequence with characters in the specified String.

10) String substring(int i)

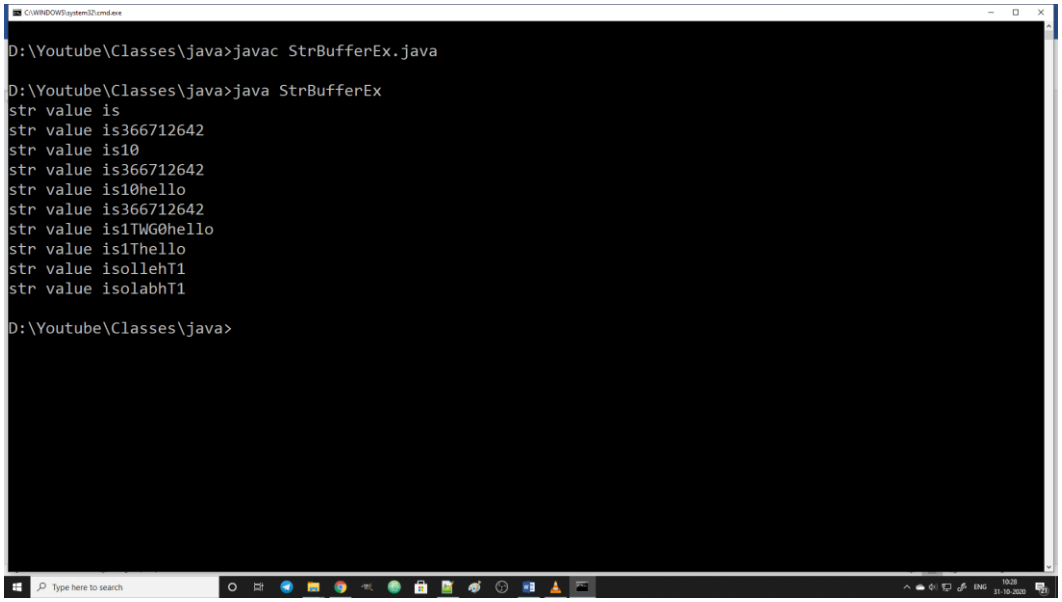
Returns a new String that contains a subsequence of characters currently contained in this character sequence.

11) String substring(int i, int j)

Returns a new String that contains a subsequence of characters currently contained in this sequence.

Example

```
//program to demonstrate String Buffer concept
class StrBufferEx
{
  public static void main(String ar[])
  {
      //creating string buffer object
                  StringBuffer str = new StringBuffer();

                  System.out.println("str value is"+str);
                  System.out.println("str value is"+str.hashCode());

                  //appending values
                  str.append(10);

      System.out.println("str value is"+str);
      System.out.println("str value is"+str.hashCode());

                  str.append("hello");
                  System.out.println("str value is"+str);
                  System.out.println("str value is"+str.hashCode());

                  //inserting elements
                  str.insert(1,"TWG");
                  System.out.println("str value is"+str);

                  //deleting
                  str.delete(2,5);
                  System.out.println("str value is"+str);
                  //reverse
                  str.reverse();
                  System.out.println("str value is"+str);

      //replace
                  str.replace(2,4,"ab");
                  System.out.println("str value is"+str);

  }

}
```


D:\Youtube\Classes\java>javac StrBufferEx.java
D:\Youtube\Classes\java>java StrBufferEx
str value is
str value is366712642
str value is10
str value is366712642
str value is10hello
str value is366712642
str value is1TWG0hello
str value is1Thello
str value isollehT1
str value isolabhT1
D:\Youtube\Classes\java>

CLASS & OBJECT IN JAVA

class:

Class specifies definition of an object. It clearly defines about properties (variables/state/data) that are contained by object and set of actions (methods/behaviour/code) that are need to be performed by that object. A class can also be defined as collection or set of objects.

object:

object is a real time entity and it is an instance of a class.

class is an imaginary and object is real time entity.

How to create a class?

A class can be created by using ' class ' keyword.

syntax:

```
class classname
{
    member variables or instance variables;

     member methods or instance methods;

}
```

variables and methods in a class are said to be instance variables and instance methods. Because for each object(instance) that is instantiated from this class contains separate copy of variables and methods.

How to create an object?

An object of a class can be created as follows.

Syntax :

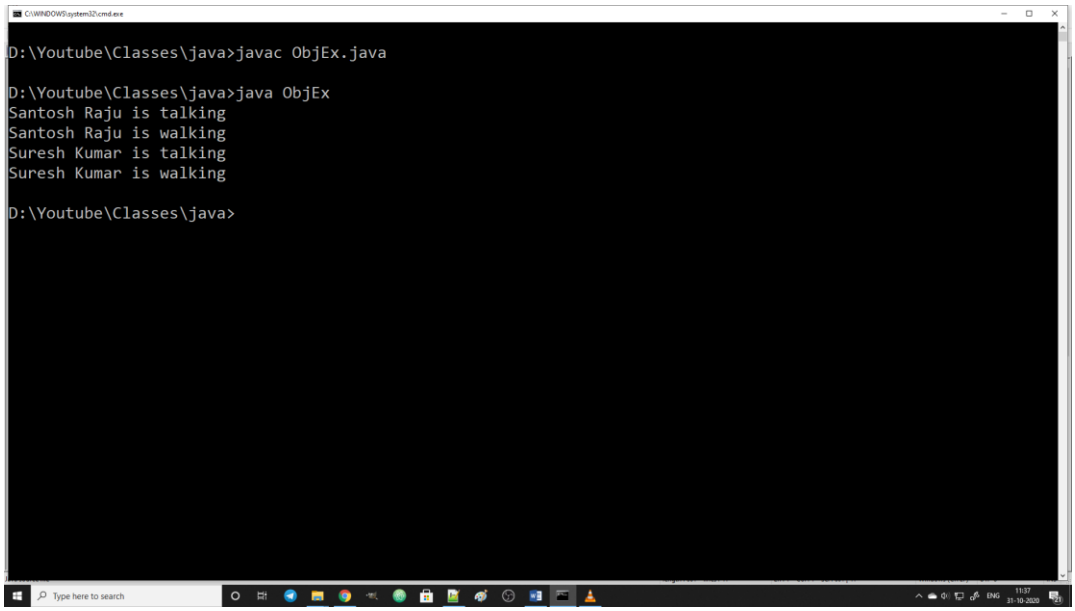
```
classname objectname = new classname(optional arguments);
```

Example

```
class human
{
  //instance variables
  String name;
  int age;

  //instance methods
  void talk(){
  System.out.println(name+" is talking");
  }

  void walk(){
  System.out.println(name+" is walking");
```

```
  }

}

//driver class
class ObjEx
{
	public static void main(String ar[])
	{
		//creating santosh object
		human santosh = new human();

		santosh.name="Santosh Raju";
		santosh.age=35;

		santosh.talk();
		santosh.walk();

		human suresh = new human();

		suresh.name="Suresh Kumar";
		suresh.talk();
		suresh.walk();


	}
}
```

output:



Above example defines a class human with properties and actions and then it creates two objects of class ' human ' in driver class (main class) ObjEx.

observe how classes and objects are created and how to invoke methods in a object.

METHODS IN JAVA

A Method represents group of statements that performs a task.

Difference between function and method ?

Functions that are defined inside a class are said to be methods.

How to create a method ?

A method can be declared as follows.

```
returntype methodname (optional parameters)
{
        // method code
}
```

We can invoke the method of a class by using object.

objectname.method(optional parameters)

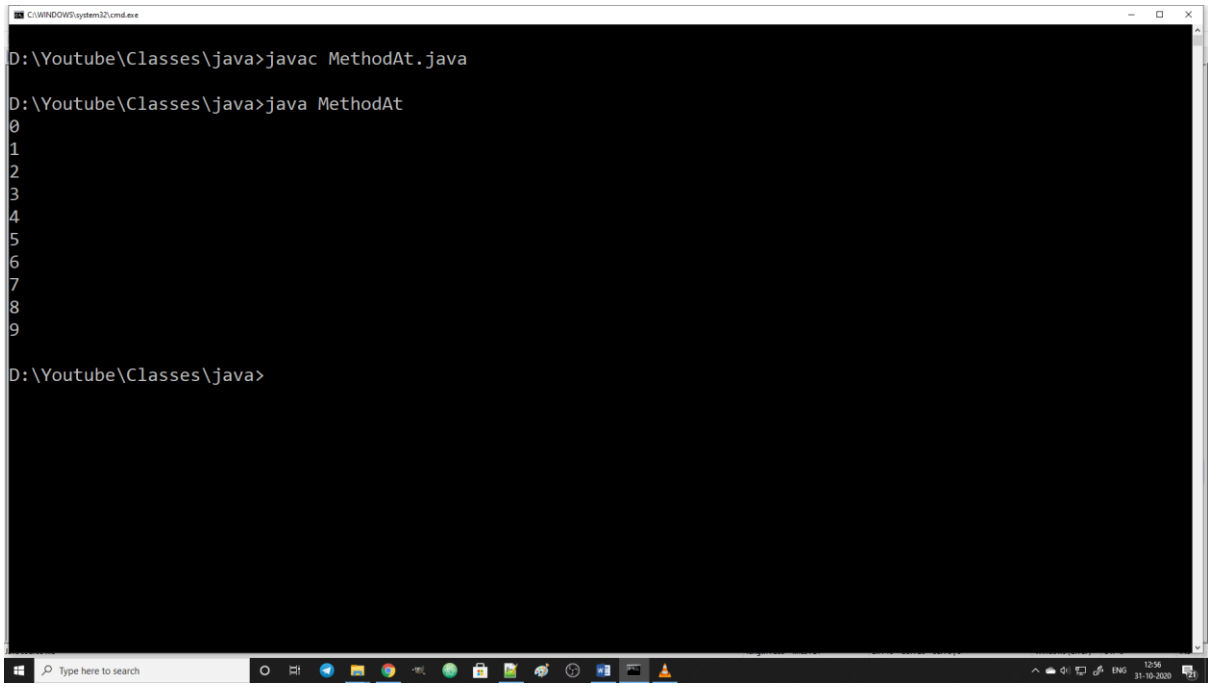
Here the parameters must be matched with the method definition in number of parameters and their data types.

following example defines printarray method in class ArEx and it is invoked from main class by creating object .

Example

```
//program to demonstrate Methods in java

class ArEx
{
  void printarray()
  {
        for(int i=0;i<10;i++)
              System.out.println(i);

  }
}

class MethodAt
{
 public static void main(String ar[])
 {
   ArEx obj = new ArEx();
        obj.printarray();

 }
}
```

Output



We can define methods with parameters also. consider the below concept method overloading and observe how we can use the same name for multiple methods with different number of parameters or different types of parameters.

Method Overloading:

Two or more methods in a class having same name with different signatures is said to be method overloading. Here signature means number of parameters and data types of parameters.

In the following example, the multiple methods are created with the same name ' add ' with differences in either number of parameters or type of parameters or both.

Example

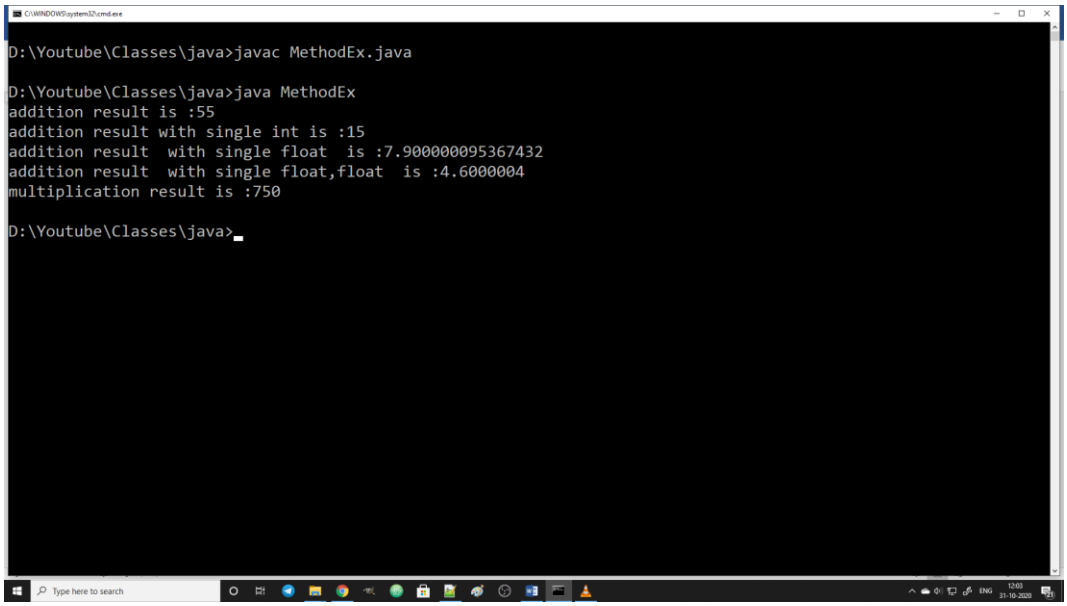
```
class Ex{

  int a;
  int b;
  //method overloading - two or more methods in a class having same name with different signature
  void add()//method definition
  {
                System.out.println("addition result is :"+(a+b));

  }

        void add(int p)//method definition
  {
                System.out.println("addition result with single int is :"+(p+5));

  }
```

```
        void add(float p)//method definition
  {
                System.out.println("addition result  with single float  is :"+(p+5.5));

  }

        void add(float p,float q)//method definition
  {
                System.out.println("addition result  with single float,float  is :"+(p+q));

  }

  void mul()
  {
                System.out.println("multiplication result is :"+(a*b));

  }

}

class MethodEx
{
  public static void main(String ar[])
  {
            int p=25;
            int q=30;

    Ex obj1 = new Ex();
            obj1.a=p;
            obj1.b=q;

            obj1.add();//method calling

            obj1.add(10);

            obj1.add(2.4f);
            obj1.add(1.2f,3.4f);



            obj1.mul();
  }
}
```

Observe that while calling methods, we need to pass the parameter values that must match with the method's definition.

whenever add with single float value is invoked then the system checks for the add method with single float argument and executes it.

Output



Call by Value & Call by Reference:

while passing parameters to the method, if we send primitive data type values then it is called call by value. Instead of passing primitive datatype values if we send objects as parameters then it is called as call by reference.

In call by value if we modify the values of formal parameters (parameters in method definition), it won't affect the values of actual parameters.

In call by reference, instead of sending values we are sending objects directly so updating in method definition definitely affects the actual parameter values also because here we are updating inside the object area.

Example

```
//program to demonstrate call by value and call by reference

class Sample
{
        //16    23
  void add(int a,int b)
        {
                int c = a+b;

                System.out.println("a,b values are :"+a+","+b);
                System.out.println("Addition result is :"+c);

                a = 67;
                b = 89;

                c = a+b;
                System.out.println("a,b values are :"+a+","+b);
                System.out.println("Addition result second time is :"+c);
```

```
        }

        void check(Ex ob1)
        {
                ob1.v1=34;
                ob1.v2=98;

        }
}

class Ex
{
  int v1;
  int v2;
}


class MethodEx2
{
  public static void main(String arr[])
  {
    int p = 16;
        int q = 23;

        Sample s = new Sample();

        System.out.println("p,q values are :"+p+","+q);
           //16 23
        s.add(p,q);//call by value

        System.out.println("p,q values are :"+p+","+q);


        Ex obj1 = new Ex();
        obj1.v1=1;
        obj1.v2=2;

        System.out.println("v1,v2 values before are :"+obj1.v1+","+obj1.v2);
        s.check(obj1);//call by reference
         System.out.println("v1,v2 values after are :"+obj1.v1+","+obj1.v2);

  }
}
```

output

```
D:\Youtube\Classes\java>javac MethodEx2.java

D:\Youtube\Classes\java>java MethodEx2
p,q values are :16,23
a,b values are :16,23
Addition result is :39
a,b values are :67,89
Addition result second time is :156
p,q values are :16,23
v1,v2 values before are :1,2
v1,v2 values after are :34,98

D:\Youtube\Classes\java>
```

In above example, In Call by value p,q values are not changed eventhough a,b values are changed.

where as in Call by Reference v1,v2 values are changed because we are passing reference instead of value so these values got modified directly in memory.

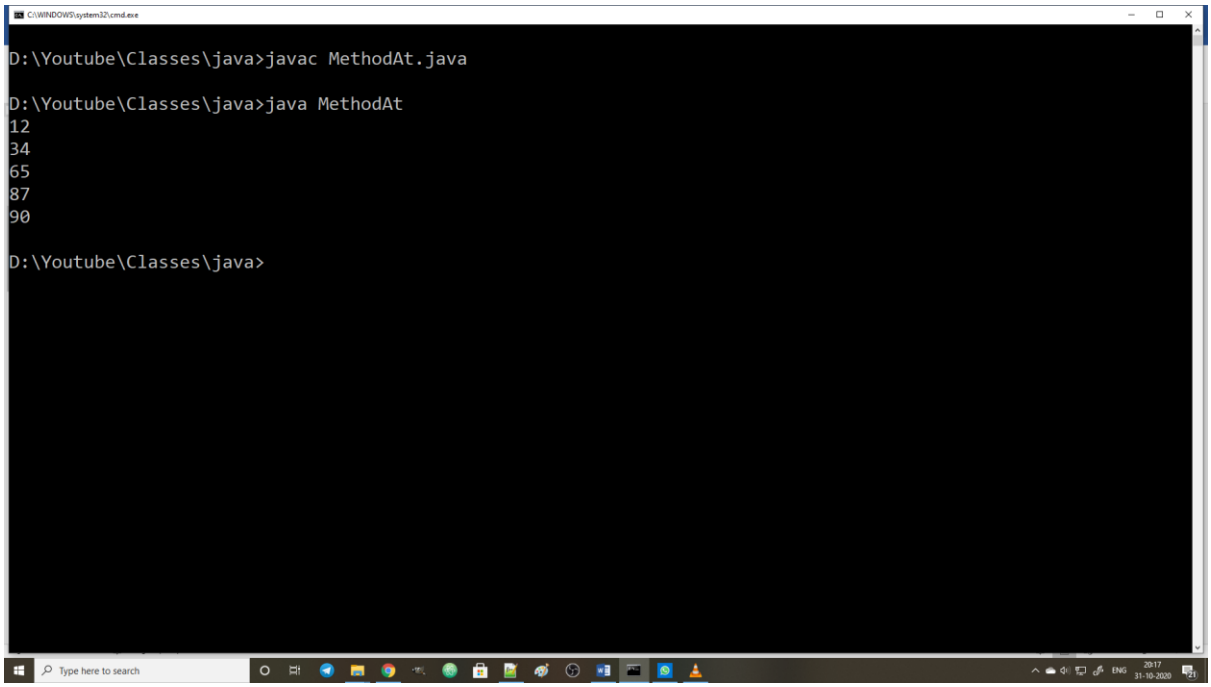
Sending Array as parameter to methods :

We can send arrays also as parameters to methods. following example shows how to send arrays as parameter to methods

Example

```
//program to demonstrate Methods in java

class ArEx
{
  void printarray(int a[])
  {
              for(int i=0;i<a.length;i++)
                        System.out.println(a[i]);

  }
}
class MethodAt
{
 public static void main(String ar[])
 {
   int p[]={12,34,65,87,90};

   ArEx obj = new ArEx();
            obj.printarray(p);

 }
}
```

Output



In above example, an array is passed as parameter to method printarray which prints all the elements of it.

STATIC VARIABLES,METHODS,BLOCKS IN JAVA

If we want to create variables & methods that are common to all objects instead of having separate copy for each object then we will create static variables (class variables) and static methods (class methods).

While accessing these static variables and methods, no need to create any object. As these are members of the class, we can access these static members using class name only.

If we want to execute any statements before executing all the other statements in our program then we may place them under static block. Java system will give the highest priority to static blocks in execution so those statements will be executed first.

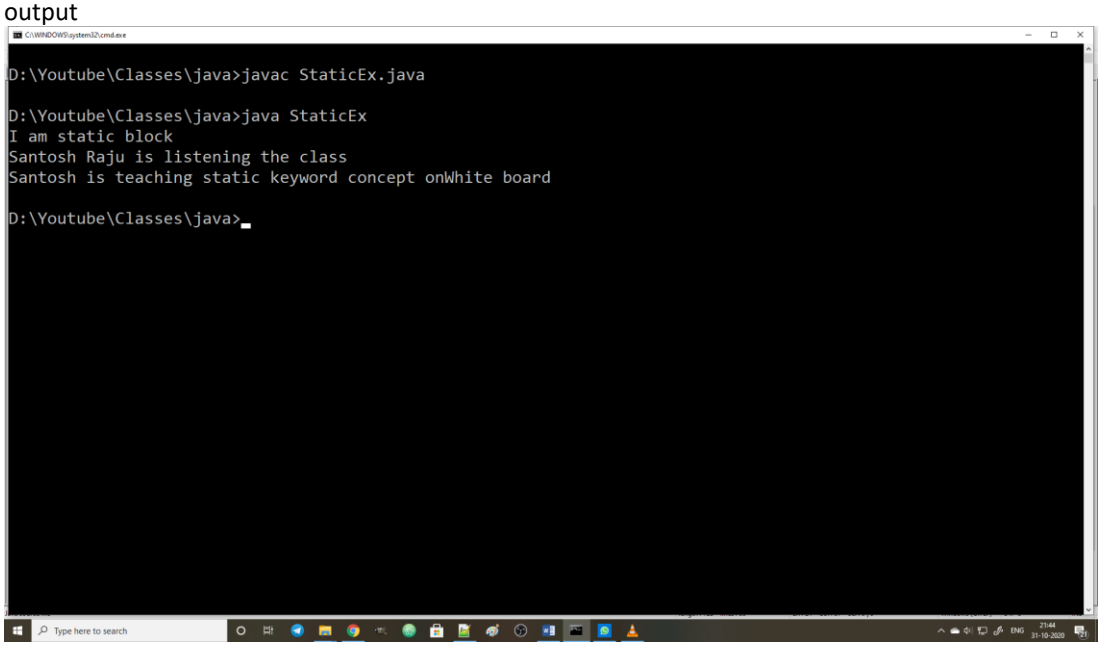
Example

```
//program to demonstrate static variables and static methods
class JavaClass
{
   String name; //instance variable
            void listen()//instance method
            {
                        System.out.println(name+" is listening the class");
            }

            static String board;//static variable
            static void teaching() //static method
   {

      System.out.println("Santosh is teaching static keyword concept on"+board+" board");
   }
}
class StaticEx
{


  public static void main(String args[])
  {
              JavaClass santosh = new JavaClass();
              santosh.name = "Santosh Raju";
              santosh.listen();

              JavaClass.board="White";
              JavaClass.teaching();

  }

static{
                        System.out.println("I am static block");
            }
}
```

In above example consider the statement that prints "I am static block" which is under static block. Observe in the following output that this statement will always be executed first.

Observe how static variable board and static method teaching () are called using class name because these are static members which are common to all objects.

output



CONSTRUCTORS IN JAVA

Constructors are also methods in which it has to meet the following characteristics

1) Its name is equivalent to its class name
2) Constructors does not have any return types

Example

```
class A
{
        A()     //constructor
        {
          //constructor code
        }

}
```

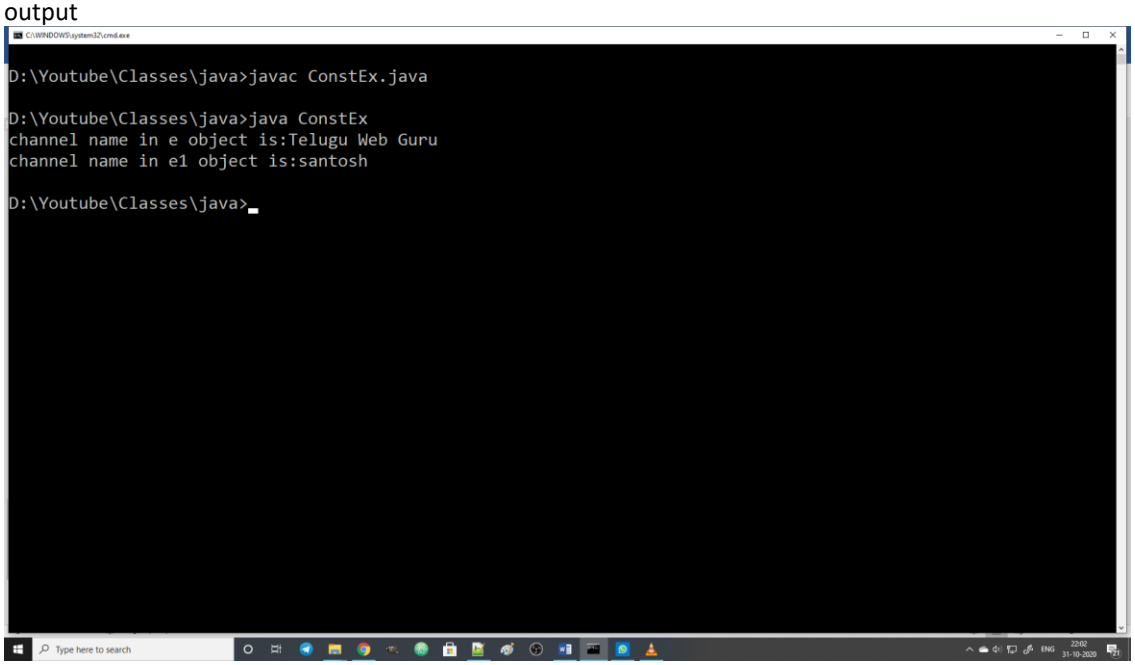
Constructors are mainly useful in initializing values to the variables.

Example

```
//program to demonstrate constructors
class Example
{
  String channel;
  //constructor overloading - same name with diff signatures
  Example()//default constructor
  {
     channel = "Telugu Web Guru";

  }
        //santosh
  Example(String channel) //parameterized constructor
  {//instance variable hiding
   this.channel = channel;//this represents current object variables
  }
}

class ConstEx
{
  public static void main(String ar[])
  {
          Example e =new Example();

          System.out.println("channel name in e object is:"+e.channel);

    Example e1 =new Example("santosh");

          System.out.println("channel name in e1 object is:"+e1.channel);
  }
}
```

Constructors are executed while creating objects only. We no need to call constructors after creating objects unlike we are calling methods

output



Constructor overloading:

In above example two constructors are overloaded with different signatures (parameters).

Two or more constructors in a class having different signatures is said to be constructor overloading.

We may invoke a particular constructor by passing the parameters that exactly matches with the constructor definition.

Note : Above example may be considered for constructor overloading too.

this KEYWORD IN JAVA

this keyword is useful in instance variable hiding.

this keyword always refers to the current object.

Example

```
class Example
{
   String channel;

    Example(String channel)
    {
        this. channel = channel;
    }
}
```

Whenever both instance variables and method / constructors temporary parameter names both are same (channel) then these temporary variables hide instance variables. This is called instance variable hiding.

for example if we write

channel = channel

then we don't know which is instance variable and which one is method / constructor's variable. So to differentiate these two and specifically to point instance variable we will use this keyword.

so the variable that is mentioned with this keyword (this.channel) is always refers to instance variable or current object variable.

INHERITANCE IN JAVA

Creating / Deriving new class from already existing class is said to be inheritance.

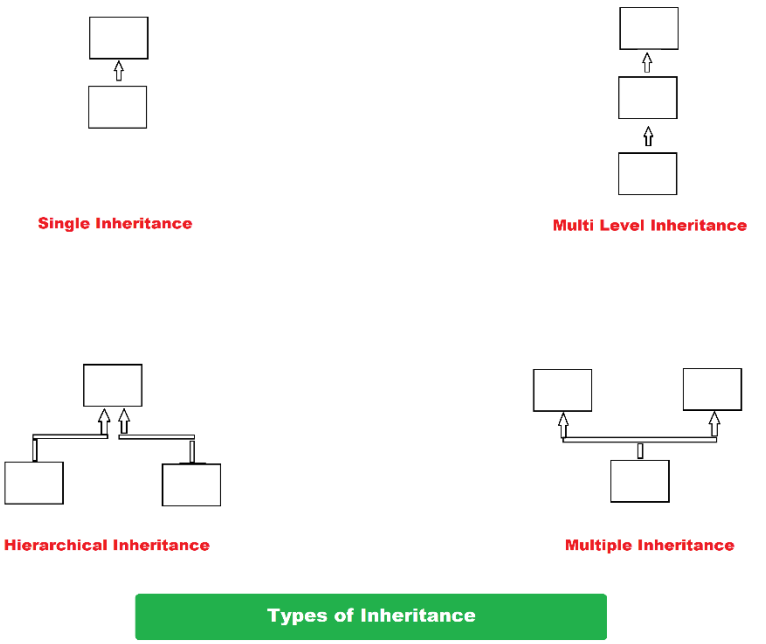
or

Object of one class acquires the properties of object of another class.

Types of Inheritance

There are four types of inheritance.

1) Single Inheritance: Only one super class and only one sub class
2) Multi-Level Inheritance: There are levels of inheritance. previous level's sub class is becoming super class for the next level's class
3) Hierarchical Inheritance: If more than one subclasses are derived from one super class
4) Multiple Inheritance: A sub class is derived from more than one super class.



Single Inheritance Example:

```
//program to demonstrate single inheritance

class A
{
  int p;

  A(){}
```

```
  A(int a)
  {
            p = a;
  }

  void display()
  {
     System.out.println("I am base class method display. My variable p value is:"+p);
  }
}


class B  extends A
{
  int q;

  B(){}

    //12    25
  B(int p,int q)
  {
            this.p = p;
    this.q = q;
  }
  void show()
  {
             System.out.println("I am derived class method show.My q value is :"+q);
  }

}


class InheritEx
{
  public static void main(String ar[])
  {
      B obj = new B(12,25);
      obj.show();

      obj.display();
  }
}
```

output

```
D:\Youtube\Classes\java>javac InheritEx.java

D:\Youtube\Classes\java>java InheritEx
I am derived class method show.My q value is :25
I am base class method display. My variable p value is:12

D:\Youtube\Classes\java>
```

In this example there is only one super class A, and only one sub class B. As class B contains both members of class A and class B, object is created for class B only.

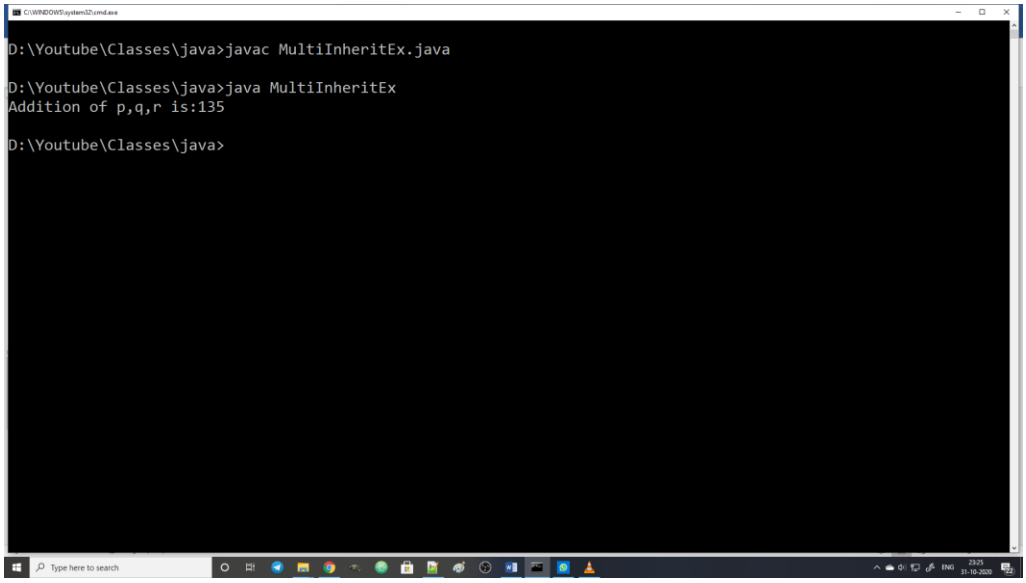
Multi-level inheritance:

In this it contains levels of inheritance that means previous level's sub class (B) is again super class for the next level class (C)

```
//program to demonstrate multilevel inheritance

class A
{
  int p;

  A(){}

  A(int p)
  {
    this.p = p;
  }

  void display()
  {
  }


}

class B extends A
{
  int q;
```

```
  B(){}

  B(int p,int q)
  {
    this.p = p;
    this.q = q;
  }

  void show()
  {
  }


}

class C extends B
{
  int r;

  C(){}
     //23   45   67
  C(int p,int q,int r)
  {
    this.p=p;
            this.q=q;
            this.r=r;

  }
  void add()
  {
           System.out.println("Addition of p,q,r is:"+(p+q+r));
  }

}



class MultiInheritEx
{
 public static void main(String ar[])
 {
          C obj = new C(23,45,67);
   obj.add();

 }

}
```

output



Hierarchical Inheritance:

Inheriting two or more classes (B, C) from a super class (A) is said to be hierarchical inheritance.

```
//program to demonstrate hierarchical inheritance

class A
{
  int p;

  A(){}
  A(int p)
  {
    this.p = p;
  }
}

class B extends A
{
 int q;

 B(){}

 B(int p,int q)
 {
   this.p = p;
          this.q = q;
 }

 void add()
  {
          System.out.println("Addition of p,q in class B is:"+(p+q));
  }
```

```
}

class C extends A
{
  int r;

  C(){}

  C(int p,int r)
  {
    this.p = p;
          this.r = r;
  }

 void add()
  {
          System.out.println("Addition of p,r in class C is:"+(p+r));
  }
}


class HierarchicalEx
{
 public static void main(String ar[])
 {

   B obj1 = new B(45,76);
          obj1.add();

   C obj2 = new C(234,657);
          obj2.add();
 }

}
```

output

```
D:\Youtube\Classes\java>javac HierarchicalEx.java

D:\Youtube\Classes\java>java HierarchicalEx
Addition of p,q in class B is:121
Addition of p,r in class C is:891

D:\Youtube\Classes\java>
```

Multiple Inheritance

Deriving a subclass from two or more super classes is said to be multiple inheritance.

But with only class concept we are unable to implement multiple inheritance because if we want to implement multiple inheritance that shown in above pic i.e., inheriting class C from classes A,B we may write the following statements which are not valid

1) class C extends A,B // comma is not allowed here so we may not write in this way
2) class C extends A extends B // this statement completely changes the meaning of it.

so we cannot implement multiple inheritance with only class concept. so we need interface to implement this multiple inheritance. Let's discuss multiple inheritance again with interface concept later.

super KEYWORD IN JAVA

super keyword is used to

1) Access super class members (variables & methods) from sub class
2) Call super class constructors from sub class constructor

Access super class members (variables & methods) from sub class

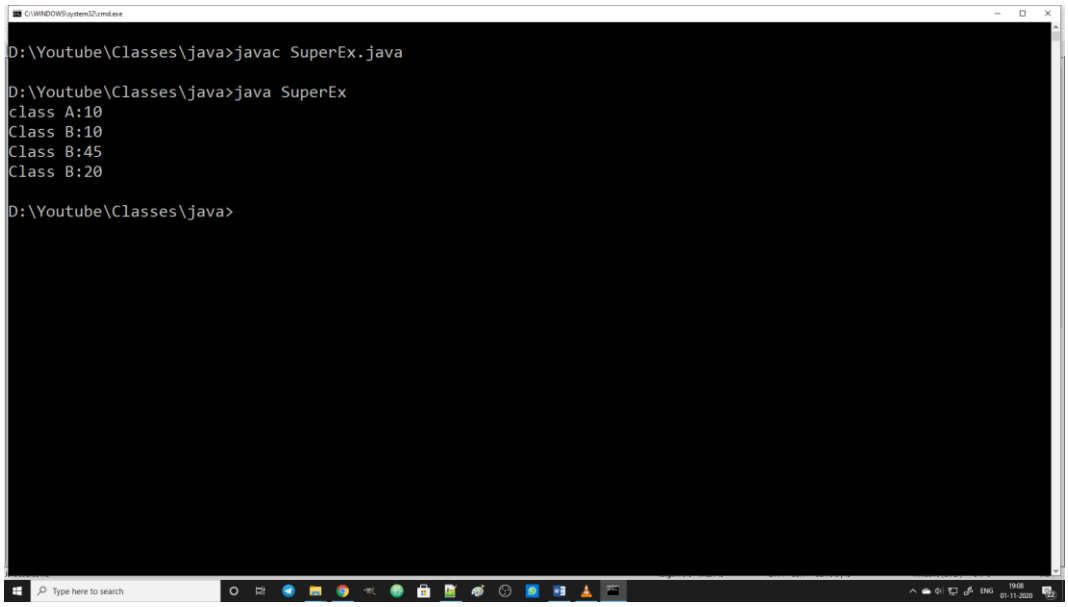
If super class members and sub class members are having same names then whenever we are calling members with subclass object then it always prefers sub class members only. So to access subclass members, we can use super keyword in sub class so that we can access super class members (super.member) as shown in example below.

Example

```
//program to demonstrate super keyword uses

class A
{
  int p = 10;       //member variables

  void show() //member methods
  {
            System.out.println("class A:"+p);
  }

}

class B extends A
{
          //p=10,show() --> from class A
          int q = 20;
          int p = 45;

          void show(){

            super.show();

            System.out.println("Class B:"+super.p);
            System.out.println("Class B:"+p);
            System.out.println("Class B:"+q);

  }


}

class SuperEx
{
          public static void main(String ar[])
          {
```

```
                B obj = new B();
                obj.show();


        }
}
```

output



In above example the sub class used super keyword to access super class members
(super.p,super.show())

Call super class constructors from sub class constructor

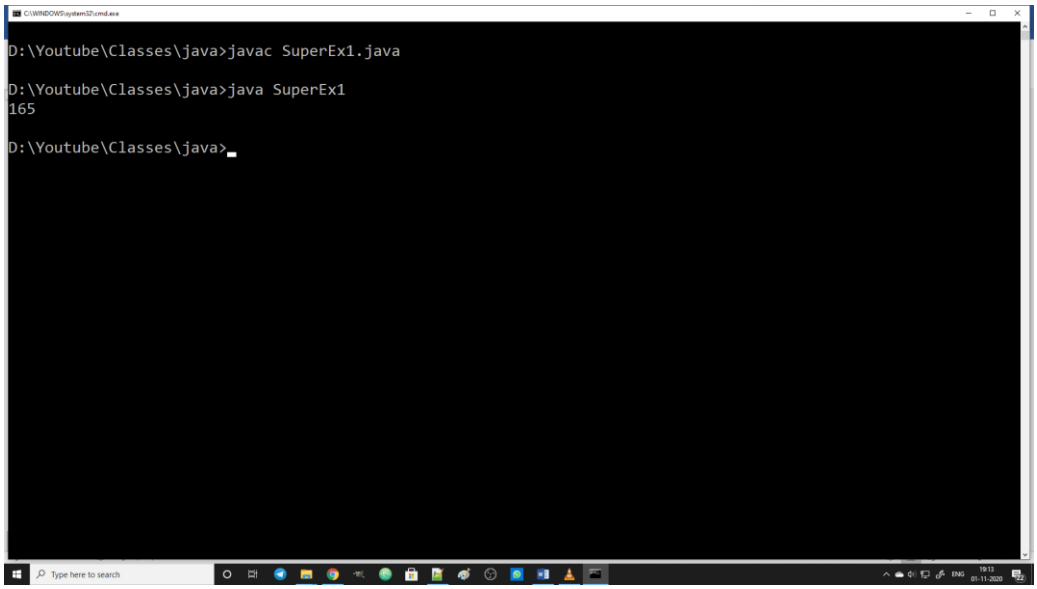
Generally, we create object for sub classes because it contains both super class members and sub class members. If we create sub class's object then sub class constructor will be executed automatically. We can access super class's constructor from sub class constructor using super(..) as shown in example below.

Example

```
//calling super class constructor from sub class constructor
class A{

   int p,q,r;

   A(){}
   A(int p,int q,int r)
        {
                this.p = p;
                this.q = q;
                this.r = r;
```

```
        }

}

class B extends A{
        //p,q,r --> from A
        int s,t;
        B(){}
        //  11    22    33    44    55
        B(int p,int q,int r,int s,int t)
        {
                super(p,q,r);//calling super class constructor
                this.s = s;
                this.t = t;
        }
        void add()
        {
          System.out.println(p+q+r+s+t);
        }
}

class SuperEx1{
public static void main(String ar[])
{
  B obj = new B(11,22,33,44,55);
  obj.add();

}
}
```

output



observe the line super(p,q,r); which calls super class's constructor from sub class's constructor.

ABSTRACT CLASSES IN JAVA

A class is said to be an abstract class in which it's having at least one abstract method.

A method that is just declared without definition is said to be an abstract method.

```
abstract class ex
{
     void display()    // concrete method
     {
        System.out.println("hello");
      }

     abstract void show();          //abstract method
}
```

In above example display method is declared with definition (method code). That's why this is called as concrete method.

where as show method is called as abstract method because there is no method definition (code) provided with its declaration. All these abstract methods must be implemented in subclasses as it is not done here.

Because class ex contains at least one abstract method show() , class ex became abstract class.

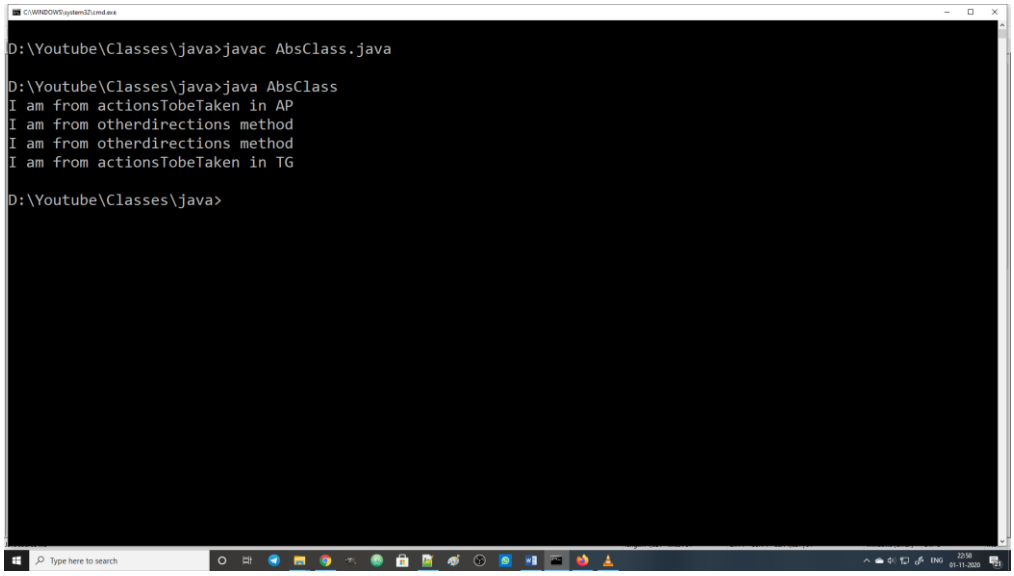
Note: As some of the methods in abstract classes are not having method definition, we cannot create objects for abstract classes.

Example

```
//program to demonstrate abstract classes in java
abstract class CGovt
{
        void otherdirections()//concrete methods
  {
    System.out.println("I am from otherdirections method");
  }
        abstract void actionsTobeTaken(); //abstract method
        abstract void actionsTobeTaken1(); //abstract method
}
class AP extends CGovt
{
        void actionsTobeTaken1()
        {
                System.out.println("I am from actionsTobeTaken1 in AP");
        }
        void actionsTobeTaken()
        {
                System.out.println("I am from actionsTobeTaken in AP");
        }
```

```
}
class TG extends CGovt
{
	void actionsTobeTaken1()
	{
		System.out.println("I am from actionsTobeTaken1 in TG");
	}
	void actionsTobeTaken()
	{
		System.out.println("I am from actionsTobeTaken in TG");
	}
}


class AbsClass
{
	public static void main(String ar[])
	{
		//CGovt cg = new CGovt();

		AP ap = new AP();
		ap.actionsTobeTaken();
		ap.otherdirections();

		TG tg = new TG();
		tg.otherdirections();
		tg.actionsTobeTaken();

	}
}
```

Output



INTERFACES IN JAVA

An Interface is similar to a class which contains collection of variables and methods where all the methods in interface are by default abstract methods and all variables in methods are static and final variables.

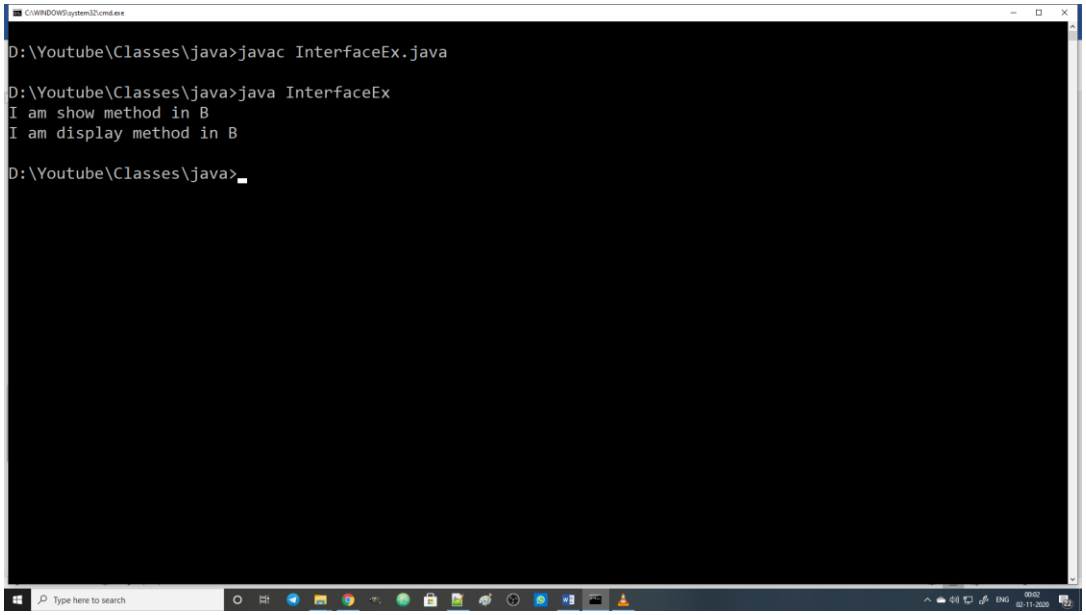
We no need to specify any method or interface with abstract keyword because by default these are abstract. All the methods in interface are implemented in sub classes of this interface.

As no methods are having definition we cannot create objects for interfaces.

Example

```
//program to demonstrate interfaces in java

interface A
{
	int p = 10; //static and final variables

	//abstract methods
	void show();
	void display();

}

class B implements A
{
   public void show()
	{
		System.out.println("I am show method in B");
	}
	public void display()
	{
		System.out.println("I am display method in B");
	}

}

class InterfaceEx
{
  public static void main(String ar[])
  {
	B ob = new B();
	ob.show();
	ob.display();
  }
}
```

Output



In above example interface A contains two methods show and display. These are by default abstract methods, so we implemented the code of these methods in implementation class B. While implementing interface's methods in subclasses we must declare it with public keyword.

Note : While inheriting a subclass from an interface we need to use “implements” keyword instead of “extends”.

Interfaces can be extended

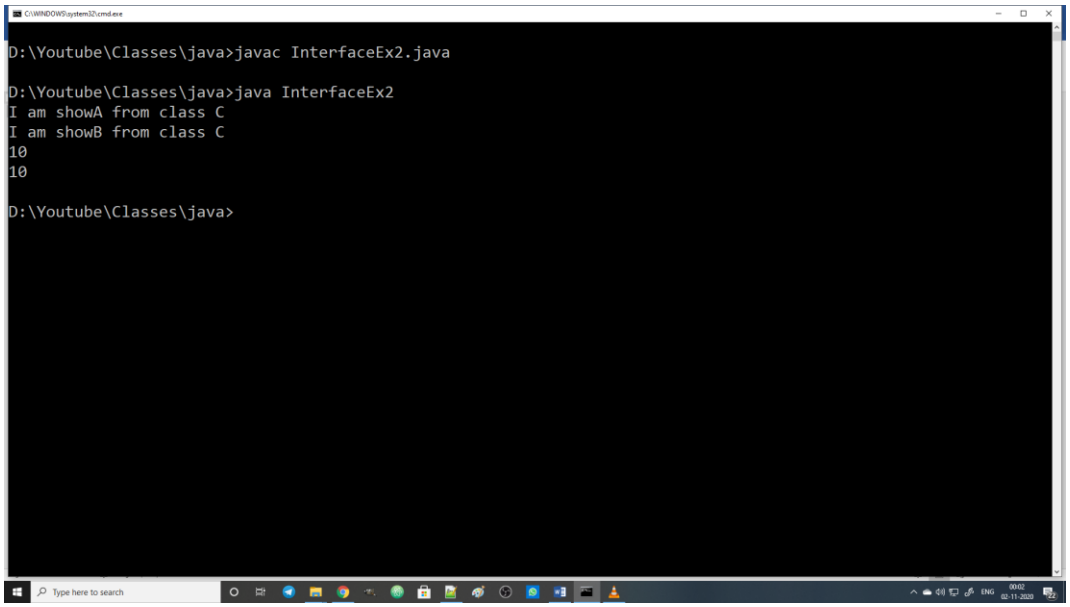
While inheriting an interface from other interface we use extends keyword. In below example interface B is inherited from interface A. As these two are interfaces we used extends keyword here.

Example

```
//program to demonstrate interfaces
interface A{
        int p = 10; //static

  void showA();
}

interface B extends A
{
         //int p = 10;
        //showA from A
        void showB();
}

class C implements B
{
        //int p = 10;
        public void showB()
        {
```

```
            System.out.println("I am showB from class C");
        }

        public void showA()
        {
            System.out.println("I am showA from class C");
        }
}


class InterfaceEx2
{
  public static void main(String ar[])
  {
            C ob = new C();
            ob.showA();
            ob.showB();
            System.out.println(ob.p);

             System.out.println(A.p);

  }

}
```

output



Multiple Inheritance using interfaces :

Now let us see how to implement multiple inheritance in java with the help of interfaces. While implementing multiple inheritance with interfaces, we should remember that while declaring super classes all the entities must be interfaces or at most one class is allowed.

For example to write a statement that inherits class C from interfaces A,B then we can write the statement as

```
class C implements A,B
{

}
```

Here comma is allowed in case of interfaces.
If we want to declare some of the entities as classes in super class level then only one class is allowed and then only we can implement multiple inheritance.

for example to inherit class C from class A and interface B we will write the statement as

```
class C extends A implements B
{

}
```

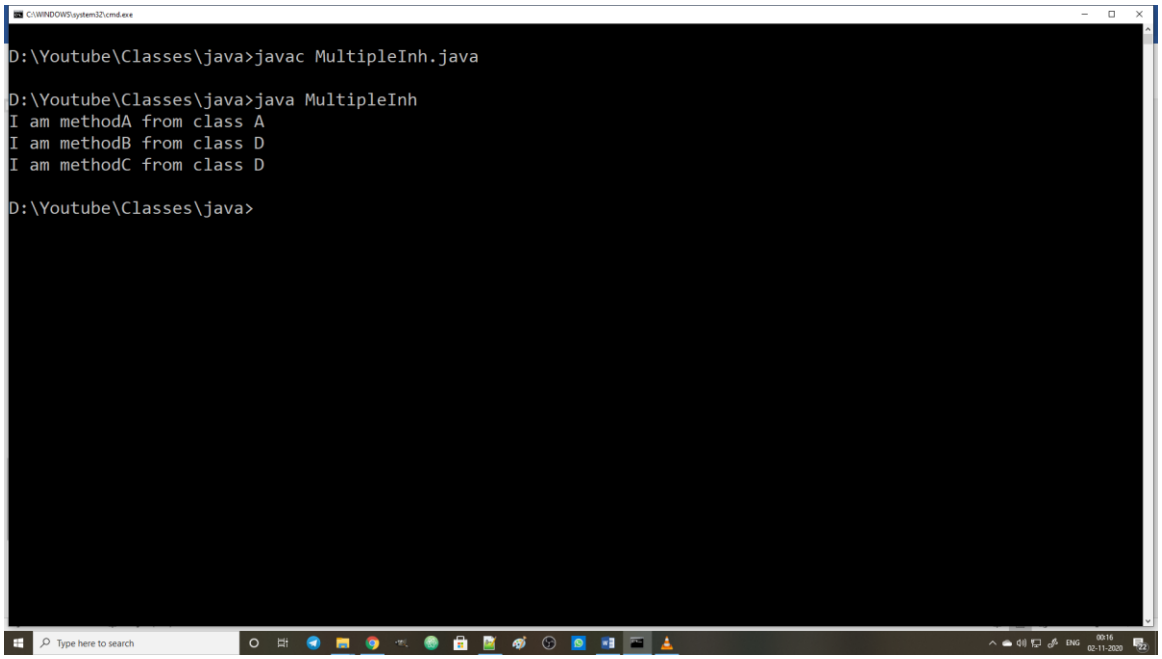
In this case first priority should be given to classes (extends).

Example

```
//program to demonstrate multiple inheritance in java using interfaces
class A
{
  void methodA()
  {
    System.out.println("I am methodA from class A");
  }
}
interface B
{
   void methodB();
}
interface C
{
        void methodC();
}
class D extends A implements B,C
{
        //methodA from Class A

   public void methodB()
        {
```

```
                System.out.println("I am methodB from class D");
        }

         public void methodC()
        {
                System.out.println("I am methodC from class D");
        }
}

class MultipleInh
{
  public static void main(String ar[])
  {
      D ob = new D();
                ob.methodA();
                ob.methodB();
                ob.methodC();

  }
}
```

output



PACKAGES IN JAVA

A package represents a directory that contains related group of classes and interfaces.

Advantages of packages:

1) All the related classes and interfaces are at one place so we can easily access all of these classes and interfaces
2) There will be no name collisions.

How to create a package ?

we can create a package by using package keyword as follows

```
package packagename;
```

How to insert classes into the package ?

Whatever classes we declared under package statement those all classes will be automatically added under package.

Example

```
package twg;
class A{

}

class B{

}
```

In above example as class A, class B are declared under package statement these two classes were automatically added under package “twg”.

Note: class in a file must be declared as public and we must save our program with that file name. if it is main class then it will automatically be public so no need to declare as public explicitly.

How to compile and Run package programs:

packages are nothing but directories so we need to create a folder manually with the package name and we need to save the java program in that directory.

or we can use the following command that will create a folder automatically with the package name and places the class file into that folder.

```
javac -d . programname.java
```

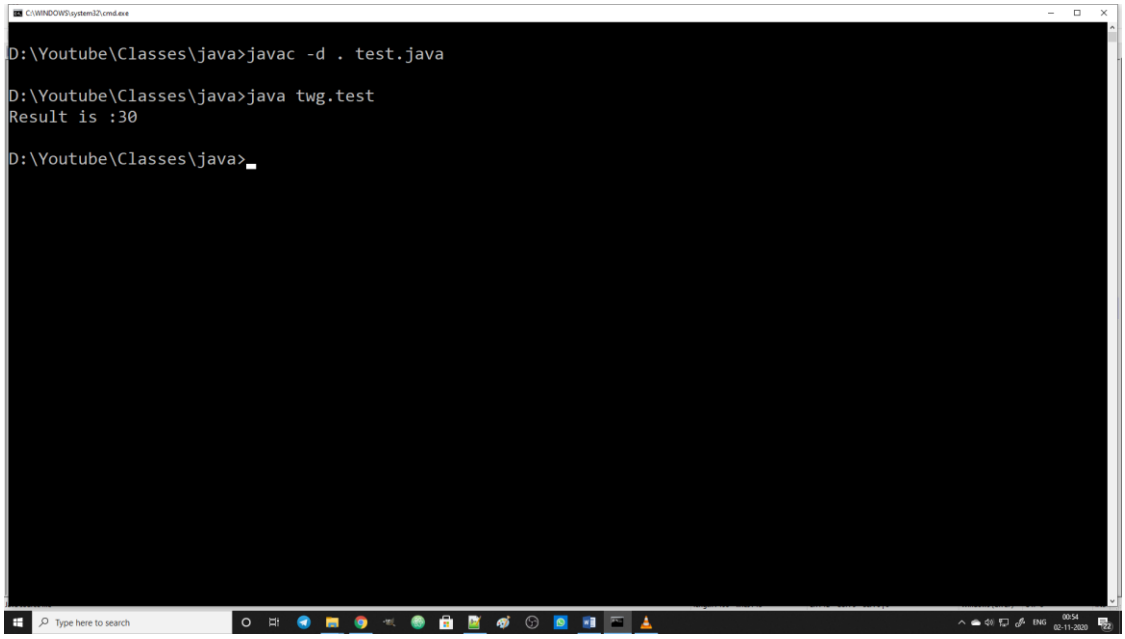
while executing package program we need to mention package name also as follows

```
java packagename.programname
```

Example Program

```
//program to demonstrate packages
package twg;

class test
{
  public static void main(String ar[])
  {
          int a=10;
          int b=20;


           System.out.println("Result is :"+(a+b));
  }

}
```

output



Accessing members of classes & interfaces from other packages :

To access the members (variables / methods) of any class or interface from other packages, then we need to import the particular class or interface or we need to import the entire package. then only we can access the members.

importing one class/interface :

import packagename.class/interface-name

importing the whole package with * :

import packagename.*

Example

```
package p1;
class A
{
    int p = 10;

}
```

if we want to access the variable p of class A in other package classes then we need to import package p1 as follows.

```
package p2
import p1.A

class B
{
   B()
  {
     A ob = new A();
     System.out.println(ob.p);
  }
}
```

As we need only class A in package p1 we imported package p1 only. If we want to access more than one class/interface from p1 then we may write separate import statements or we may import entire package at a time by using * . We can create and insert our classes and interfaces in sub packages also.

```
package p1.sub1.sub2
class A{

}
```

here p1 package will be created and within p1 package, a sub package sub1 is created and within sub1 another sub package sub2 is created and within sub2, class A will be added.

you can find more examples on packages in the next section : access modifiers

ACCESS MODIFIERS IN JAVA

By using access specifiers or access modifiers we can control the access of member variables and member methods.

We have 4 types of access specifiers in java

1) public : public members can be accessible from any class and any package
2) private : private members are accessible in the class where these are declared only
3) protected : only non sub classes of other packages are blocked to access these members
4) No Modifier (Default Modifier) : other package classes are not allowed to access these members.

The following image will give clear detail about how member variable of a class can be accessed in other classes that are related to the same or different package.

	public	private	protected	Default
Same Class	✓	✓	✓	✓
Same Package Sub Class	✓	✗	✓	✓
Same Package Non Subclass	✓	✗	✓	✓
Other Package Sub Class	✓	✗	✓	✗
Other Package Non Sub Class	✓	✗	✗	✗

Now let us see the same in examples.

Following example will explain how variables that are declared in a class can be accessible with in a class

Example of same class :

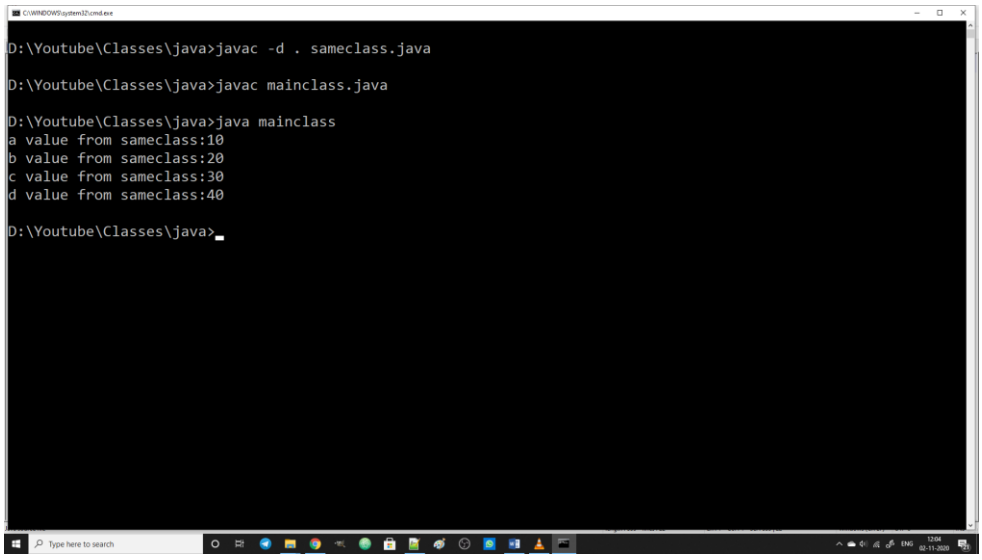
```
package p1;

public class sameclass{

   private int a = 10;
        int b = 20;
        protected int c = 30;
        public int d = 40;
```

```
  public sameclass()
        {
           System.out.println("a value from sameclass:"+a);
                System.out.println("b value from sameclass:"+b);
                System.out.println("c value from sameclass:"+c);
                System.out.println("d value from sameclass:"+d);
        }

}
```

main class

```
import p1.sameclass;
class mainclass
{
  public static void main(String ar[])
  {
    sameclass sc = new sameclass();

  }
}
```

output



As shown in output , variables with all access specifiers can be accessible within the class where these members are declared.
Following example shows how other subclasses in the same package can access the members of a class

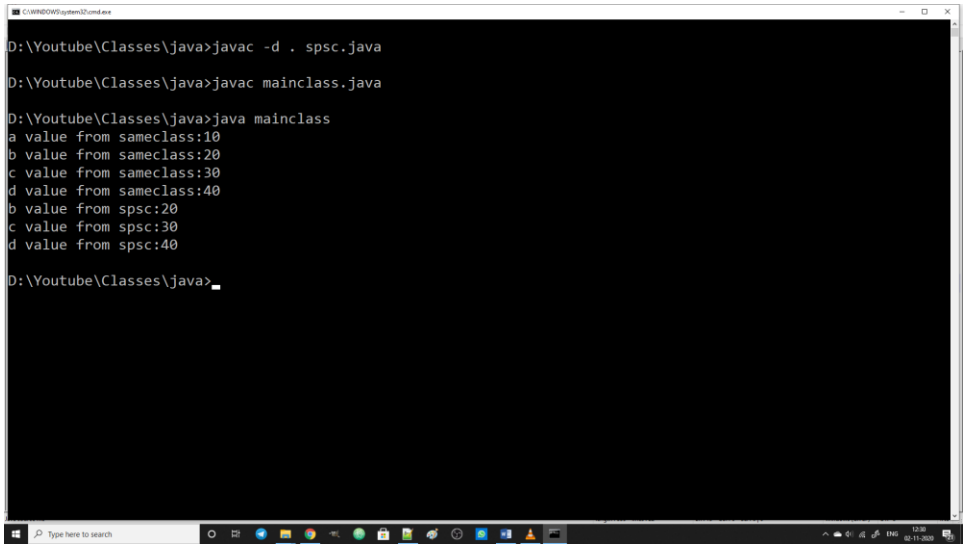
Example of same package sub class

```
package p1;
```

```
public class spsc extends sameclass
{
        public spsc()
        {
          // System.out.println("a value from spsc:"+a);
                System.out.println("b value from spsc:"+b);
                System.out.println("c value from spsc:"+c);
                System.out.println("d value from spsc:"+d);
        }
}
```

Main class

```
import p1.spsc;
class mainclass
{
  public static void main(String ar[])
  {
          spsc ob1 = new spsc();
  }
}
```



As shown in above example subclasses of the same package can access all the members except private members.
The following example shows how other classes of the same package can access the members of a class.

Example of same package non-sub class

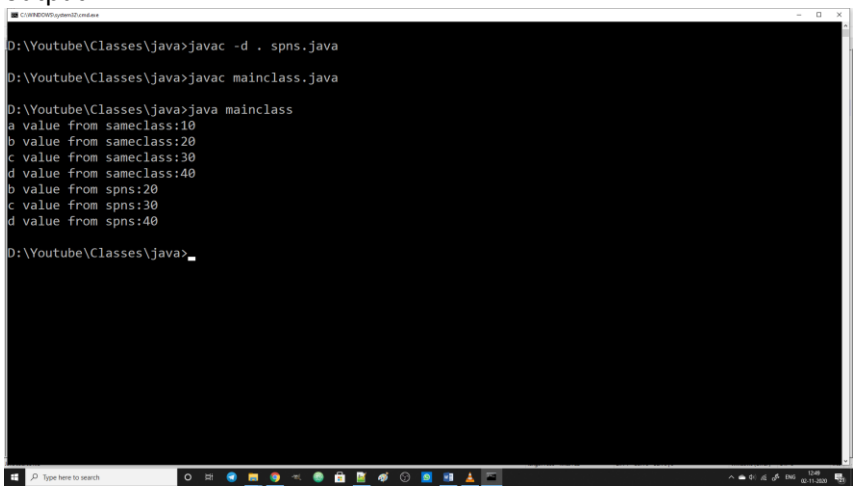
```
package p1;

public class spns
{
        public spns()
        {
          sameclass sc = new sameclass();
```

```
    // System.out.println("a value from spns:"+sc.a);
            System.out.println("b value from spns:"+sc.b);
            System.out.println("c value from spns:"+sc.c);
     System.out.println("d value from spns:"+sc.d);

          }
}
```

Main class

```
import p1.spns;
class mainclass
{
  public static void main(String ar[])
  {
          spns ob = new spns();
  }
}
```

output



observe here that non subclasses of the same package also can access all the members except private.
The following example shows how subclasses of other packages can access the members of a class

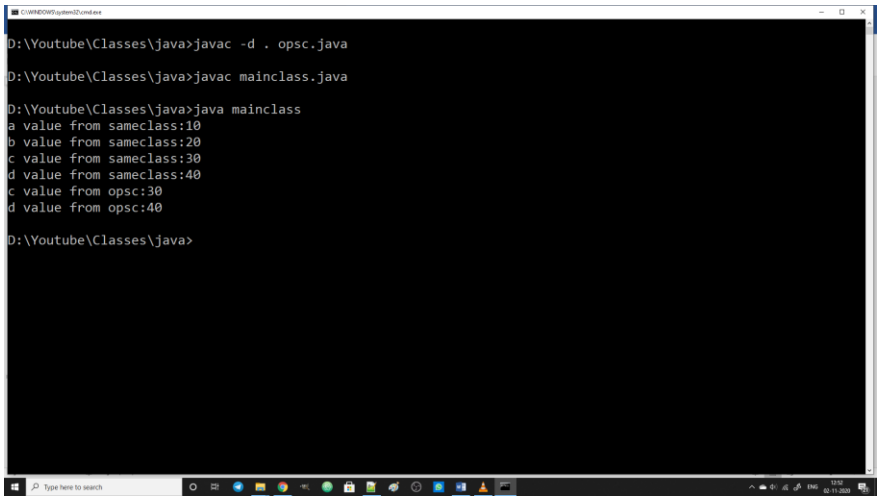
Example of other package sub class

```
package p2;

import p1.sameclass;

public class opsc extends sameclass
{
          public opsc()
          {
           //  System.out.println("a value from opsc:"+a);
                  //System.out.println("b value from opsc:"+b);
                  System.out.println("c value from opsc:"+c);
                  System.out.println("d value from opsc:"+d);
```

```
        }

}
```

main class

```
import p2.opsc;
class mainclass
{
  public static void main(String ar[])
  {
        opsc ob = new opsc();
  }
}
```



observe here that sub classes of other packages can access protected & public members only.

The following example shows how non-sub classes from other packages can access the members of a class

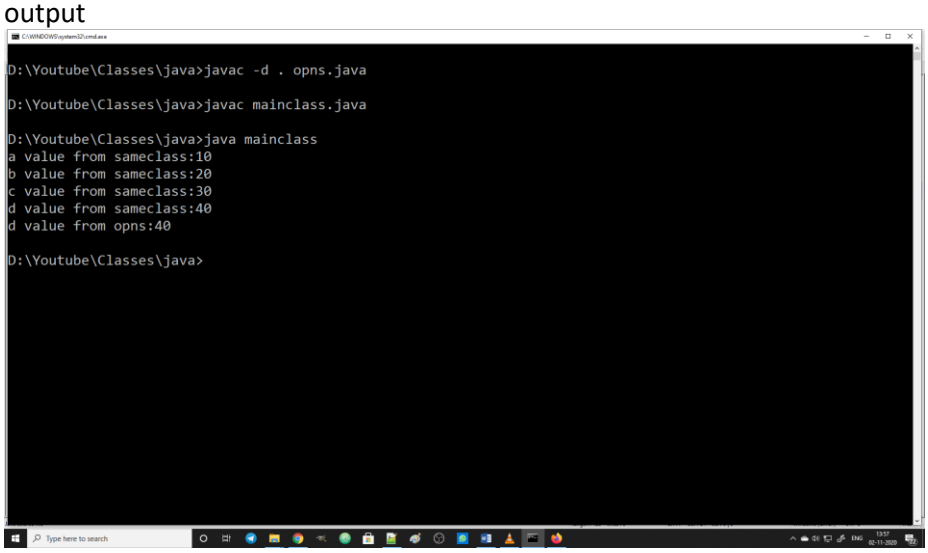
Example of another package non-sub class

```
package p3;

import p1.sameclass;

public class opns
{
        public opns()
        {
     sameclass ob = new sameclass();
          //System.out.println("a value from opns:"+ob.a);
          //System.out.println("b value from opns:"+ob.b);
          //System.out.println("c value from opns:"+ob.c);
    System.out.println("d value from opns:"+ob.d);
        }
}
```

Example of Main class

```
import p3.opns;
class mainclass
{
  public static void main(String ar[])
  {
     opns ob = new opns();

  }
}
```

output



In above example other package non sub class is allowed to access only public members of a class.

final KEYWORD IN JAVA

final keyword is used to

1) declare constant variables
2) prevent method overriding
3) prevent inheritance

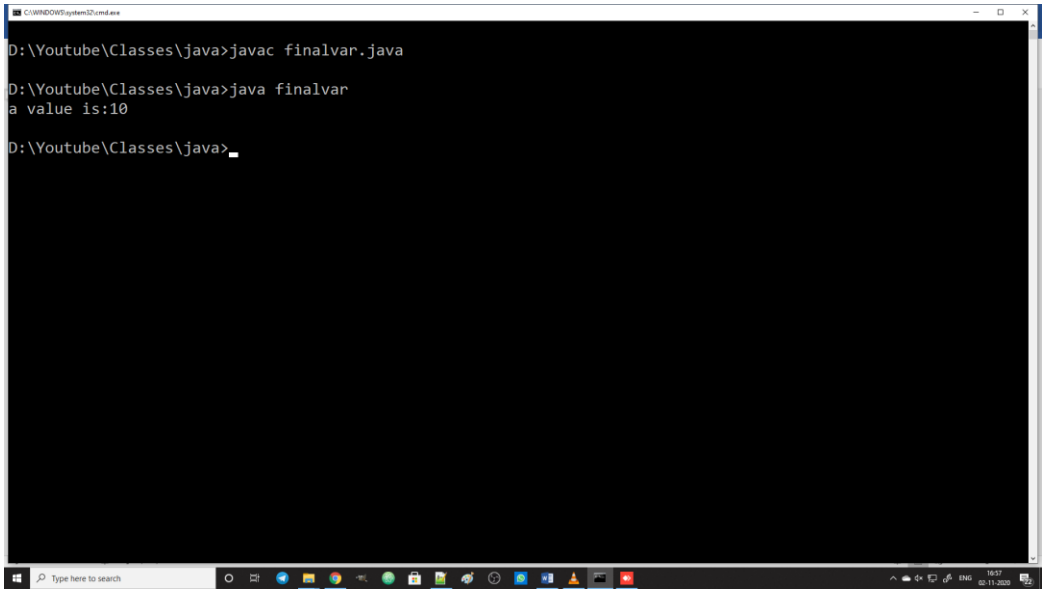
final keyword with variables - becomes constants :

If we declare final keyword with variables, they become constants and system will not allow us to change the value of it

Example

```
class finalvar
{
        public static void main(String ar[])
        {
          final int a=10;//constant
          System.out.println("a value is:"+a);
          //a=11; error

        }

}
```

output



Try to run above program by removing comment to the line a=11 which returns error because once a variable is declared with final keyword we can't change it's value as it becomes constant.

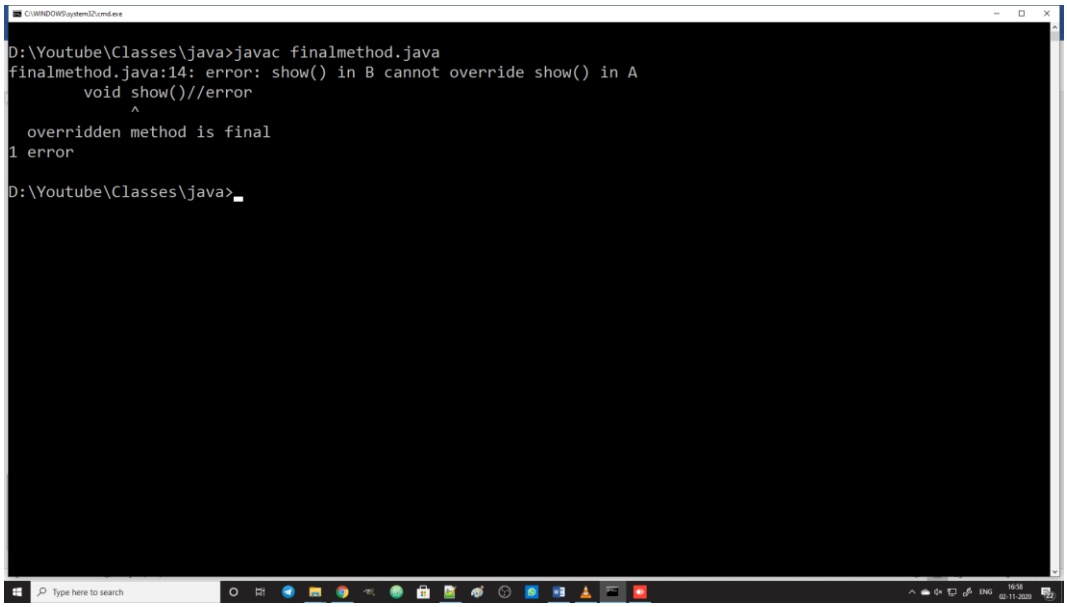
final keyword with methods – prevents method overriding

If we declare methods as final then subclasses are not allowed to override this method. Below example returns error because we can't override a method once it is declared as final.

Example

```
//final with methods to prevent overriding

class A
{
  final void show()
  {
    System.out.println("I am show method from A");
  }
}
class B extends A
{
        //show from A
        void show()//error
        {
          System.out.println("I am show method from B");
        }

}

class finalmethod
{
 public static void main(String ar[])
 {
        B ob = new B();
          ob.show();
 }
}
```

output



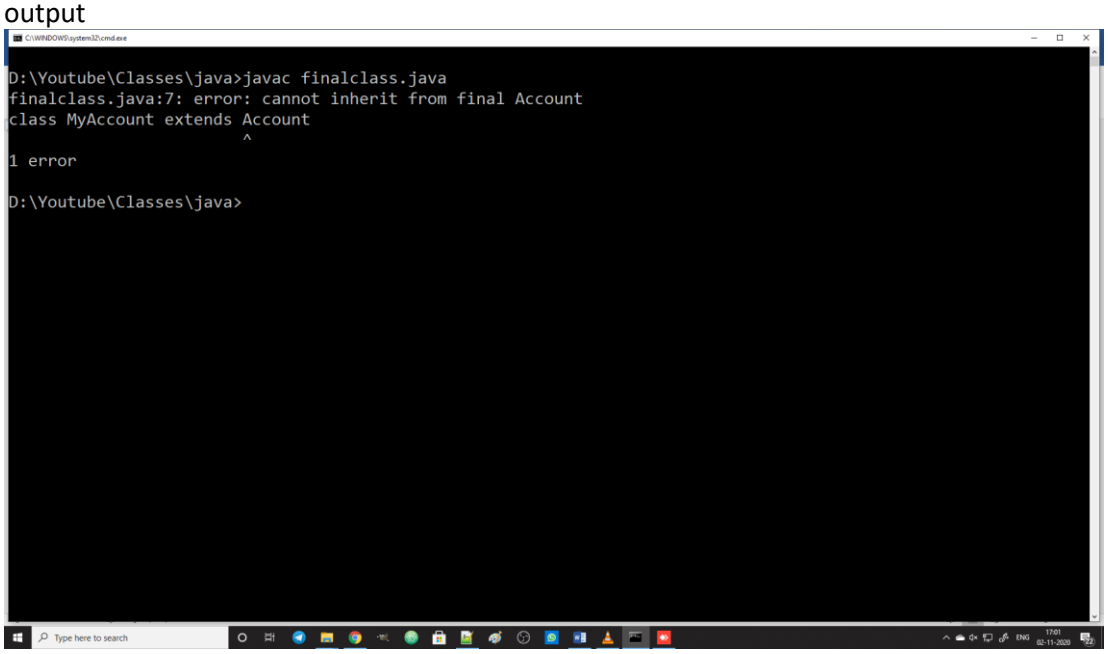
final keyword with classes – prevents inheritance

If we declare class as final then we are not allowed to inherit the class.

Example

```
//program to demonstrate final with classes
final class Account
{
  int balance = 2000;
}

class MyAccount extends Account
{
        //balance

}

class finalclass
{
        public static void main(String ar[])
        {
                MyAccount obj = new MyAccount();
                obj.balance=50000;
                System.out.println(obj.balance);

        }
}
```

output



Above example returns error because class Account is final and if any class tries to inherit from the final class then java system immediately returns error.

EXCEPTION HANDLING IN JAVA

Errors in Java Program

There are 3 types of errors

1) compile time errors : These are syntactical errors found in the code

   Examples:
   1) if we write system.out.println() instead of System.out.println() it will generates a compile time error.
   2) if we forgot semicolon at the end of any statement in our program

2) runtime errors : These errors will occur due to inefficiency of the computer system (ex : insufficient memory)

   Example :

   if we forgot to write string array inside main method as follows

```
class A
{
  public static void main()
   {

        System.out.println("Welcome");

   }
}
```

   for example in the above program we forgotten to include String ar[] inside main. syntactically it is a method without any arguments in a class so it can't be a compile time error but java system unable execute this program because it can't find proper main method to start the program execution.

3) logical errors: These errors depict flaws in the logic of the program. It may give some output but not as expected.

```
void add()
{
   int a =10;
   int b=20;
   System.out.println(a*b);

}
```

In above program instead of using + operator we used * which generates wrong output (logical Error)

Exceptions:

Exceptions are Runtime errors. Whenever an exception occurs then the java system stops the program's execution. so, exception is defined as

"An Exception is an abnormal condition which stops the normal execution of a program ".

Exceptions are Two types:

1) Checked Exceptions
2) Unchecked Exceptions

All exceptions are occurred at runtime only. But some are detected at compile time and some are detected at runtime. The exceptions that are checked at compile time are said to be checked exceptions and the exceptions that are checked at runtime are said to be unchecked exceptions.

All these exception related classes and interfaces are belongs to java.lang package

The following classes are top in the errors and exceptions class hierarchy.

Throwable: Represents all errors and exceptions in java
Exception: This is super class of all exceptions in java.

Exception Handling:

We can handle exceptions in java by using 5 keywords

1) try
2) catch
3) throw
4) throws
5) finally

try: try keyword will identify the exceptions within the statements in its block. Remember that we must provide either catch or finally block as a pair to try block.

catch: catch will handle the exceptions that are identified by try. Generally, we provide multiple catch blocks (each catch block represents an exception). The catch that matches with the exception identified will be invoked and it handles the exception by executing the code within it.

throw: to manually create and throw exception objects. Generally, we use throw for user defined exceptions. Because predefined exceptions are automatically identified (no need to throw manually).

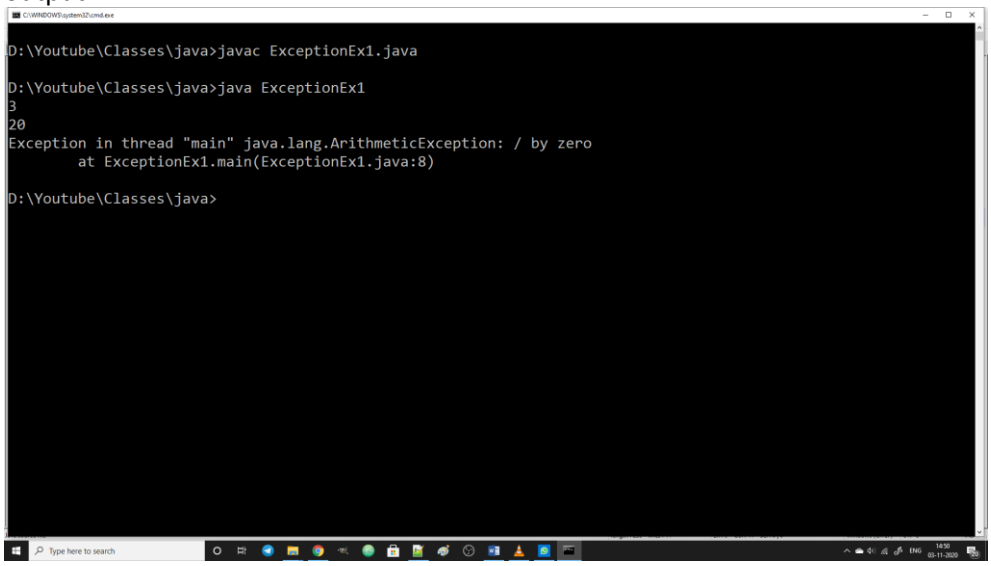
throws: We can handover the exception handling part to the system by using throws keyword

finally: The statements in the finally block will always be executed regardless of the existence of error or not.

Example

```
//program to demonstrate exception handling in java
class ExceptionEx1
{
		public static void main(String ar[])
		{
				System.out.println(1+2);
				System.out.println(4*5);
				System.out.println(6/0);
				System.out.println(7*4);
		}

}
```

output



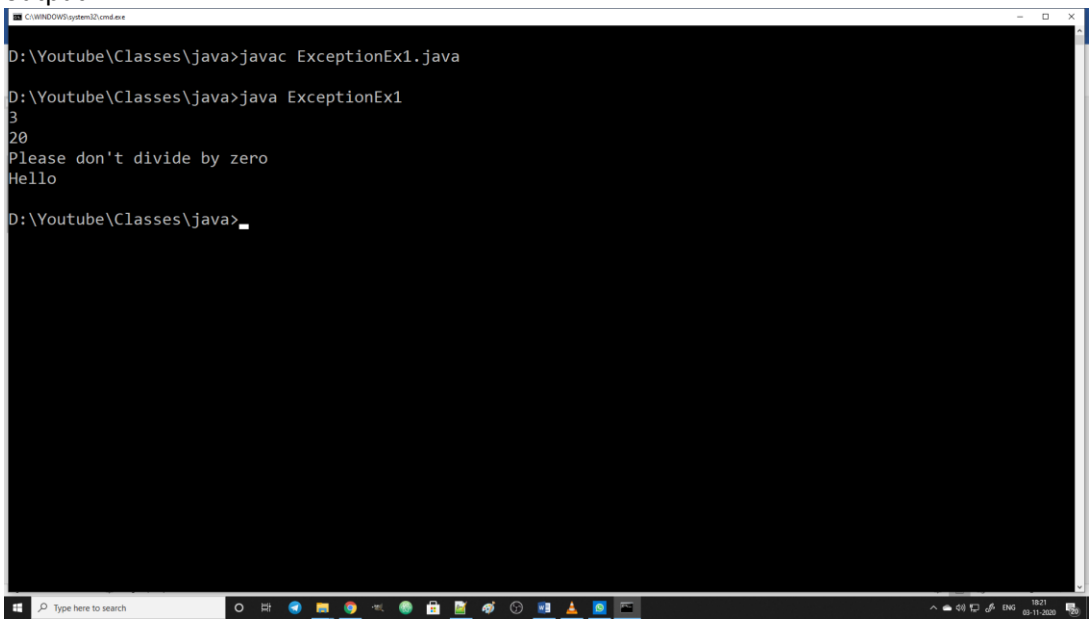
consider the above example. In programming anything divided by zero is an error. it is called as ArithmeticException (/ by zero). This is one of the Unchecked Exceptions.

Because of this exception, forth line that multiplies 7*4 is also not executed. Now we need to handle this exception by using try catch blocks.

```
//program to demonstrate exception handling in java
class ExceptionEx1
{
		public static void main(String ar[])
		{
			try{
				System.out.println(1+2);
				System.out.println(4*5);
				System.out.println(6/0);
				System.out.println(7*4);
			  }
```

```
                    catch(ArithmeticException e)
                {
                        System.out.println("Please don't divide by zero");

                }
                catch(IndexOutOfBoundsException e)
                {

                }

                finally{
                 System.out.println("Hello");
                }


        }

}
```

output



Now check above code. We provided try catch blocks now. try block will identify the exception and throw the exception object to the corresponding catch block.

now catch block will handle the exception.

One important point to be noted here is, within try block all the next statements to the line that contains error will never executed.

So, the line System.out.println(7*4); will never be executed.

So, the statements that are compulsorily executed must be within finally block.

The following are the list of checked exceptions and unchecked exceptions.

Unchecked Exceptions	Checked Exceptions
ArithmeticException IndexOutOfBoundsException ArrayIndexOutOfBoundsException StringIndexOutOfBoundsException NullPointerException	ClassNotFoundException InstantiationException NoSuchMethodException IOException FileNotFoundException InterruptedIOException

Unchecked Exceptions

ArithmeticException

Thrown when an exceptional arithmetic condition has occurred. For example, an integer "divide by zero" throws an instance of this class.

ArrayIndexOutOfBoundsException

Thrown to indicate that an array has been accessed with an illegal index. The index is either negative or greater than or equal to the size of the array.

StringIndexOutOfBoundsException

Thrown by String methods to indicate that an index is either negative or greater than the size of the string. For some methods such as the charAt method, this exception also is thrown when the index is equal to the size of the string.

NullPointerException

Thrown when an application attempts to use null in a case where an object is required.

These include:

Calling the instance method of a null object.

Accessing or modifying the field of a null object.

Taking the length of null as if it were an array.

Accessing or modifying the slots of null as if it were an array.

Throwing null as if it were a Throwable value.

Unchecked Exceptions

ClassNotFoundException

Thrown when an application tries to load in a class through its string name using:

- The forName method in class Class.
- The findSystemClass method in class ClassLoader .
- The loadClass method in class ClassLoader.

InstantiationException

Thrown when an application tries to create an instance of a class using the newInstance method in class Class, but the specified class object cannot be instantiated. The instantiation can fail for a variety of reasons including but not limited to:

- the class object represents an abstract class, an interface, an array class, a primitive type, or void
- the class has no nullary constructor

NoSuchMethodException

Thrown when a particular method cannot be found.

FileNotFoundException

File that we are searching for is not found

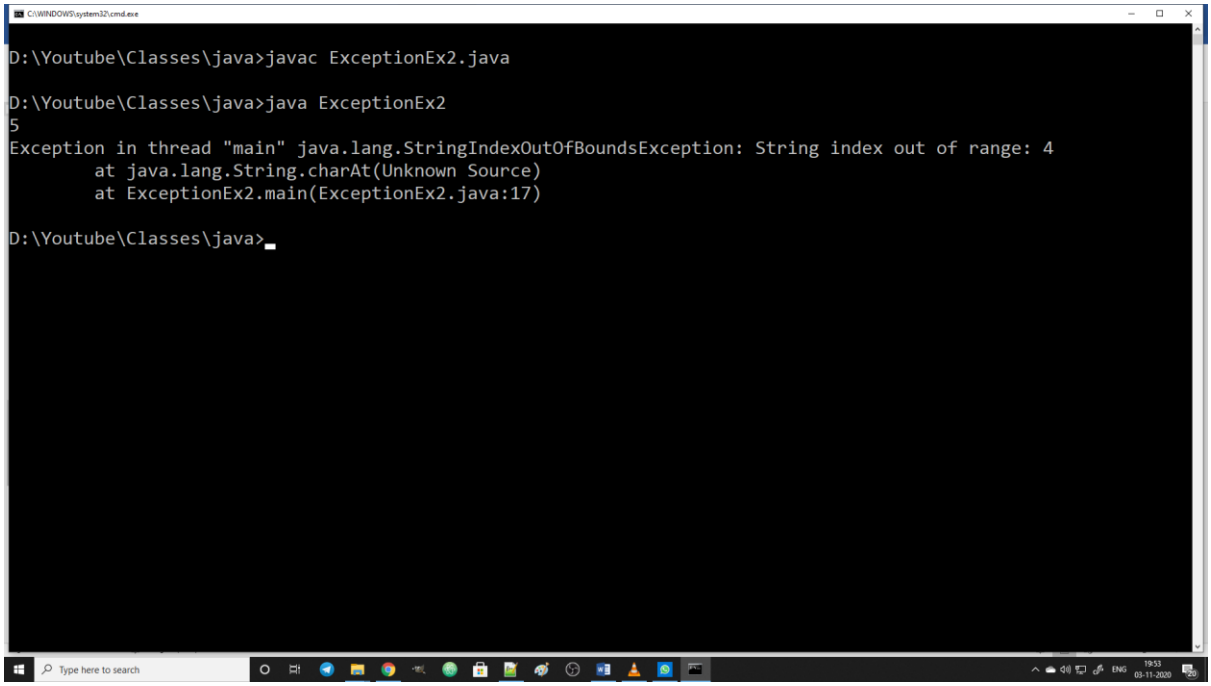
InterruptedIOException

Signals that an I/O operation has been interrupted. An InterruptedIOException is thrown to indicate that an input or output transfer has been terminated because the thread performing it was interrupted. The field bytesTransferred indicates how many bytes were successfully transferred before the interruption occurred.

The following program explains ArrayIndexOutOfBoundsException, StringIndexOutOfBoundsException in java.

```
//program to demonstrate ArrayIndexOutOfBoundsException, StringIndexOutOfBoundsException in java

class A
{
  int marks[]={10,9,5,6,7};
  String name="twg";
}
```

```
class ExceptionEx2
{
		public static void main(String ar[])
		{
			A ob = new A();

			System.out.println(ob.marks[2]);
			//System.out.println(ob.marks[5]);
			System.out.println(ob.name.charAt(4));

		}

}
```



If we are trying to access an element in an array that does not exists then ArrayIndexOutOfBoundsException will occur.

In the above example only marks[0] to marks[4] elements are existed but we are trying to access marks[5] which returns ArrayIndexOutOfBoundsException.
In the same way if we are trying to access a character at a particular index in a string that does not exists then we will get StringIndexOutOfBoundsException.

in the above example charAt(4) does not exists so it returns StringIndexOutOfBoundsException.

User Defined Exceptions

We can create our own user defined exception classes and use them as per our requirements in our programs.

To create user defined exceptions

1) create a class that extends Exception class

```
class MyException extends Exception
{
```

2) We may define constructor / toString method in it

```
MyException(){
   System.out.println("Our own message");
}
```

or

```
public string toString(){
      System.out.println("Our own message");

}
```

3) Now as per our conditions, throw our own exception by using throw keyword

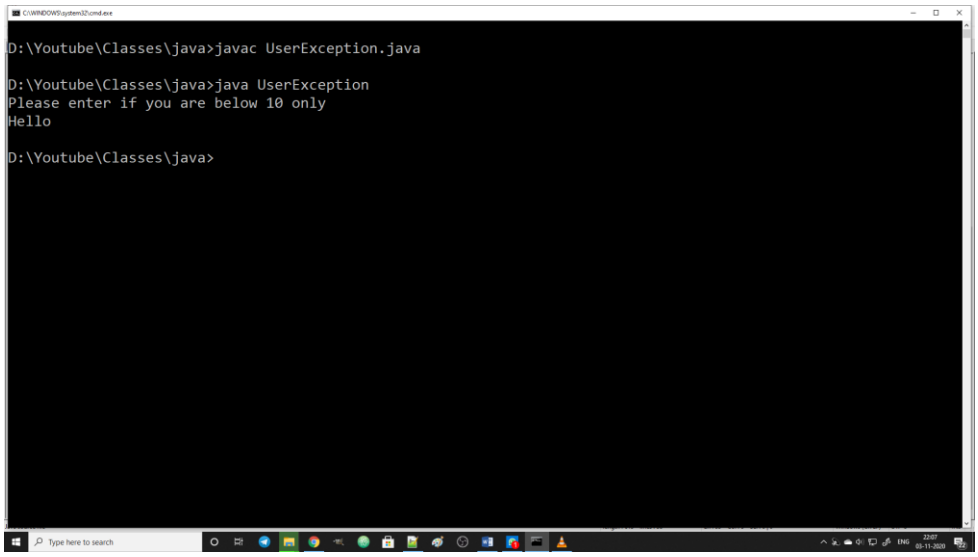
Consider the example below:

Following example will raise our own exception whenever variable value exceeds 10. This is user defined exception. So, we may not raise predefined exceptions for our own conditions. So, we defined our own exception class MyException and it will be raised from our other classes which fails our condition. As user defined exceptions never identified and throw exception objects automatically, throw keyword is used to throw the exception object.

```
//program to demonstrate user defined exceptions

class MyException extends Exception
{
        MyException(){
                System.out.println("This Competititon is for 10 years below children only.");
        }

        MyException(String str){
                super(str);
        }


}
```

```
class UserException
{
  public static void main(String ar[]) throws Exception
  {
    int age=11;
          try{
          if(age>10)
                throw new MyException("please enter into comp if you are below 10");

          }
          catch(Exception e)
          {
                System.out.println("Please enter if you are below 10 only");
          }



                System.out.println("Hello");

  }
}
```

output



Consider the above example that whenever the variable value is greater than 10 then our own exception will be raised and the message that we defined in constructor or toString method will be displayed.

COMMAND LINE ARGUMENTS IN JAVA

Sending arguments to java program through command prompt is said to be command-line arguments.

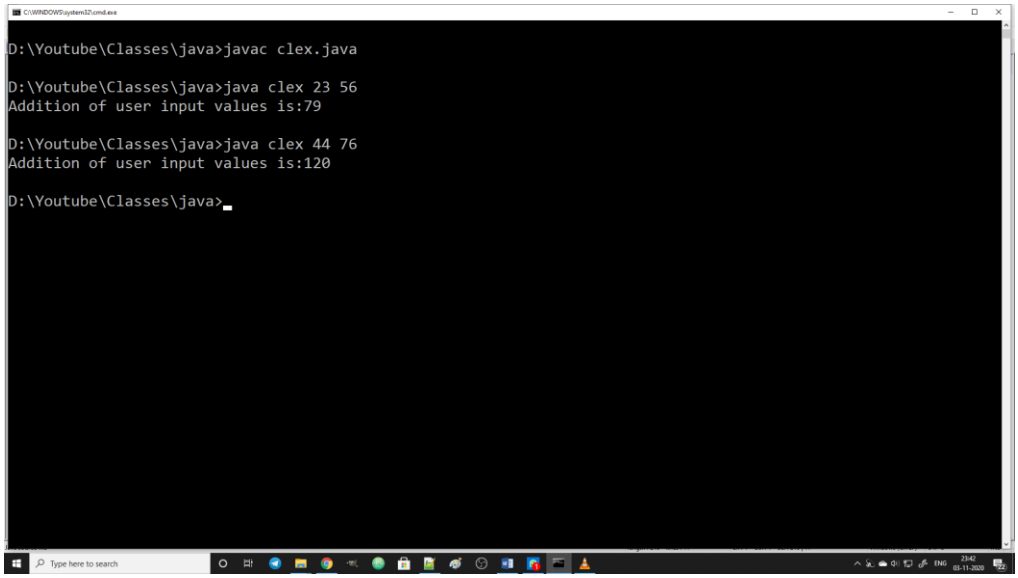
While executing java program we send arguments along with the java program. These arguments will be saved in string array which is available in main method.

We can retrieve these arguments and use in our program. please note that all arguments are saved as strings in array. so, if we want to perform some arithmetic operations then we must type cast them to the required data type and then do our calculation part

Example

```
class clex
{
	public static void main(String ar[])
	{
		int a = Integer.parseInt(ar[0]);
		int b = Integer.parseInt(ar[1]);
		System.out.println("Addition of user input values is:"+(a+b));

	}
}
```

output



In above example while executing program we sent 23,56 as arguments. These two will be saved in string array ar. These are converted into integers and then addition is performed. The same program is executed once again with different arguments 44,76 and we got the addition result as 120.

This is how we can send arguments through command prompt and use those input values in our program dynamically.

FILES IN JAVA

We can use files concept to check whether file exists or not, retrieve files metadata, we can perform different operations on files like read, write etc.,

All the classes and interfaces that are related to implement files will be in the package java.io. So, we must import this package.

How to get details about a file (existed or not, is it readable, when it is last modified etc.,)

We can check about existence of a file using java program as follows

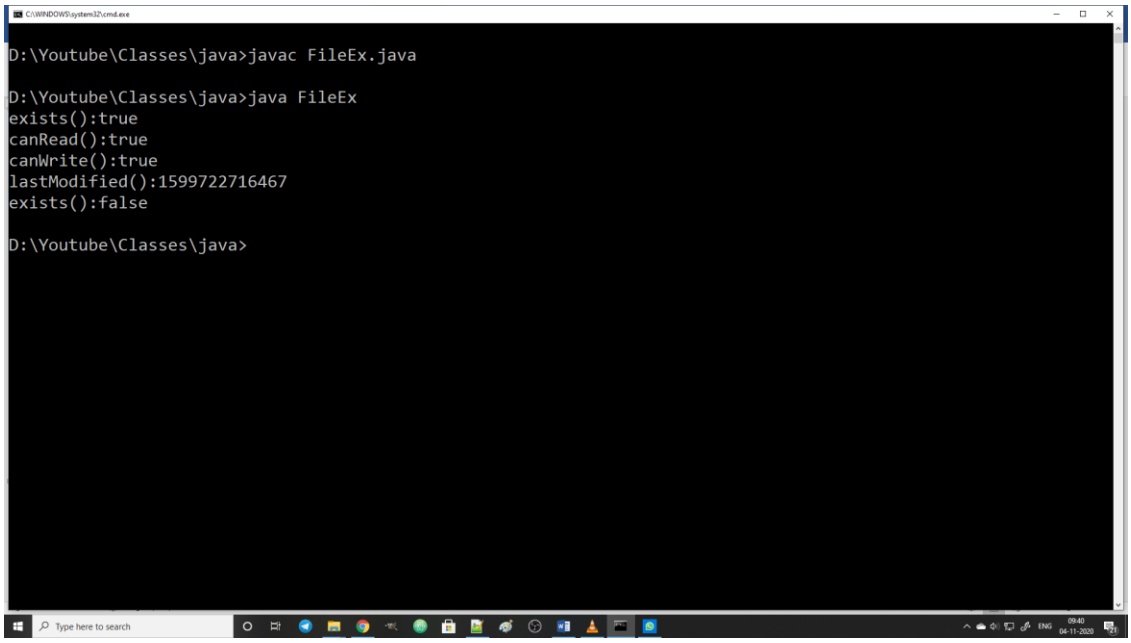
1) First, we need to create the File object with required file (for ex: sant.txt)
   File f = new File("sant.txt");
2) Now use the predefined methods and retrieve the information required as shown in example below

Example

```
//program to demonstrate Files
import java.io.*;

class FileEx
{
  public static void main(String ar[])
  {
          File f = new File("sample.txt");

          //methods in File class
          System.out.println("exists():"+f.exists());

          if(f.exists()==true)
          {
                  System.out.println("canRead():"+f.canRead());
                  System.out.println("canWrite():"+f.canWrite());
                  System.out.println("lastModified():"+f.lastModified());
                  f.delete();
                  System.out.println("exists():"+f.exists());
          }


  }
}
```

Observe the methods of File class in above example
exists : will check whether a file exists or not
canRead : will check whether file allows reading content from it
canWrite:will check whether file allows writing content into it
lastModified : will retrieve when file is modified recently.

Output:



In above example check the metadata that we retrieved.

exisits() : will returns true if file that we are searching for exists, else it returns false
canRead() : will returns true if file is readable
canWrite() : will returns true if file is writeable
lastModified() : will returns the last modified date and time in milliseconds

FILE READING, WRITING & COPYING :

To transfer data from one place to another place we need to use Streams in Java.

Stream:

A stream carries data just as a water pipe carries water from one place to another place. Streams can be categorized as 'input streams' and 'output streams'. Input streams are the streams which receive or read data while output streams are the streams which send or write data. All streams are represented by classes in java.io package.

We have two types of streams

1) Byte Streams : Transfer byte by byte
2) Character Streams : Transfer character by character

Generally, while working with image files, audio and video files we will use byte streams and while working with text files we will use character streams.

The classes and interfaces that are suffixed with InputStream & OutputStream are byte streams.
The classes and interfaces that are suffixed with Reader & Writer are character streams.

Reading contents from file will be accomplished in two steps

1) Open the file in read mode
2) read the contents from the file

First we will learn to read the contents from the file using character streams.

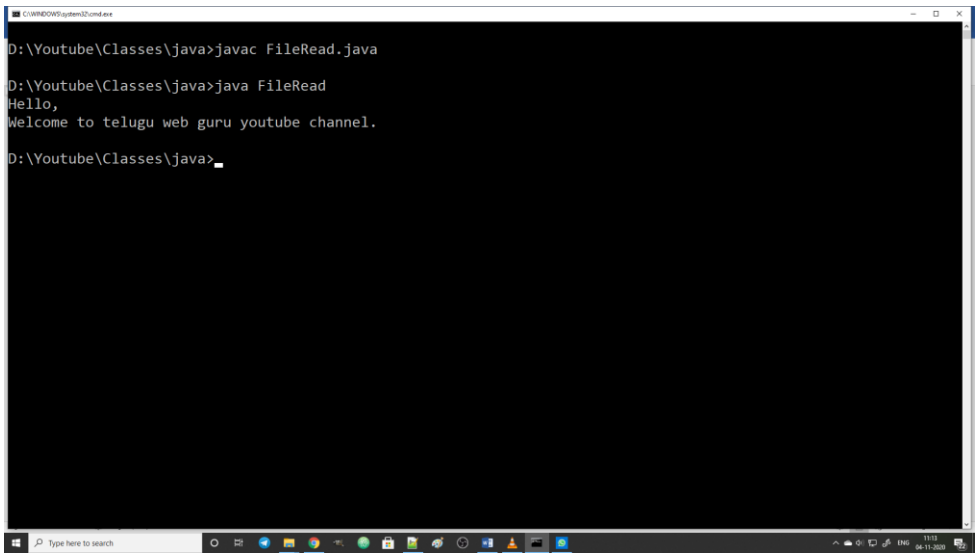
We used FileReader class to open the class.To read content more effectively we used BufferedReader

Now by using read / readLine methods from BufferedReader.The following example shows how to read contents from file

Example

```
//program to demonstrate reading contents from file
import java.io.*;
class FileRead
{
 public static void main(String ar[]) throws Exception
 {
          //open file in read mode
           FileReader fr = new FileReader("twg.txt");

          // int ch;

           //read the contents and print on the console.
          /* while((ch=fr.read())!=-1)
           {
                    System.out.print((char)ch);
           }*/

           BufferedReader br = new BufferedReader(fr);
           String s;

           while((s=br.readLine())!=null)
                    System.out.println(s);

            fr.close();

 }
}
```

Output



Writing contents into file

Now let us write the contents into the file

Writing also can be done in two steps

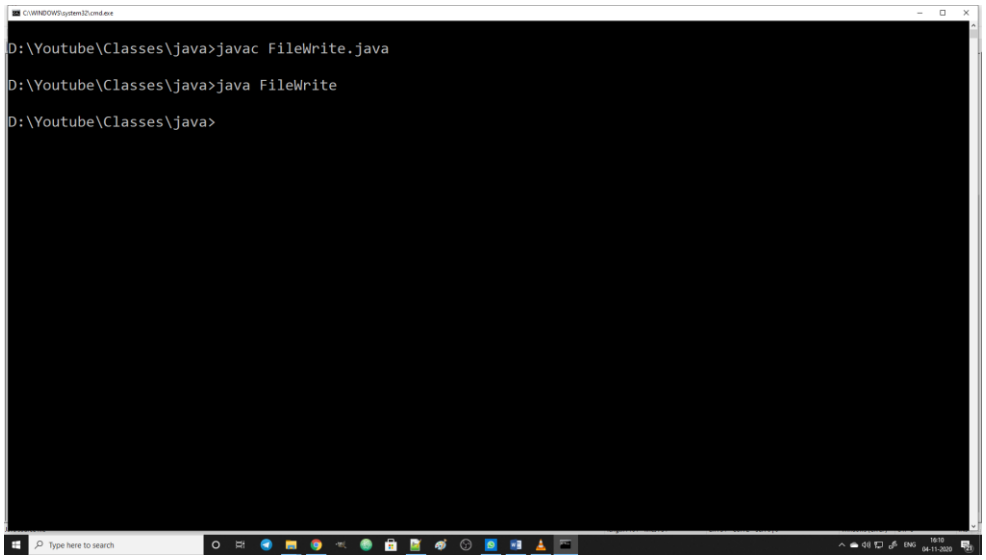
1) Open the file in write mode
2) Write the contents into the file.

Example

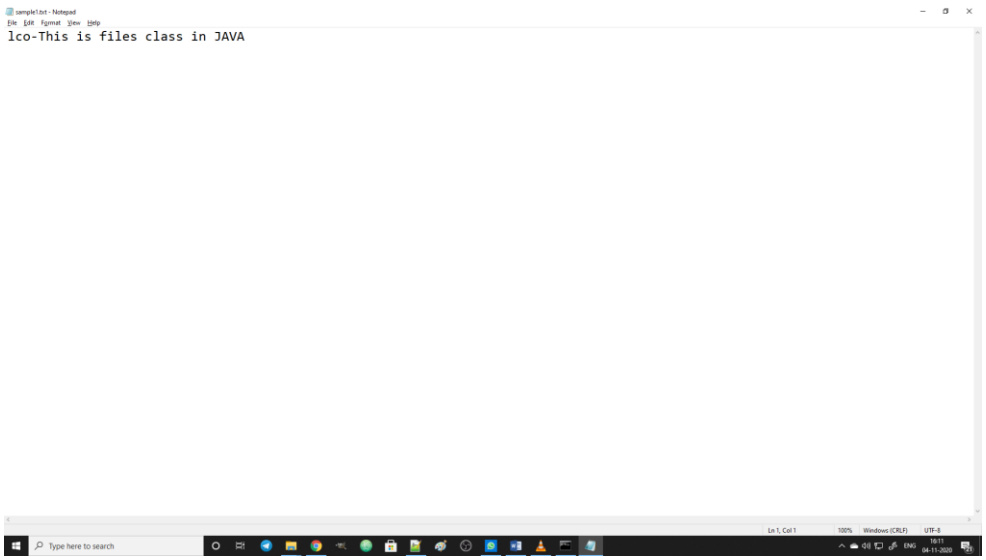
```
//program to demonstrate file writing
import java.io.*;

class FileWrite
{
        public static void main(String ar[]) throws Exception
        {
         // open the file in write mode

         FileWriter fw = new FileWriter("sample1.txt",true);

         //connect FileWriter fw with BufferedWriter

         BufferedWriter bw = new BufferedWriter(fw);

         //write the contents into the file

         char c[] = {'w','e','l','c','o','m','e'};

         bw.write(c, 2, 3); // it writes the 3 characters from char array c starts from index 2

         String s = "-This is files class in JAVA";

          bw.write(s, 0, s.length()); // write this string also into file from 0 index to full string
```

```
            //close the file
            bw.close();
            fw.close();

        }
}
```

output



Now let us check the sample1.txt file



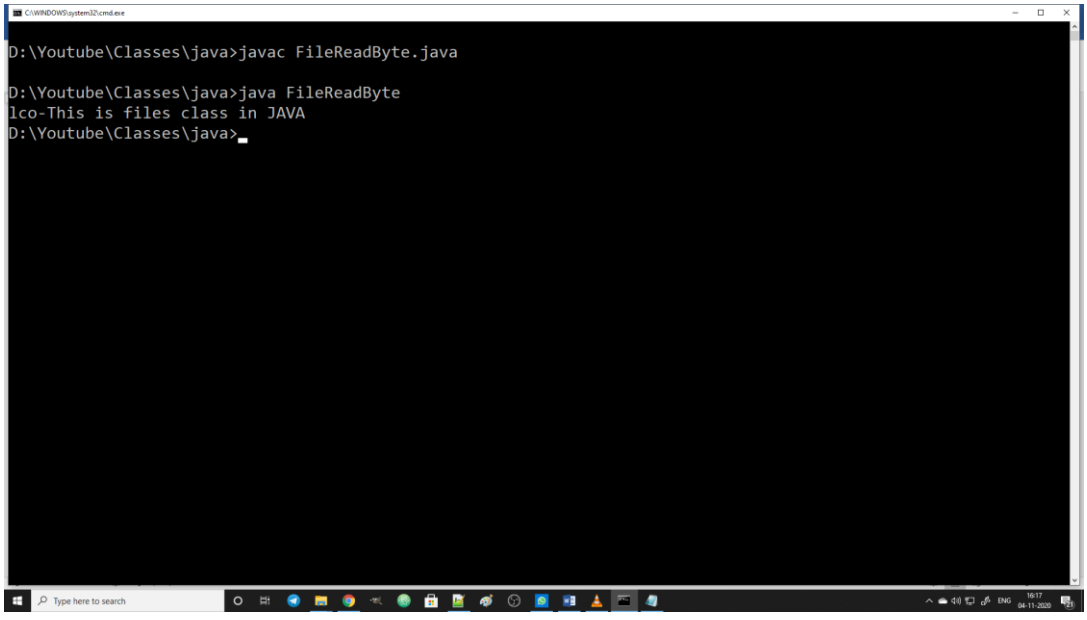
lco is came from char array c
-This is files class in JAVA is came from string

File Reading & Writing using Byte Streams:

Reading File content using Byte Streams :

```
import java.io.*;

class FileReadByte
{
 public static void main(String ar[]) throws Exception
 {
   //open file by using byte stream - FileInputStream
        FileInputStream fin = new FileInputStream("sample.txt");



        int ch;
        //read and print the contents
        while((ch=fin.read())!=-1)
          System.out.print((char)ch);


        fin.close();

 }
}
```

Output

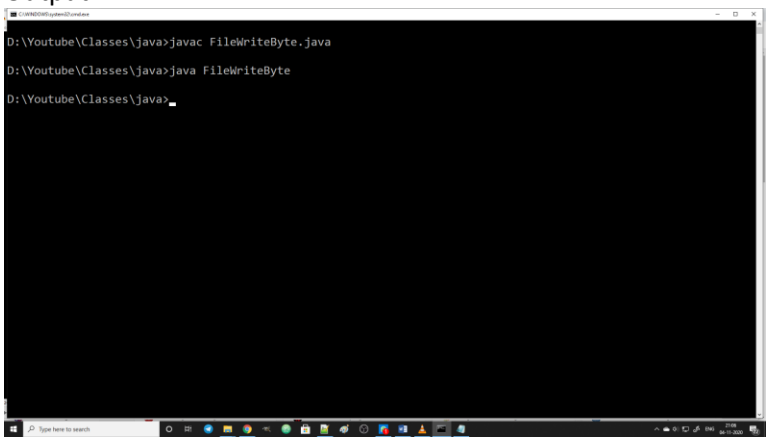


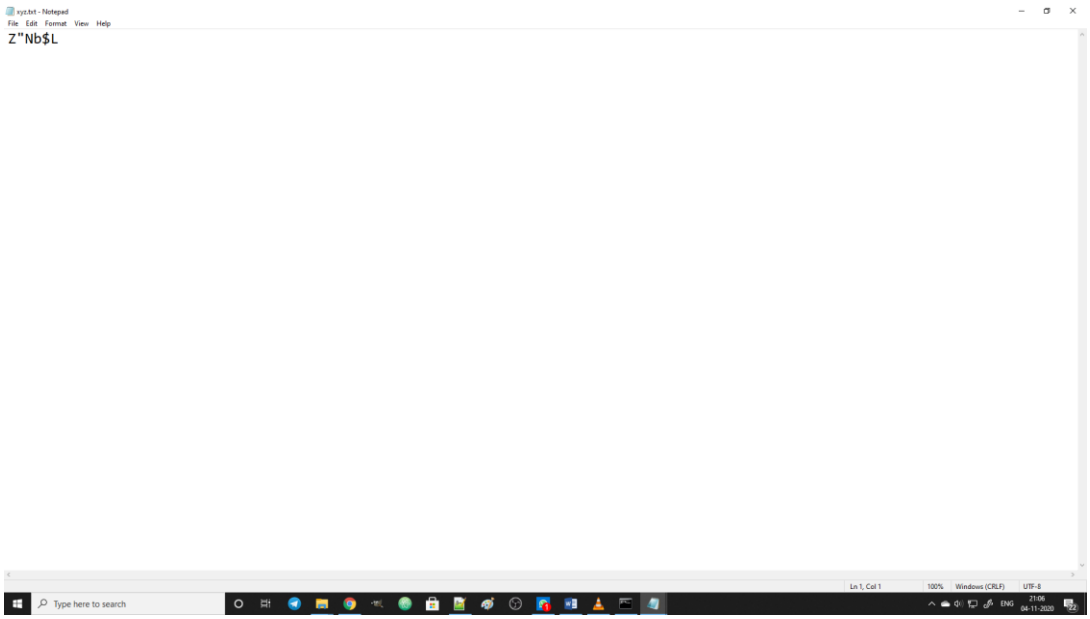
Here we need to read byte by byte which returns an integer value which in turn type casted into character type.

Writing File content using Byte Streams :

```
import java.io.*;

class FileWriteByte
{
	public static void main(String ar[]) throws Exception
	{
		//open the file in write mode
		FileOutputStream fos=new FileOutputStream("xyz.txt");

		//write the contents into the file
		fos.write((int)'Z');
		byte b[]={34,78,98,36,76};
		fos.write(b);
		fos.close();
	}

}
```

Output





In above program Initially xyz.txt file is opened in write mode and then contents was written into the file. Here the equivalent characters of each and every value were written into the file.

NOTE :

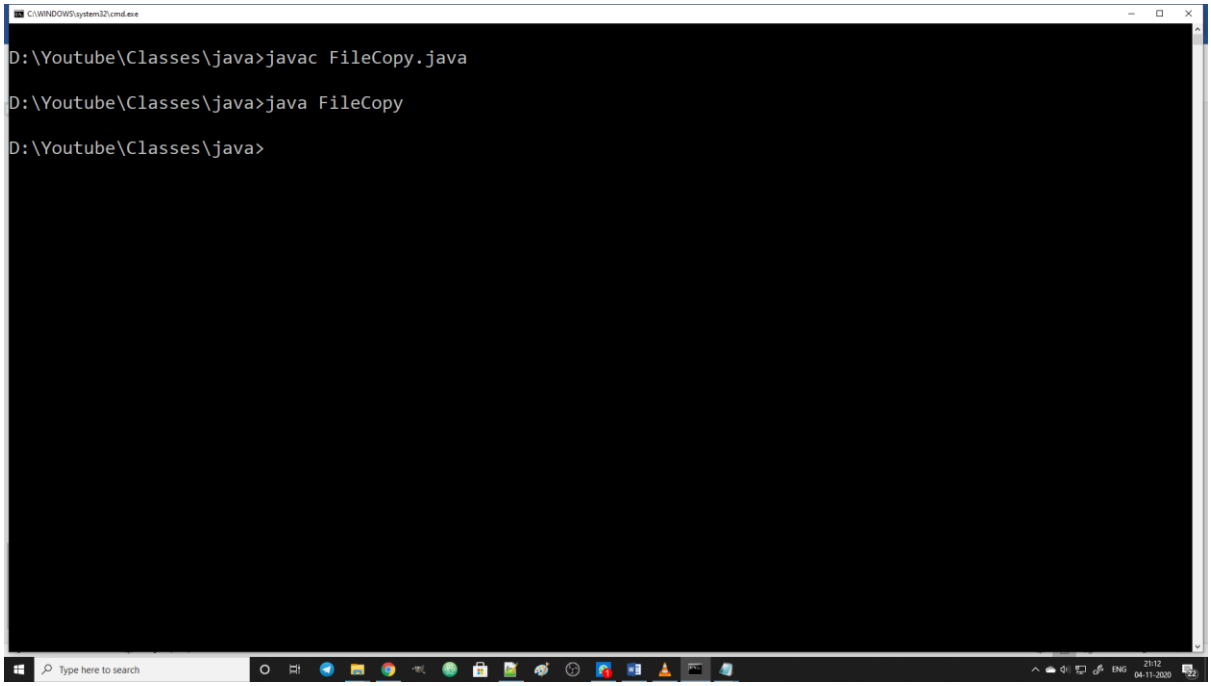
To Indicate end of the file byte streams uses -1 where as Character streams uses 'null'.

Following program is another example of files which copy's the contents of a file into another file

Example

```
//program to demonstrate copy one file contents into another file
import java.io.*;

class FileCopy
{
	public static void main(String ar[]) throws Exception
	{

	    //open logo.png image file in read mode
	    FileInputStream fin=new FileInputStream("small.png");


	   //open new file in write mode
	   FileOutputStream fout = new FileOutputStream("twglogo.png");


	   //perform the copy operation
	   int ch;

	   while((ch=fin.read())!=-1)
	         fout.write(ch);


	   //close the streams
	   fin.close();
	   fout.close();

	}
}
```

output



The following is the existing file small.png

and this file contents are copied into another file twglogo.png as follows

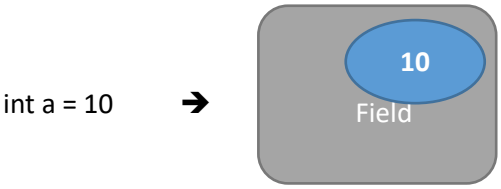
NOTE : For Image files, audio and video files we must use byte streams only.

WRAPPER CLASSES IN JAVA

Collections in java always excepts objects only. So, we may not use primitive data types like int, float, double etc.,

So, to convert primitive data types into objects we will use wrapper classes.

Example



primitive data type object

The following are available Wrapper classes

1) Number

   It is super class of all wrapper classes related to numbers. Integer, Byte, Short, Long, Float, Double are wrapper classes related to numbers.

   for example to convert int a =10 into wrapper class we will use Integer Wrapper class as shown below

   Integer I = new Integer(a);

2) Character

   to convert char type into wrapper class we will use Character Wrapper class

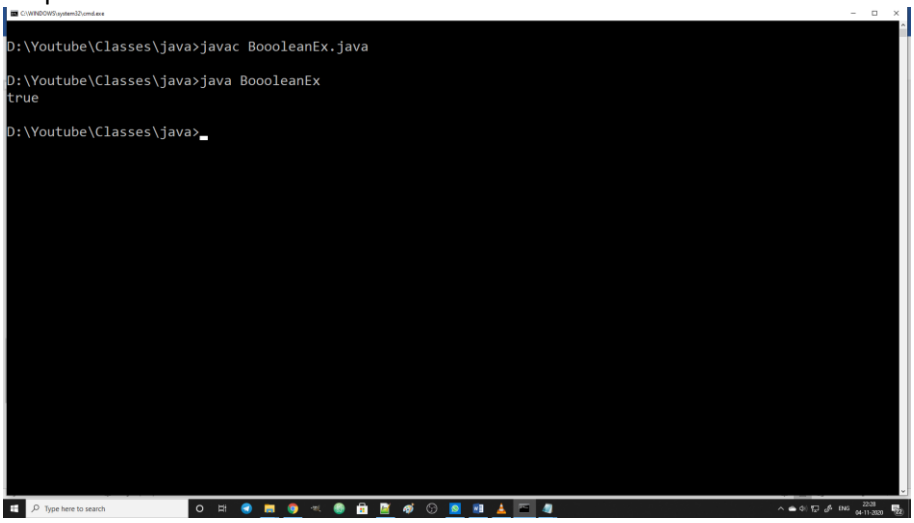
3) Boolean

   to convert boolean type value into wrapper class we will use Boolean wrapper class

   Example

```
//Wrapper classes - Boolean
class BoooleanEx
{
  public static void main(String ar[])
  {
     boolean a=true;

          Boolean b = new Boolean(a);
          System.out.println(b.booleanValue());

  }
}
```

output



To retrieve boolean value again from wrapper class object we will use booleanValue method

Example 2:

```
//Wrapper classes - Byte
class ByteEx
{
 public static void main(String ar[])
 {
    byte a = 34;

          Byte b = new Byte(a);

          System.out.println(b.byteValue());
          System.out.println(b.intValue());
         System.out.println(b.floatValue());
         System.out.println(b.doubleValue());
 }
}
```

Now observe the above program .

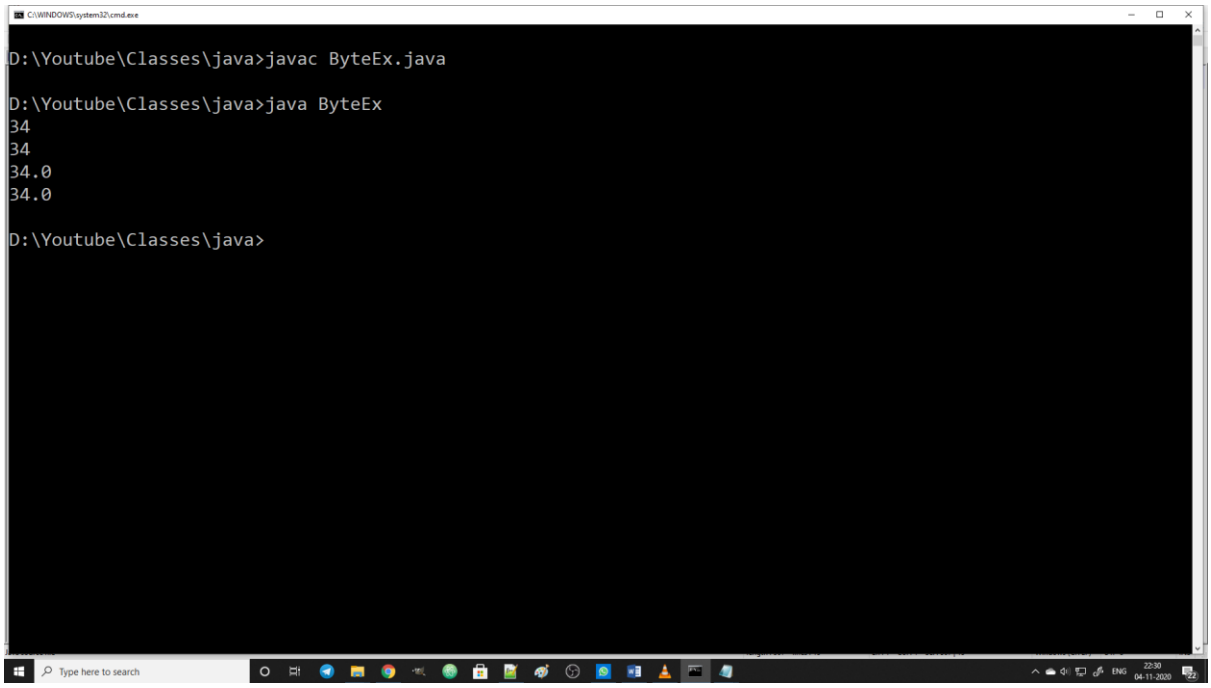
Even though b is byte type, we can retrieve the value from this variable as per our required type.

for example

if you retrieve the value using byteValue() method then it returns the value as byte,

if you use floatValue() method it returns the value as float.

output


C:\WINDOWS\system32\cmd.exe
D:\Youtube\Classes\java>javac ByteEx.java
D:\Youtube\Classes\java>java ByteEx
34
34
34.0
34.0
D:\Youtube\Classes\java>
Type here to search

THE COLLECTION FRAMEWORK IN JAVA

A collection represents a group of objects, known as its elements. Some collections allow duplicate elements and others do not. Some are ordered and others unordered. The JDK does not provide any direct implementations of this interface: it provides implementations of more specific sub interfaces like Set and List. This interface is typically used to pass collections around and manipulate them where maximum generality is desired.

All the collection related classes and interfaces are available in java.util package.

The following are the list of interfaces available and corresponding implementation classes.

Interface Type	Implementation Classes
Set<T>	HashSet<T> LinkedHashSet<T>
List<T>	Stack<T> LinkedList<T> ArrayList<T> Vector<T>
Queue<T>	LinkedList<T>
Map<K,V>	HashMap<K,V> Hashtable<K,V>

The following are the main interfaces that are available in collections framework.

1) Set: is a collection that contains no duplicate elements.
2) List: An ordered collection (also known as a sequence). It allows duplicate elements.
3) Queue: A collection designed for holding elements prior to processing. Besides basic Collection operations, queues provide additional insertion, extraction, and inspection operations.
4) Map: An object that maps keys to values. A map cannot contain duplicate keys; each key can map to at most one value.

Now Let us see each implementation class under interfaces with examples

SETS

HashSet:

This class implements the Set interface which makes no guarantees as to the iteration order of the set; in particular, it does not guarantee that the order will remain constant over time. This class permits the null element.

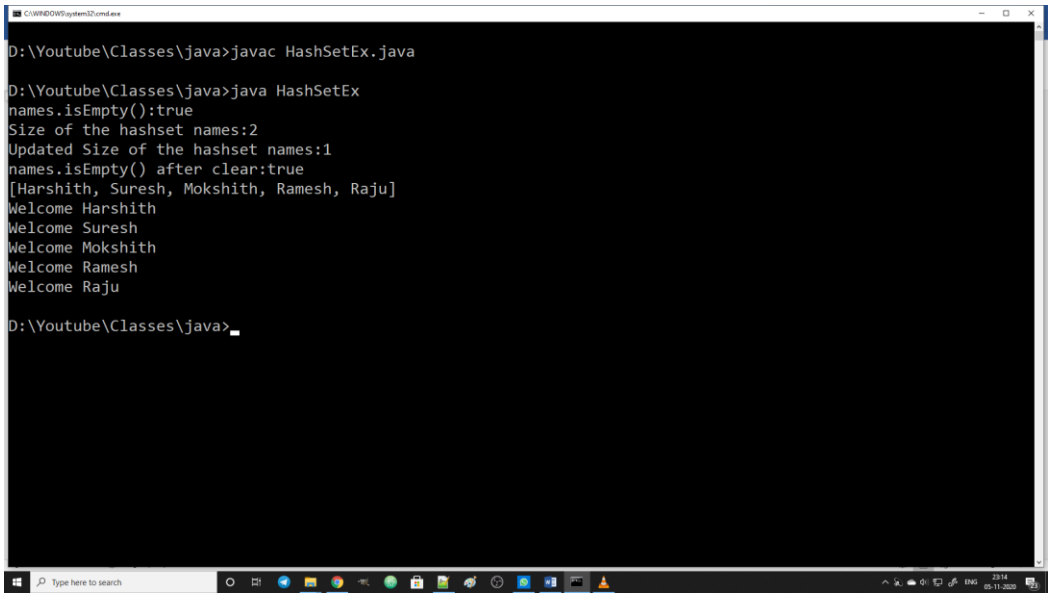
Methods in HashSet:
add(E e): Adds the specified element to this set if it is not already present.
clear():Removes all of the elements from this set.
contains(Object o): Returns true if this set contains the specified element.
isEmpty():Returns true if this set contains no elements.
iterator():Returns an iterator over the elements in this set.
remove(Object o) : Removes the specified element from this set if it is present.
size():Returns the number of elements in this set (its cardinality).

Example:

```
//program to demonstrate HashSet in java
import java.util.*;

class HashSetEx
{
  public static void main(String ar[])
  {
            HashSet<String> names = new HashSet<String>();

            System.out.println("names.isEmpty():"+names.isEmpty());

            //adding elements/objects
            names.add("Santosh");
            names.add("Telugu Web Guru");

            if(names.isEmpty())
                        System.out.println("names hashset is still empty");
            else
                        System.out.println("Size of the hashset names:"+names.size());

            if(names.contains("Telugu Web Guru"))
                        names.remove("Telugu Web Guru");

             System.out.println("Updated Size of the hashset names:"+names.size());

             names.clear();

     System.out.println("names.isEmpty() after clear:"+names.isEmpty());
```

```
        names.add("Raju");
        names.add("Harshith");
        names.add("Mokshith");
        names.add("Suresh");
        names.add("Ramesh");

        System.out.println(names);

       /* for(String n : names)
        {
                String newvalue = "Welcome "+n;
                System.out.println(newvalue);
        }*/

        Iterator<String> newNames = names.iterator();
        while(newNames.hasNext())
        {
                String n = newNames.next();
                 String newvalue = "Welcome "+n;
                System.out.println(newvalue);
        }



 }

}
```

output



LinkedHashSet:

Hash table and linked list implementation of the Set interface, with predictable iteration order. This implementation differs from HashSet in that it maintains a doubly-linked list running through all of its entries. This linked list defines the iteration ordering, which is the order in which elements were inserted into the set (insertion-order). Note that insertion order is not affected if an element is re-inserted into the set. (An element e is reinserted into a set s if s.add(e) is invoked when s.contains(e) would return true immediately prior to the invocation.)

Stack:

The Stack class represents a last-in-first-out (LIFO) stack of objects. It extends class Vector with five operations that allow a vector to be treated as a stack. The usual push and pop operations are provided, as well as a method to peek at the top item on the stack, a method to test for whether the stack is empty, and a method to search the stack for an item and discover how far it is from the top.

When a stack is first created, it contains no items.

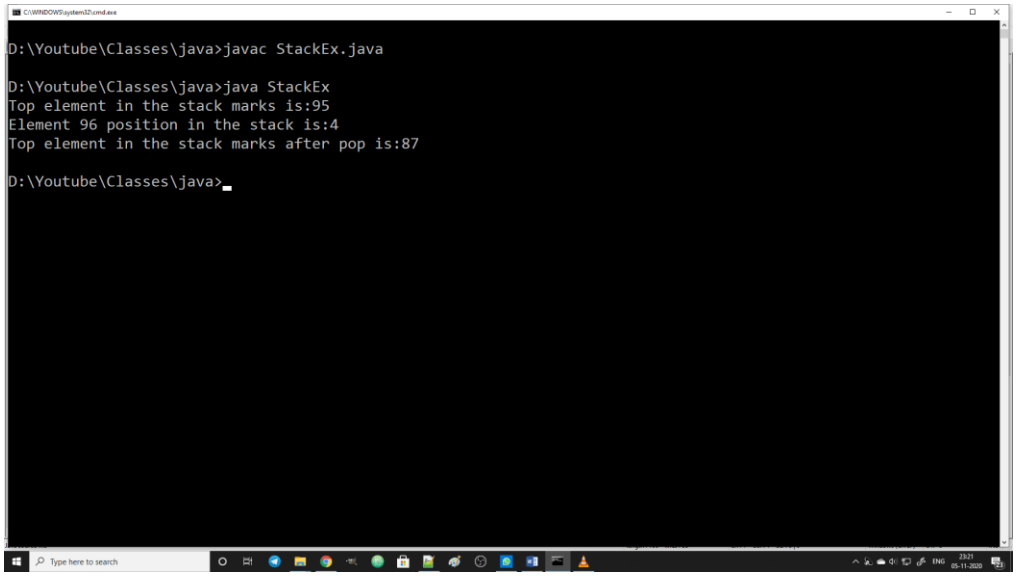
Methods in Stack:
empty() : Tests if this stack is empty.
peek() : Looks at the object at the top of this stack without removing it from the stack.
pop() : Removes the object at the top of this stack and returns that object as the value of this function.
push(E item): Pushes an item onto the top of this stack.
search(Object o): Returns the 1-based position where an object is on this stack.

Example

```
//program to demonstrate Stack (LIFO) in Java
import java.util.*;
class StackEx
{
	public static void main(String ar[])
	{
		Stack<Integer> marks = new Stack<Integer>();

		//push an element
		marks.push(99);
		marks.push(98);
		marks.push(96);
		marks.push(87);
		marks.push(87);
		marks.push(95);

		if(marks.empty())//it checks whether stack is empty or not
			System.out.println("Stack is empty");
		else
```

```
                    System.out.println("Top element in the stack marks
is:"+marks.peek());//returns top element


                System.out.println("Element 96 position in the stack is:"+marks.search(96));


                //it removes the top element from the stack
                marks.pop();

                System.out.println("Top element in the stack marks after pop is:"+marks.peek());




        }
}
```

output



LinkedList:

Doubly-linked list implementation of the List and Deque interfaces. Implements all optional list operations, and permits all elements (including null).
All of the operations perform as could be expected for a doubly-linked list. Operations that index into the list will traverse the list from the beginning or the end, whichever is closer to the specified index.

Note that this implementation is not synchronized. If multiple threads access a linked list concurrently, and at least one of the threads modifies the list structurally, it must be synchronized externally.

Methods in LinkedList :

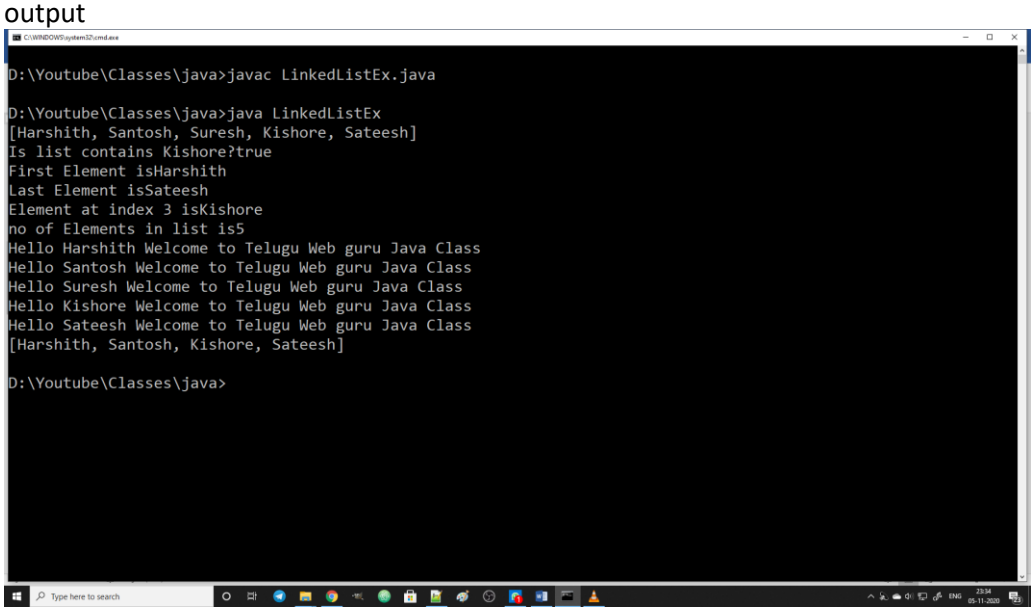
add(E e): Appends the specified element to the end of this list.
add(int index, E element): Inserts the specified element at the specified position in this list.
addFirst(E e): Inserts the specified element at the beginning of this list.
addLast(E e): Appends the specified element to the end of this list.
clear():Removes all of the elements from this list.
clone():Returns a shallow copy of this LinkedList.
contains(Object o): Returns true if this list contains the specified element.
get(int index): Returns the element at the specified position in this list.
getFirst():Returns the first element in this list.
getLast():Returns the last element in this list.
indexOf(Object o): Returns the index of the first occurrence of the specified element in this list, or -1 if this list does not contain the element.
pop():Pops an element from the stack represented by this list.
push(E e) : Pushes an element onto the stack represented by this list.

Example

```
//program to demonstrate linked list
import java.util.*;
class LinkedListEx
{
	public static void main(String ar[])
	{
		LinkedList<String> list = new LinkedList<String>();

		list.add("Santosh");
		list.add("Kishore");

		list.add(1,"Suresh");

		list.addLast("Sateesh");
		list.addFirst("Harshith");

		System.out.println(list);

		System.out.println("Is list contains Kishore?"+list.contains("Kishore"));

		System.out.println("First Element is"+list.getFirst());
		System.out.println("Last Element is"+list.getLast());
		System.out.println("Element at index 3 is"+list.get(3));


		System.out.println("no of Elements in list is"+list.size());

		for(String s:list)
			System.out.println("Hello "+s+" Welcome to Telugu Web guru Java Class");

		list.remove(2);
```

```
                System.out.println(list);

        }
}
```

output



HashMap:

Hash table based implementation of the Map interface. This implementation provides all of the optional map operations, and permits null values and the null key. This class makes no guarantees as to the order of the map; in particular, it does not guarantee that the order will remain constant over time.

Methods in HashMap:

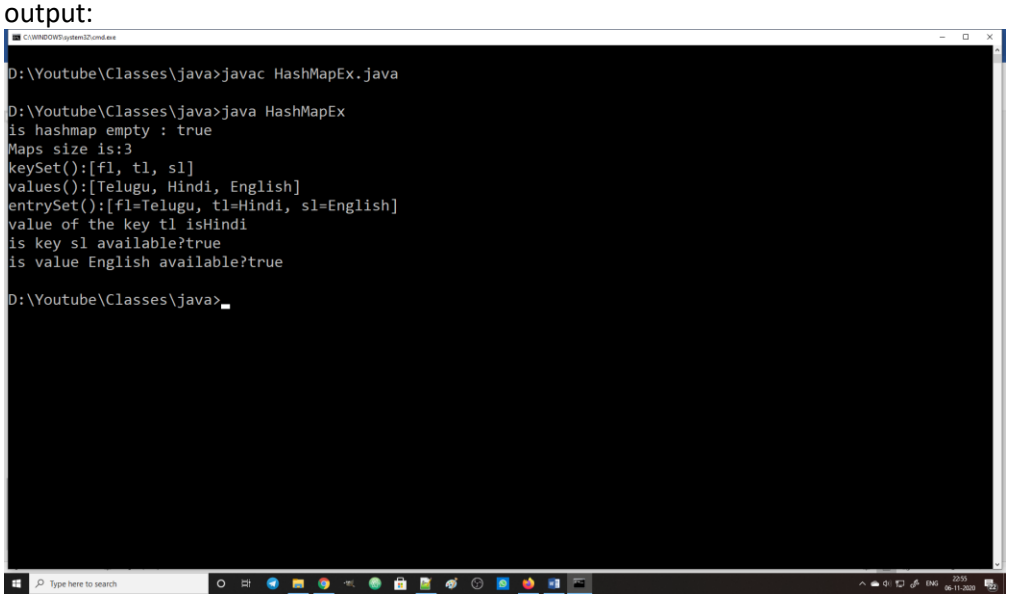
clear(): Removes all of the mappings from this map.
clone():Returns a shallow copy of this HashMap instance: the keys and values themselves are not cloned.
containsKey(Object key): Returns true if this map contains a mapping for the specified key.
containsValue(Object value): Returns true if this map maps one or more keys to the specified value.
isEmpty():Returns true if this map contains no key-value mappings.
keySet():Returns a Set view of the keys contained in this map.
size():Returns the number of key-value mappings in this map.
values():Returns a Collection view of the values contained in this map.

Example:

```
//program to demonstrate HashMap in java
import java.util.*;

class HashMapEx
{
```

```
	public static void main(String ar[])
	{
		HashMap<String,String> maps = new HashMap<String,String>();

		System.out.println("is hashmap empty : "+maps.isEmpty());

		maps.put("fl","Telugu");
		maps.put("sl","English");
		maps.put("tl","Hindi");

		if(maps.isEmpty())
			System.out.println("Map is still empty");
		else
			System.out.println("Maps size is:"+maps.size());

		System.out.println("keySet():"+maps.keySet());

		System.out.println("values():"+maps.values());

		System.out.println("entrySet():"+maps.entrySet());

		System.out.println("value of the key tl is"+maps.get("tl"));

		System.out.println("is key sl available?"+maps.containsKey("sl"));

		System.out.println("is value English available?"+maps.containsValue("English"));



	}
}
```

output:



HashTable:

This class implements a hash table, which maps keys to values. Any non-null object can be used as a key or as a value.

To successfully store and retrieve objects from a hashtable, the objects used as keys must implement the hashCode method and the equals method.

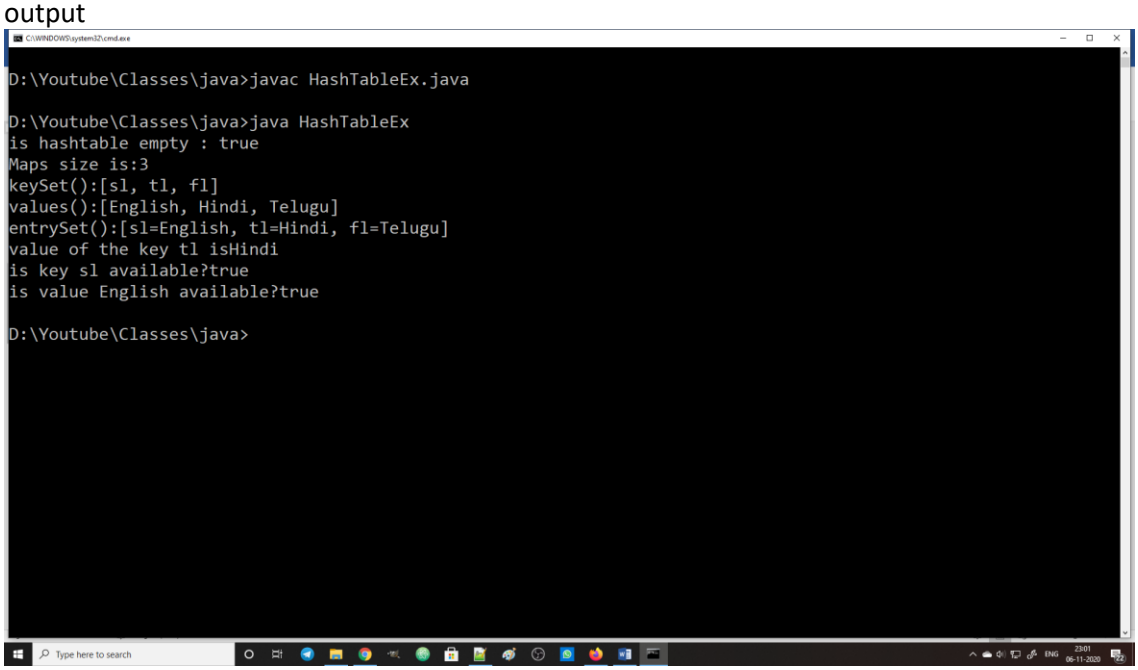
Methods in HashTable:
clear(): Clears this hashtable so that it contains no keys.
clone():Creates a shallow copy of this hashtable.
containsKey(Object key): Tests if the specified object is a key in this hashtable.
containsValue(Object value): Returns true if this hashtable maps one or more keys to this value.
isEmpty():Tests if this hashtable maps no keys to values.
keySet():Returns a Set view of the keys contained in this map.
size():Returns the number of key-value mappings in this table.
values():Returns a Collection view of the values contained in this map.

Example

```
//program to demonstrate HashTable in java
import java.util.*;

class HashTableEx
{
	public static void main(String ar[])
	{
		Hashtable<String,String> maps = new Hashtable<String,String>();

		System.out.println("is hashtable empty : "+maps.isEmpty());

		maps.put("fl","Telugu");
		maps.put("sl","English");
		maps.put("tl","Hindi");

		if(maps.isEmpty())
			System.out.println("Map is still empty");
		else
			System.out.println("Maps size is:"+maps.size());

		System.out.println("keySet():"+maps.keySet());

		System.out.println("values():"+maps.values());

		System.out.println("entrySet():"+maps.entrySet());

		System.out.println("value of the key tl is"+maps.get("tl"));

		System.out.println("is key sl available?"+maps.containsKey("sl"));

		System.out.println("is value English available?"+maps.containsValue("English"));
```

```
        }
}
```

output


D:\Youtube\Classes\java>javac HashTableEx.java
D:\Youtube\Classes\java>java HashTableEx
is hashtable empty : true
Maps size is:3
keySet():[sl, tl, fl]
values():[English, Hindi, Telugu]
entrySet():[sl=English, tl=Hindi, fl=Telugu]
value of the key tl isHindi
is key sl available?true
is value English available?true
D:\Youtube\Classes\java>

IMPORTANT CLASSES IN java.util PACKAGE

Arrays :

This class contains various methods for manipulating arrays (such as sorting and searching).

Methods in Arrays :

binarySearch(byte[] a, byte key) : Searches the specified array of bytes for the specified value using the binary search algorithm. The array must be sorted (as by the sort(byte[]) method) prior to making this call. If it is not sorted, the results are undefined.

equals(byte[] a, byte[] a2) : Returns true if the two specified arrays of bytes are equal to one another.

fill(boolean[] a, boolean val) : Assigns the specified boolean value to each element of the specified array of booleans.

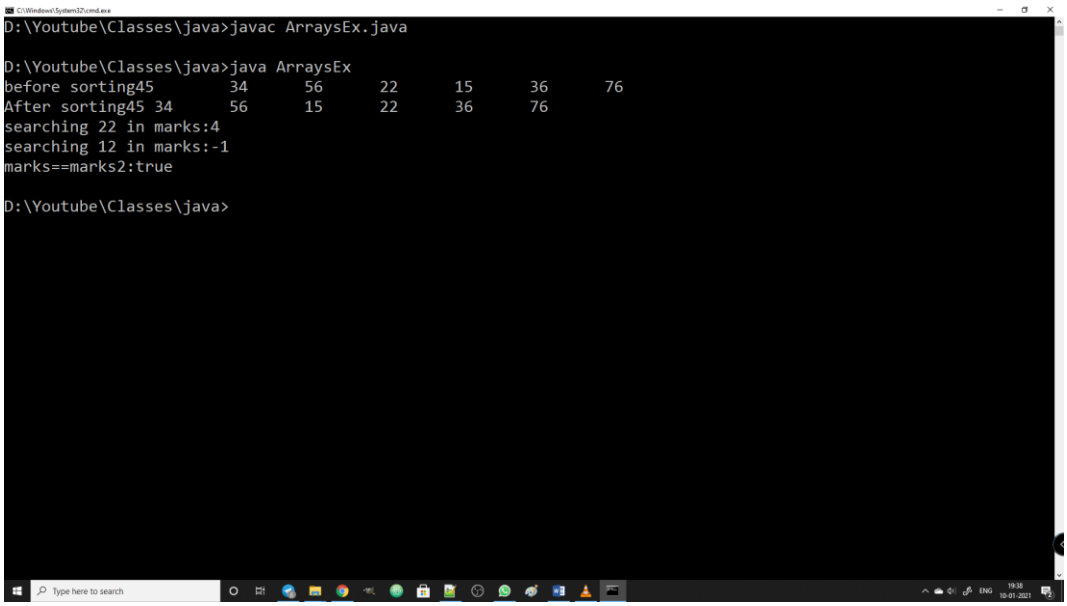
sort(byte[] a) : Sorts the specified array into ascending numerical order.

toString(byte[] a) : Returns a string representation of the contents of the specified array.

Example

```
//program to demonstrate Arrays class in java.util
import java.util.*;
class ArraysEx
{
	public static void main(String ar[])
	{
	int marks[] = {45,34,56,22,15,36,76};
	int marks2[] = {45,34,56,22,15,36,76};
	int marks3[] = {45,34,56,22,15,36,76};

	System.out.print("before sorting");

	for(int i=0;i<marks.length;i++)
		System.out.print(marks[i]+"\t");

	Arrays.sort(marks,3,7);

	System.out.print("\nAfter sorting");
	for(int i=0;i<marks.length;i++)
		System.out.print(marks[i]+"\t");

	System.out.println();

	System.out.println("searching 22 in marks:"+Arrays.binarySearch(marks, 22));

	System.out.println("searching 12 in marks:"+Arrays.binarySearch(marks, 12));

	System.out.println("marks==marks2:"+Arrays.equals(marks3,marks2));

	}

}
```

Output:



StringTokenizer class :

The string tokenizer class allows an application to break a string into tokens. The tokenization method is much simpler than the one used by the StreamTokenizer class. The set of delimiters (the characters that separate tokens) may be specified either at creation time or on a per-token basis. A StringTokenizer object internally maintains a current position within the string to be tokenized. Some operations advance this current position past the characters processed.

Method Summary:

countTokens() : Calculates the number of times that this tokenizer's nextToken method can be called before it generates an exception.

hasMoreElements() : Returns the same value as the hasMoreTokens method.

hasMoreTokens() : Tests if there are more tokens available from this tokenizer's string.

nextElement() : Returns the same value as the nextToken method, except that its declared return value is Object rather than String.

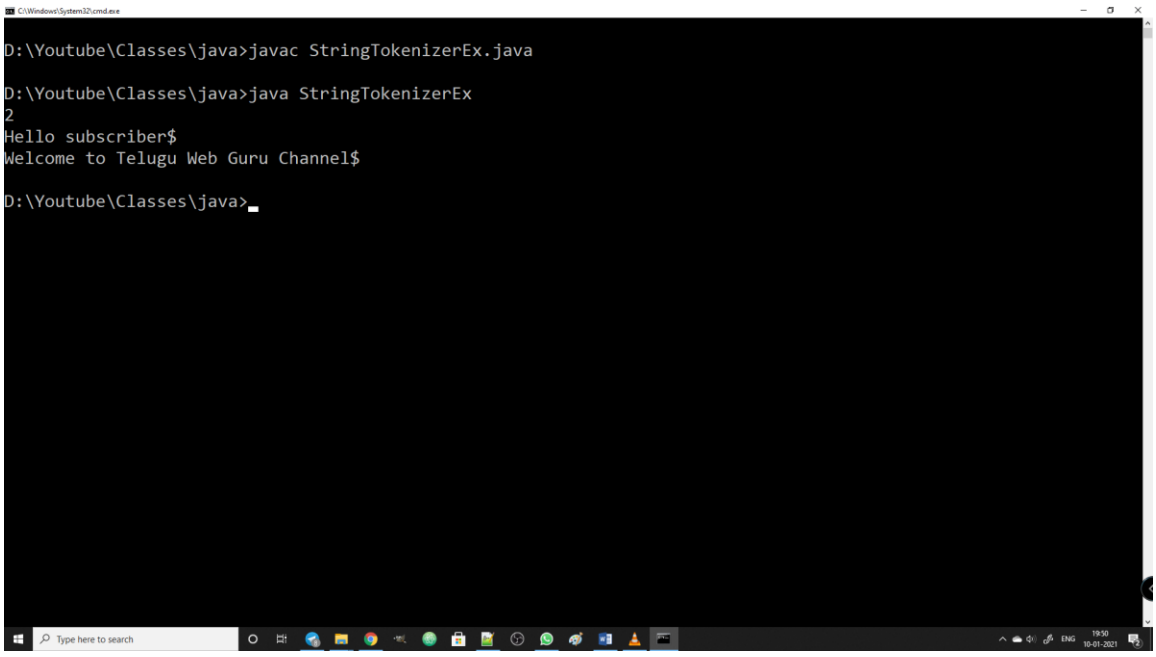
nextToken() : Returns the next token from this string tokenizer.

nextToken(String delim) : Returns the next token in this string tokenizer's string.

Example:

```
//program to demonstrate StringTokenizer class
import java.util.*;
class StringTokenizerEx
{
	public static void main(String ar[])
	{
		String s = "Hello subscriber,Welcome to Telugu Web Guru Channel";

		StringTokenizer st = new StringTokenizer(s,",");

      System.out.println(st.countTokens());

		while(st.hasMoreTokens())
			System.out.println(st.nextToken()+"$");



	}

}
```

Output:



Calendar class:

The Calendar class is an abstract class that provides methods for converting between a specific instant in time and a set of calendar fields such as YEAR, MONTH, DAY_OF_MONTH, HOUR, and so on, and for manipulating the calendar fields, such as getting the date of the next week. An instant in time can be represented by a millisecond value that is an offset from the Epoch, January 1, 1970 00:00:00.000 GMT (Gregorian).

Methods:

add(int field, int amount) : Adds or subtracts the specified amount of time to the given calendar field, based on the calendar's rules.

after(Object when) : Returns whether this Calendar represents a time after the time represented by the specified Object.

before(Object when) : Returns whether this Calendar represents a time before the time represented by the specified Object.

clear() : Sets all the calendar field values and the time value (millisecond offset from the Epoch) of this Calendar undefined.

compareTo(Calendar anotherCalendar) : Compares the time values (millisecond offsets from the Epoch) represented by two Calendar objects.

equals(Object obj) : Compares this Calendar to the specified Object.

getInstance() : Gets a calendar using the default time zone and locale.

getMaximum(int field) : Returns the maximum value for the given calendar field of this Calendar instance.

getMinimum(int field) : Returns the minimum value for the given calendar field of this Calendar instance.

getTime() : Returns a Date object representing this Calendar's time value

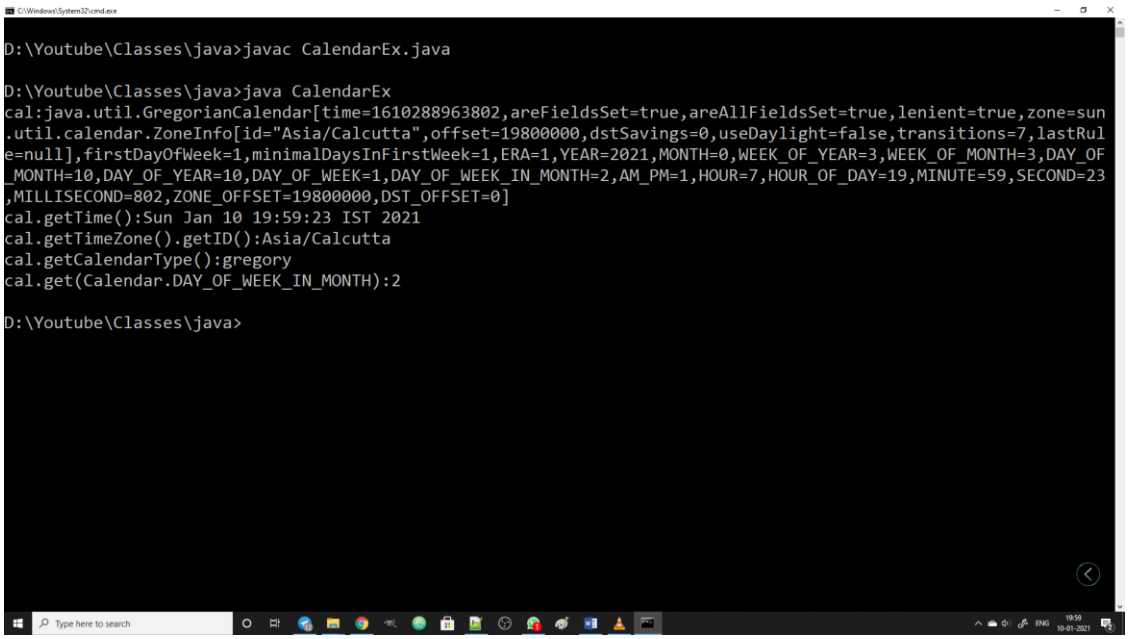
getTimeInMillis() : Returns this Calendar's time value in milliseconds.

getTimeZone() : Gets the time zone.

Example :

```
//program to demonstrate Calendar class (abstract)
import java.util.*;

class CalendarEx
{
  public static void main(String ar[])
  {
            Calendar cal = Calendar.getInstance();

            System.out.println("cal:"+cal);
            System.out.println("cal.getTime():"+cal.getTime());

            System.out.println("cal.getTimeZone().getID():"+cal.getTimeZone().getID());
```

```
            System.out.println("cal.getCalendarType():"+cal.getCalendarType());


System.out.println("cal.get(Calendar.DAY_OF_WEEK_IN_MONTH):"+cal.get(Calendar.DAY_OF_WEEK_IN_MONTH));
  }
}
```



Class Date

The class Date represents a specific instant in time, with millisecond precision.

Methods :

after(Object when) : Tests if this date is after the specified date.

before(Object when) : Tests if this date is before the specified date.

compareTo(Date anotherDate) : Compares two Dates for ordering.

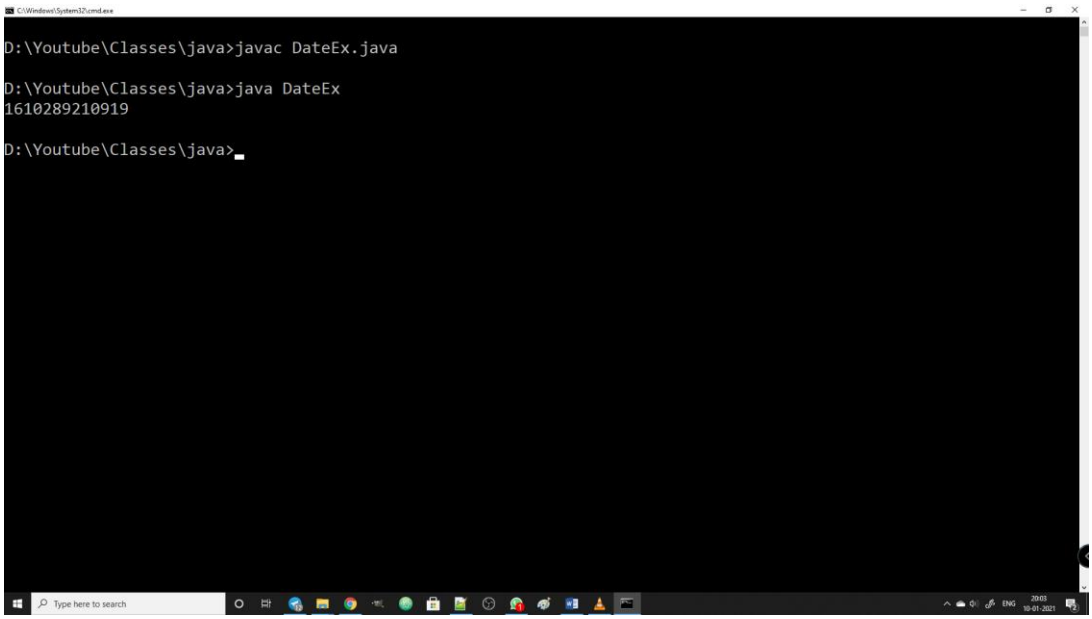
equals(Object obj) Compares two dates for equality.

Example

```
//program to demonstrate Date class in java
import java.util.*;
class DateEx
{
        public static void main(String ar[])
        {
                Date d = new Date();

                System.out.println(d.getTime());

        }
```

```
}
```

Output:


D:\Youtube\Classes\java>javac DateEx.java
D:\Youtube\Classes\java>java DateEx
1610289210919
D:\Youtube\Classes\java>
Type here to search

Multi-Threading in JAVA

Generally, operating system executes different processes sequentially or concurrently.

Concurrent or parallel execution of multiple tasks is said to be multi-tasking.

There are two types of multi-tasking

1) Process Based Multitasking
2) Thread Based Multitasking

Process Based Multi-Tasking:

Here each process will be considered as a separate program. so, Memory and other resources are separately allocated for each process. As the number of processes are increased, the more amount of resources required to implement process based multi-tasking. so, this is called as heavy weight processing.

Thread Based Multi-Tasking:

Here all the tasks are considered under one process where each task will be executed as a thread. In this case all the threads under a process will share the same resources that are allotted to the process. So, with less amount of resources only we can implement thread based multi-tasking. That's why this is called as Light Weight processing.

What is a Thread?

A Thread represents single flow of execution or separate path of execution.

How to create Threads?

We can create threads by extending Thread class or by implementing Runnable interface.

The following are the steps to be followed to create threads.

1) Create a class that extends Thread or implements Runnable

```
class MyThread extends Thread
{

}

        or
class MyThread implements Runnable
{

}
```

2) override run() method

```
public void run(){
  //main code of the thread
}
```

3) create an object to the above class

```
class MainClass {

    public static void main(String ar[])
        {
            MyThread ob=new MyThread();


        }

}
```

4) Create thread object and link it with our class object which is created in step-3

```
Thread t = new Thread(ob);
```

5) Run the thread using start()

```
t.start()
```

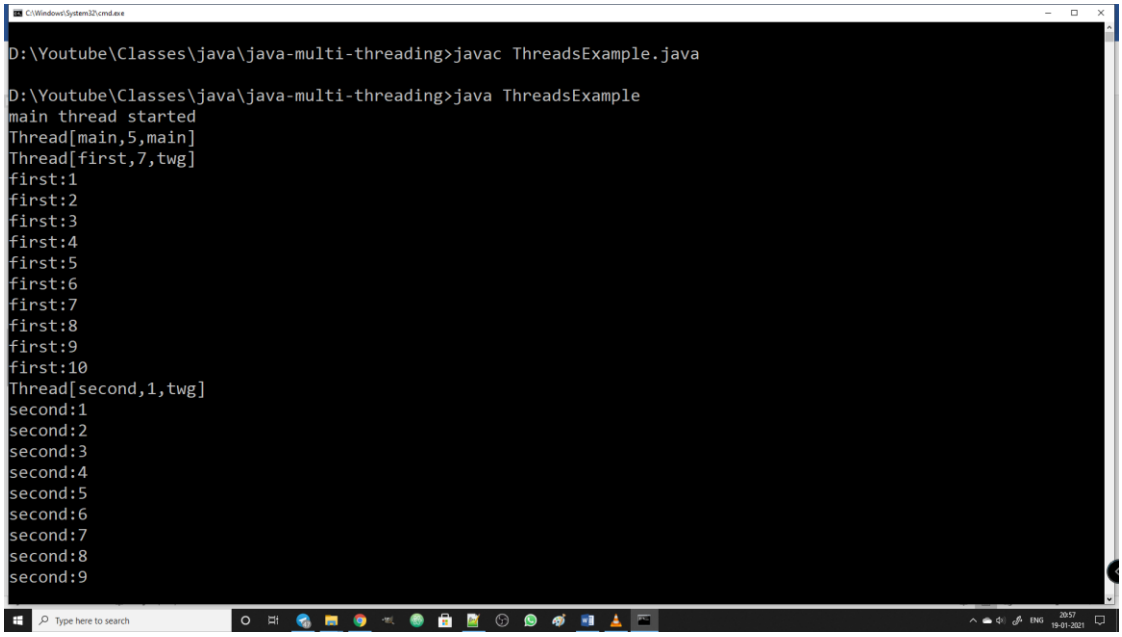
Example:

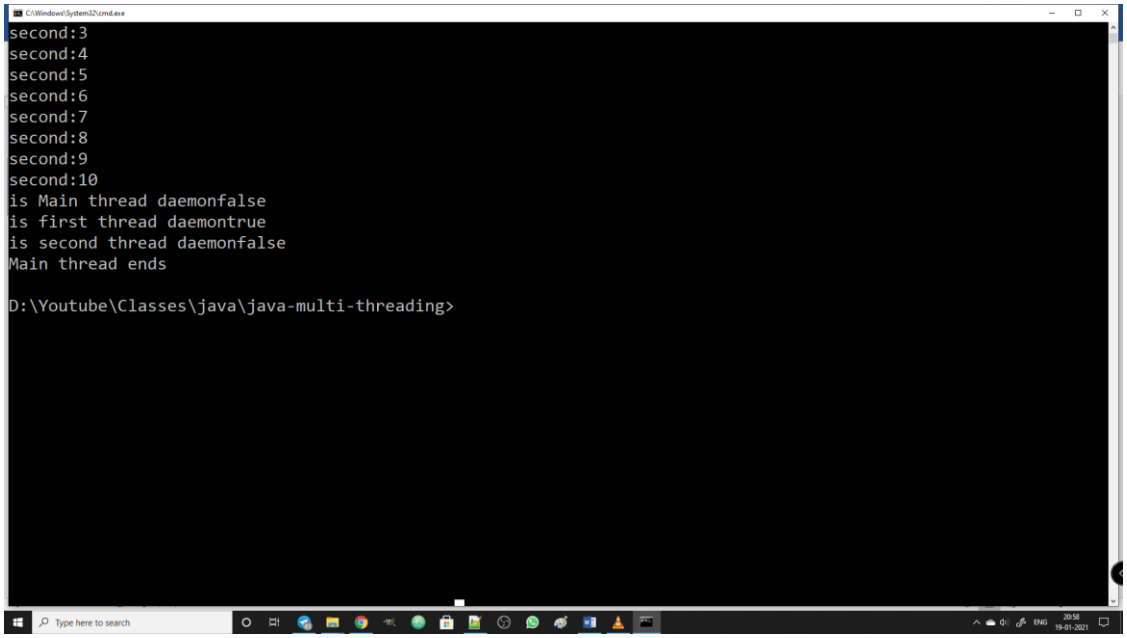
```
//program to demonstrate multithreading in java

//step-1 : create subclass of Thread class or Runnable interface
class MyThread extends Thread
{
        String tname;

        MyThread(String s)
        {
                tname = s;
        }

        //step-2 write run method
        public void run()
        {
                for(int i=1;i<=10;i++)
                {
                        System.out.println(tname+":"+i);

                        try{
                        Thread.sleep(1000);
                        }
                        catch(Exception e){}

                }

        }

}

class ThreadsExample
{
```

```
	public static void main(String ar[])
	{
		System.out.println("main thread started");

		Thread t = Thread.currentThread();
		System.out.println(t);

		ThreadGroup tg = new ThreadGroup("twg");


		//step-3 create  an object for above class
		MyThread th = new MyThread("first");

		//step-4 create an object for thread class and link it with above object th
		Thread tr = new Thread(tg,th);
		tr.setName("first");
		tr.setPriority(7);

		//set a user thread as daemon
		tr.setDaemon(true);

		System.out.println(tr);


		//step-5 : run the thread using start() method
		tr.start();

	    try{
		tr.join();
		}
		catch(Exception e){}


		//second child thread
		MyThread th1 = new MyThread("second");
		Thread tr1 = new Thread(tg,th1);
		tr1.setName("second");
		tr1.setPriority(Thread.MIN_PRIORITY);
		System.out.println(tr1);

		tr1.start();
		try{
		tr1.join();
		}
		catch(Exception e){}


		System.out.println("is Main thread daemon"+t.isDaemon());

		System.out.println("is first thread daemon"+tr.isDaemon());

		System.out.println("is second thread daemon"+tr1.isDaemon());


		System.out.println("Main thread ends");
```

```
        }

}
```

Output: (will be shown in multiple parts)





Threads can group together by using ThreadGroup class. We can make a thread as daemon (Server Thread) by using setDaemon(true) method.

One more point is , to call run method of a thread we need to use start() method instead of calling it directly.

Threads Priorities:

We can Change the priority of threads by using setPriority method. Threads with higher priority are executed in preference to threads with lower priority.

```
public final void setPriority(int newPriority)
```

Here we can pass the priority value between 1 to 10.

or we can set the priority by using any of the following constants also.

MAX_PRIORITY : The maximum priority that a thread can have.
NORM_PRIORITY : The default priority that is assigned to a thread.
MIN_PRIORITY : The minimum priority that a thread can have.

Note: Thread priority Example is already shown in previous topic.

How to Suspend / sleep a thread?

We can suspend a thread by using one of the following ways.

1) public static void sleep(long millis) :

   We can sleep a thread for a specified number of time (milliseconds) so that system will execute the next thread that is waiting for execution in the queue. once the specified time completes the thread will be added for ready queue again.

2) public final void suspend() & public final void resume()

   This method has been deprecated, as it is inherently deadlock-prone. If the target thread holds a lock on the monitor protecting a critical system resource when it is suspended, no thread can access this resource until the target thread is resumed.

3) public final void wait() & public final void notify() :

   Causes the current thread to wait until another thread invokes the notify() method or the notifyAll() method for this object.

join() Method in Thread class

join method will be useful to control the execution of threads. Waits for this thread to die.

Example:

```
t.join()
```

above statement will stop all the other threads until the currently running thread t finishes it's execution.

Thread Groups :

A thread group represents a set of threads. In addition, a thread group can also include other thread groups. The thread groups form a tree in which every thread group except the initial thread group has a parent. A thread is allowed to access information about its own thread group, but not to access information about its thread group's parent thread group or any other thread groups.

How to create a thread group ?

We can create a thread group by using ThreadGroup class in java.lang package.

```
ThreadGroup tg = new ThreadGroup("twg");
```

Now we can add the threads to above created group tg

```
Thread t = new Thread(tg,th)
```

Here th is the object of our already created thread class.

Daemon Threads :

Daemon threads are also called server threads. generally, these server threads will start with the system and ends with the system. These daemon threads provide service to the user threads.

We can convert any user thread as daemon thread by using setDaemon method.

```
t1.setDaemon(true);
```

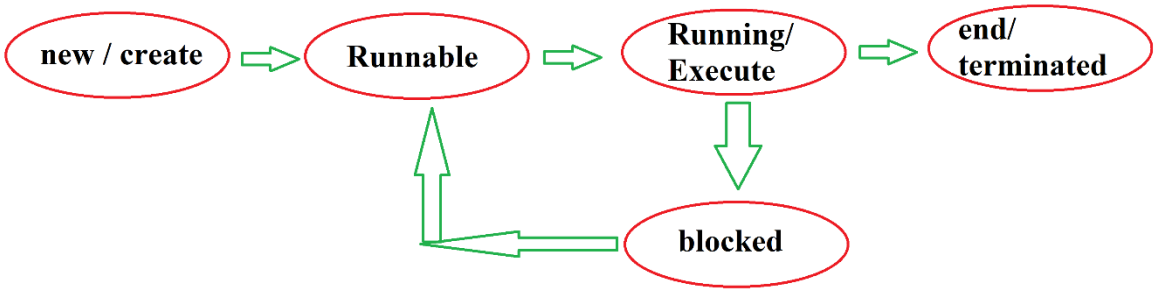
Above statement makes thread t1 as daemon thread.

We can check whether a thread is daemon thread or not by using isDaemon() method

```
if(t1.isDaemon())
{
   System.out.println("Yes t1 is daemon");
}
```

Thread Life Cycle

Every thread will go through different states in their execution process. These set of states are said to be life cycle.



The following are the set of states in thread life cycle

1)	new	:	While creating a thread it Is under this state
2)	runnable	:	When a thread is ready for execution it will be in this state.
3)	running / execute	:	Only one thread from runnable state will be selected for execution and it Is said to be in running / execute state.
4)	end / terminated	:	When a thread completes it s execution will be sent to end state.
5)	blocked	:	When a thread is waiting for an event or I/O then it will be into blocked state until the resources that are becomes available.

Thread Synchronization :

Synchronization in java is the capability to control the access of multiple threads to any shared resource. Java Synchronization is better option where we want to allow only one thread to access the shared resource.

Consider the case of “Movie ticket booking” where more than one person are competing for the same seat. if we cant control the access by allowing only one thread to book a seat, there may be chances of getting wrong results.

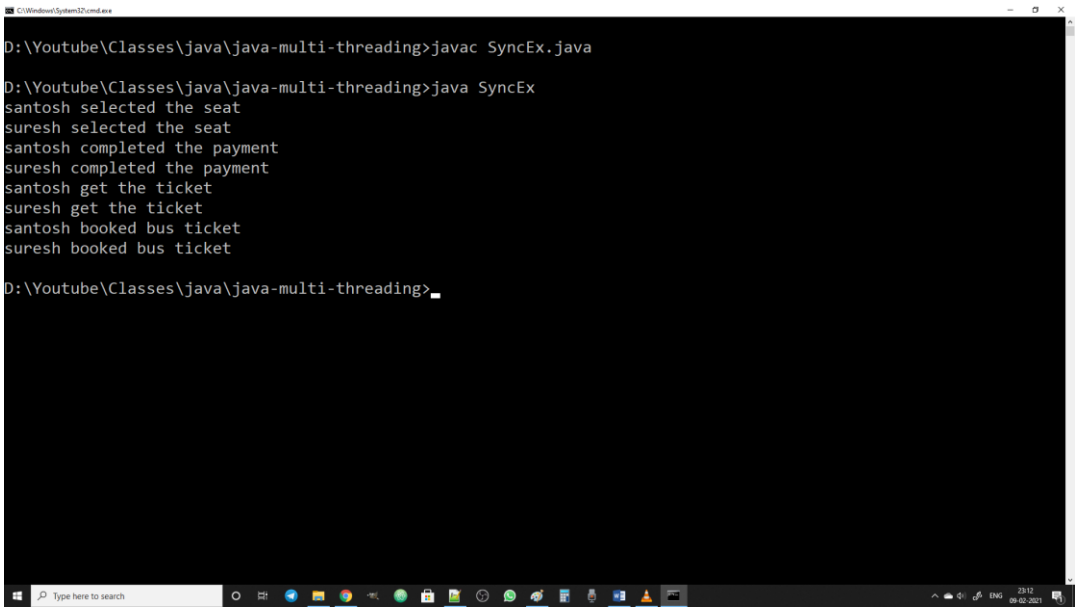
Example:

```
//program to demonstrate thread synchronization
class ticketbooking
{
        boolean booked=false;

        public  void bookTicket(String name)
        {

                if(booked==false){
                System.out.println(name+" selected the seat");
```

```
                    try{
                    Thread.sleep(1000);
                    }catch(Exception e){}


                    System.out.println(name+" completed the payment");

                    try{
                    Thread.sleep(1000);
                    }catch(Exception e){}

                    System.out.println(name+" get the ticket");
                    System.out.println(name+" booked bus ticket");
                    booked=true;
                    }
                    else
                    {
                     System.out.println("Sorry "+name+", This seat is already booked");
                    }



                    try{
                    Thread.sleep(1000);
                    }catch(Exception e){}


                    System.out.println(name+" booked bus ticket");

                    }



}

class threadclass extends Thread
{
    ticketbooking tkt;
            String name;

          threadclass(ticketbooking obj,String s)
          {
                    tkt=obj;
                    name=s;
          }
          public void run()
          {
                    tkt.bookTicket(name);
          }

}


class SyncEx
{
          public static void main(String ar[])
```

```
        {
                ticketbooking seat7 = new ticketbooking();

                threadclass santosh = new threadclass(seat7,"santosh");
                Thread san = new Thread(santosh);
                san.start();




                threadclass suresh = new threadclass(seat7,"suresh");
                Thread sur = new Thread(suresh);
                sur.start();


        }
}
```

Output



Here, as there is no control mechanism on ticket booking, system allowed both the persons to book the same seat.

Now we can control this by using synchronization concept. Synchronization allow only one thread to access the synchronized resource at a time.

This can be achieved by using one of the two ways:

1) syncrhronized block : if you want to apply synchronization to set of independent instructions then we can place them under a synchronized block
   synchronized
   {

   }

2) synchronized method : If we want to apply the synchronization to the entire method then we may declare the method with synchronized keyword

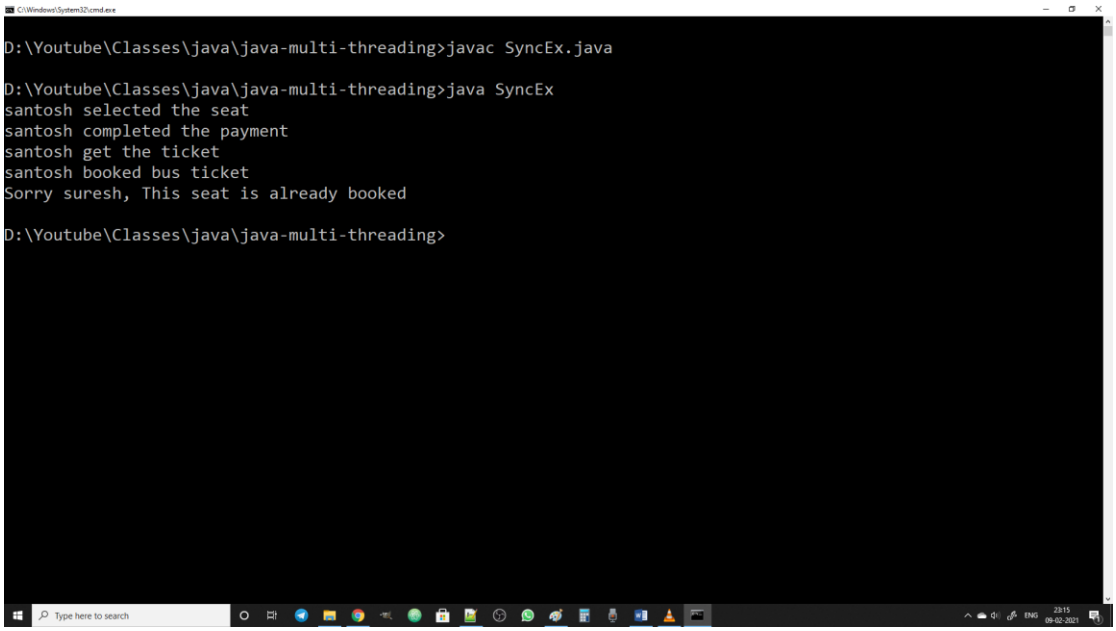
```
public void synchronized method_name()
{

}
```

Example with Synchronization:

```
//program to demonstrate thread synchronization
class ticketbooking
{
        boolean booked=false;

        public  void bookTicket(String name)
        {
                synchronized(this){

                if(booked==false){
                System.out.println(name+" selected the seat");

                try{
                Thread.sleep(1000);
                }catch(Exception e){}


                System.out.println(name+" completed the payment");

                try{
                Thread.sleep(1000);
                }catch(Exception e){}

                System.out.println(name+" get the ticket");
                System.out.println(name+" booked bus ticket");
                booked=true;
                }
                else
                {
                 System.out.println("Sorry "+name+", This seat is already booked");
                }

                }

                try{
                Thread.sleep(1000);
                }catch(Exception e){}




                }


}
```

```
class threadclass extends Thread
{
    ticketbooking tkt;
            String name;

        threadclass(ticketbooking obj,String s)
        {
                tkt=obj;
                name=s;
        }
        public void run()
        {
                tkt.bookTicket(name);
        }

}


class SyncEx
{
        public static void main(String ar[])
        {
                ticketbooking seat7 = new ticketbooking();

                threadclass santosh = new threadclass(seat7,"santosh");
                Thread san = new Thread(santosh);
                san.start();




                threadclass suresh = new threadclass(seat7,"suresh");
                Thread sur = new Thread(suresh);
                sur.start();


        }
}
```

Output

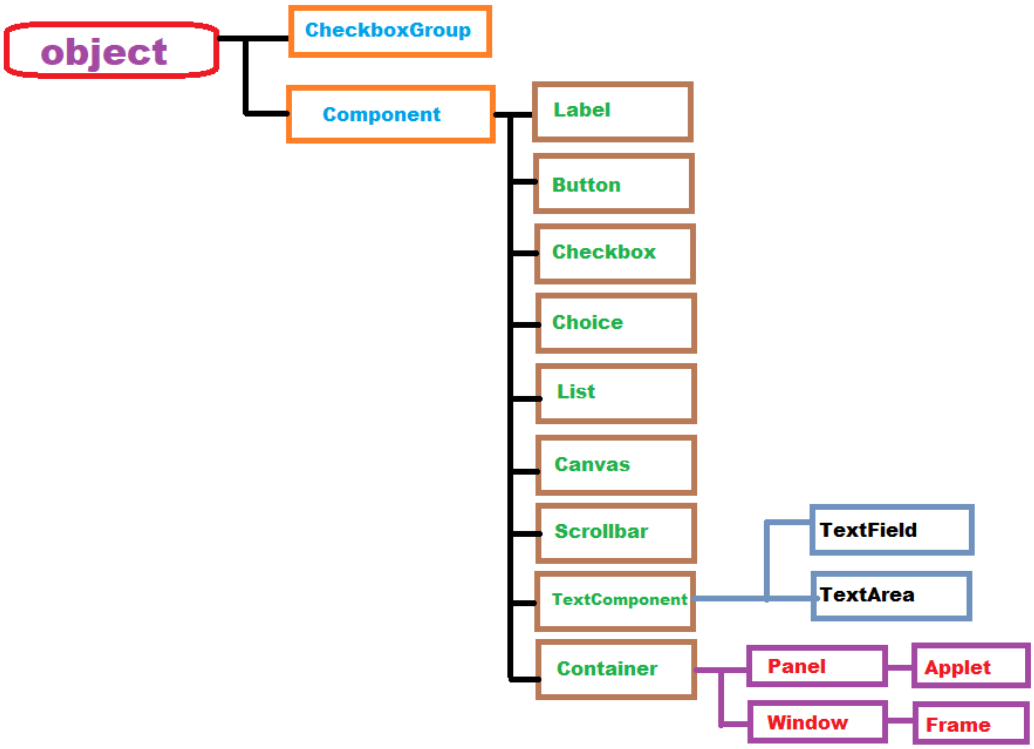


After applying synchronization only one person is allowed to book a seat.

AWT (Abstract Window Toolkit)

By using awt, we can develop Graphical User Interface (GUI) applications. java.awt package contains all of the classes for creating user interfaces and for painting graphics and images.

Now let us discuss about classes and interfaces in java.awt package.



Object:

Class Object is the root of the class hierarchy. Every class has Object as a superclass. All objects, including arrays, implement the methods of this class.

There are Two main classes in the hierarchy to develop GUI applications.

1) CheckboxGroup
2) Component

CheckboxGroup:

The CheckboxGroup class is used to group together a set of Checkbox buttons.

Exactly one check box button in a CheckboxGroup can be in the "on" state at any given time. Pushing any button sets its state to "on" and forces any other button that is in the "on" state into the "off" state.

Component:

A component is an object having a graphical representation that can be displayed on the screen and that can interact with the user. Examples of components are the buttons, checkboxes, and scrollbars of a typical graphical user interface.

Container:

A container object is a component that can contain other AWT components. Generally, we create container either by using Applet (sub class of Panel) or Frame (sub class of Window).

Let's discuss about frame later. now let us learn about how to create a container by using Frame

Frame:

A Frame is a top-level window with a title and a border. The size of the frame includes any area designated for the border.

The default layout for a frame is BorderLayout. (Let's discuss about Layout Manager in the next concept).

We can create a frame by using any of the following constructors

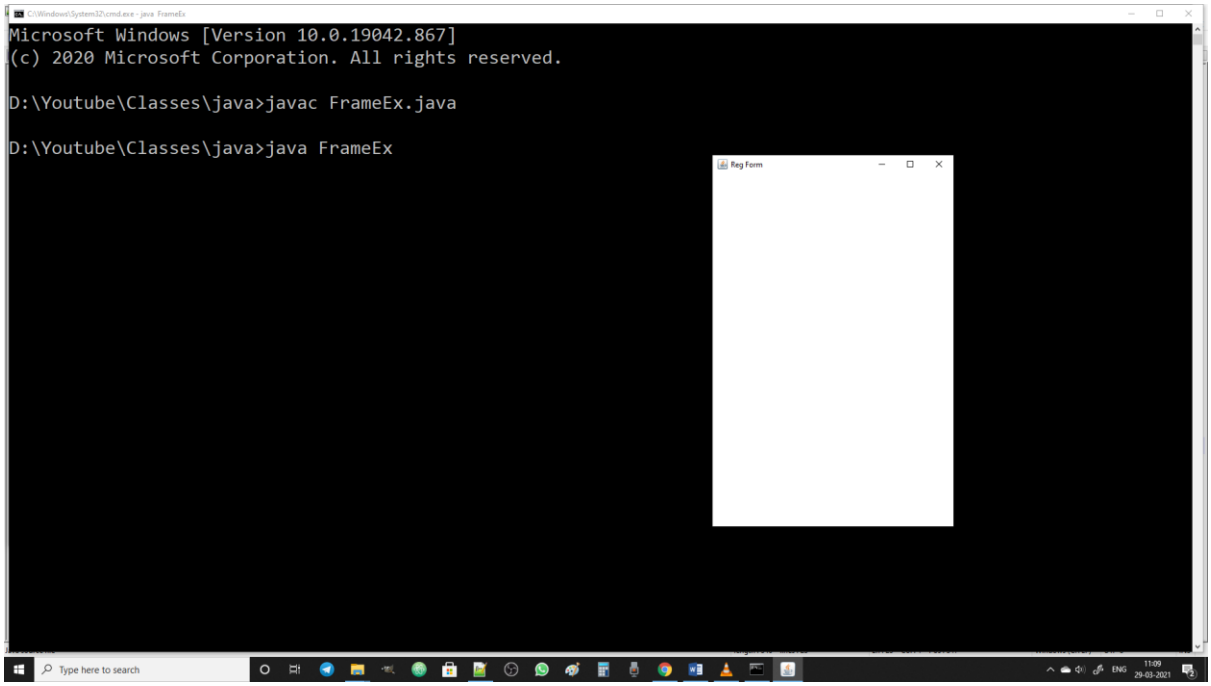
Frame() – Creates a frame with no title

Frame(String title) – creates a frame with the specified title

Example :

```
//program to demonstrate java.awt
import java.awt.*;
import java.awt.event.*;

class FrameEx
{
	public static void main(String ar[])
	{
		//create a frame
		Frame f = new Frame("Reg Form");

		//setting size of the container - frame
		f.setSize(400,600);

		//set visibility to true
		f.setVisible(true);



	}
}
```

output



Frame will be created as separate window. Now this frame window will work as a container for our application. we can add the remaining components as per our requirements and implement the application.

observe "Reg Form" is set as title for the frame window created.

We can set the dimensions of the output frame window by using setSize method.

By default, frame's visibility is set to false. We need to set it to true to make it visible.

Event Delegation Model:

This helps us in implementing all the actions that are related to the components in awt package.

Event :

Performing some action on a component is said to be an event.

Steps in the Event Delegation Model:

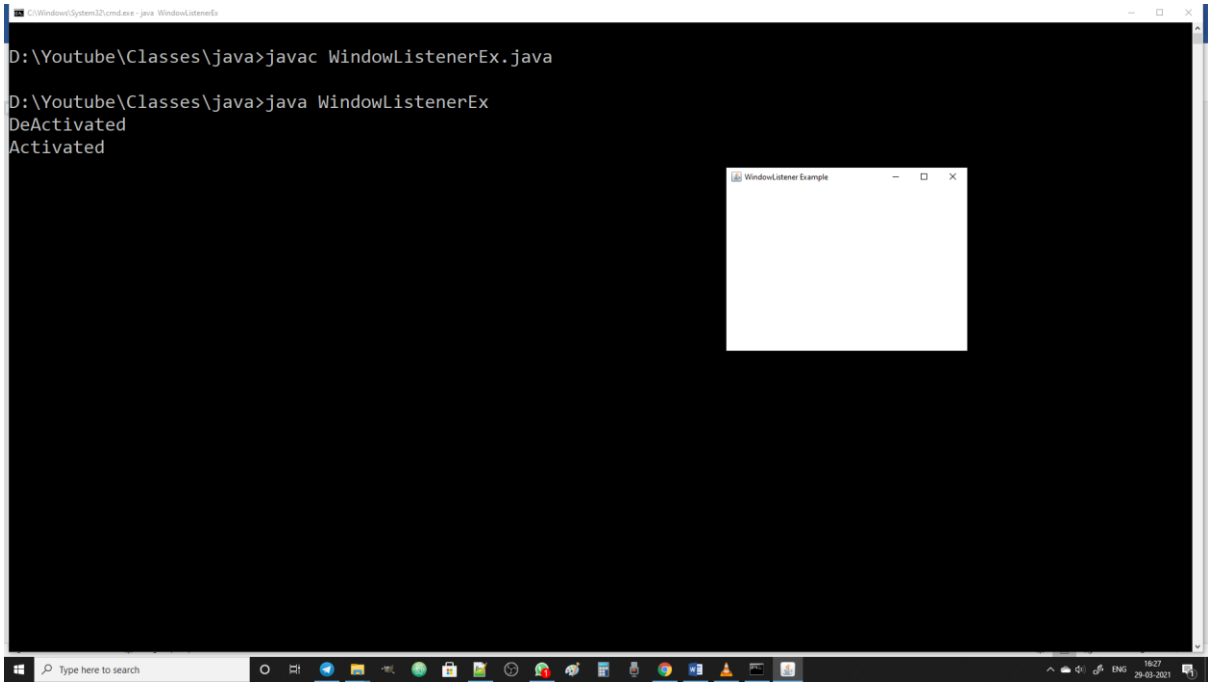
1) Add the component to the application.
2) Add Listeners to the components. Listeners are capable of sensing user actions on the components
3) Implement methods related to the listeners (Listener will listens the user actions and invoke this method immediately).

Example :

```
//program to demonstrate event delegation model
import java.awt.*;
import java.awt.event.*;

class WindowListenerEx
{
        public static void main(String ar[])
```

```
                {
                        //step-1 add the component
                   Frame f = new Frame("WindowListener Example");
                   f.setSize(400,300);
            f.setVisible(true);

            //step-2 add the listener
            f.addWindowListener(new MyListener());


                }
        }
        class MyListener implements WindowListener
        {
                //methods
                public void windowActivated(WindowEvent e){System.out.println("Activated");}
                public void windowClosed(WindowEvent e){System.out.println("Closed");}

                public void windowClosing(WindowEvent e)
                {
                        System.exit(0);
                }

                public void windowDeactivated(WindowEvent e){System.out.println("DeActivated");}
                public void windowDeiconified(WindowEvent e){System.out.println("DeIconified");}
                public void windowIconified(WindowEvent e){System.out.println("Iconified");}
                public void windowOpened(WindowEvent e){System.out.println("Opened");}

}
```

Output:



Observe the window is deactivated when you are switched to another window and activated again when it is invoked.

As WindowListener is an interface, we must implement all the methods of that interface in subclasses.

Observe the windowClosing method in which coding is provided. This method will be invoked on closing the output window.

System.exit method will forcibly close the output window.

Drawing Graphics in Frame :

Graphics class in java.awt package allows us to draw graphics in our container. The Graphics class is the abstract base class for all graphics contexts that allow an application to draw onto components that are realized on various devices, as well as onto off-screen images.

We use paint method to add different graphics to our container.

```
public void paint (Graphics g)
{
    //we write the code to draw ovals, text, arcs ….
}
```

Some methods in Graphics class :

drawLine(int x1, int y1, int x2, int y2) :
Draws a line, using the current color, between the points (x1, y1) and (x2, y2) in this graphics context's coordinate system.

drawOval(int x, int y, int width, int height):
Draws the outline of an oval.

fillOval(int x, int y, int width, int height):
Fills an oval bounded by the specified rectangle with the current color.

drawPolygon(int[] xPoints, int[] yPoints, int nPoints):
Draws a closed polygon defined by arrays of x and y coordinates.

fillPolygon(int[] xPoints, int[] yPoints, int nPoints):
Fills a closed polygon defined by arrays of x and y coordinates.

drawRect(int x, int y, int width, int height):
Draws the outline of the specified rectangle.

fillRect(int x, int y, int width, int height):
Fills the specified rectangle.

fillRoundRect(int x, int y, int width, int height, int arcWidth, int arcHeight):
Fills the specified rounded corner rectangle with the current color.

drawRoundRect(int x, int y, int width, int height, int arcWidth, int arcHeight):
Draws an outlined round-cornered rectangle using this graphics context's current color.

drawString(String str, int x, int y):
Draws the text given by the specified string, using this graphics context's current font and color.

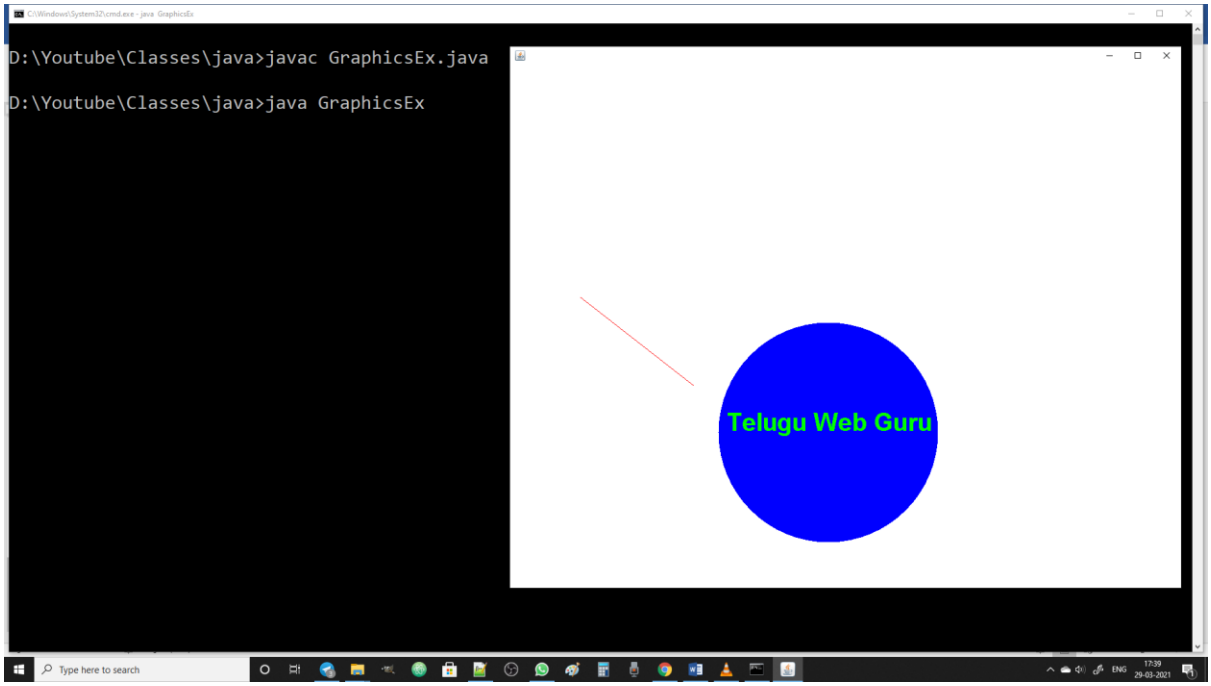
drawArc(int x, int y, int width, int height, int startAngle, int arcAngle):
Draws the outline of a circular or elliptical arc covering the specified rectangle.

fillArc(int x, int y, int width, int height, int startAngle, int arcAngle):
Fills a circular or elliptical arc covering the specified rectangle.

Example

```
//program to demonstrate Graphics
import java.awt.*;

class GraphicsEx extends Frame
{
	public void paint(Graphics g)
	{
		g.setColor(Color.red);
		g.drawLine(120, 400, 300, 540);
		g.setColor(Color.blue);
		g.fillOval(340,440,350,350);
		g.setColor(Color.green);
		Font fnt = new Font("Arial",Font.BOLD,40);
		g.setFont(fnt);
		g.drawString("Telugu Web Guru", 354,613);

	}

	public static void main(String ar[])
	{
		GraphicsEx f = new GraphicsEx();
		f.setSize(500,400);
		f.setVisible(true);


	}
}
```

output



Layout Managers :

Layout Managers are used to decide arrangement of components in the container.

There are 5 layouts.

1) Flow Layout
2) Border Layout
3) Grid Layout
4) Card Layout
5) GridBag Layout

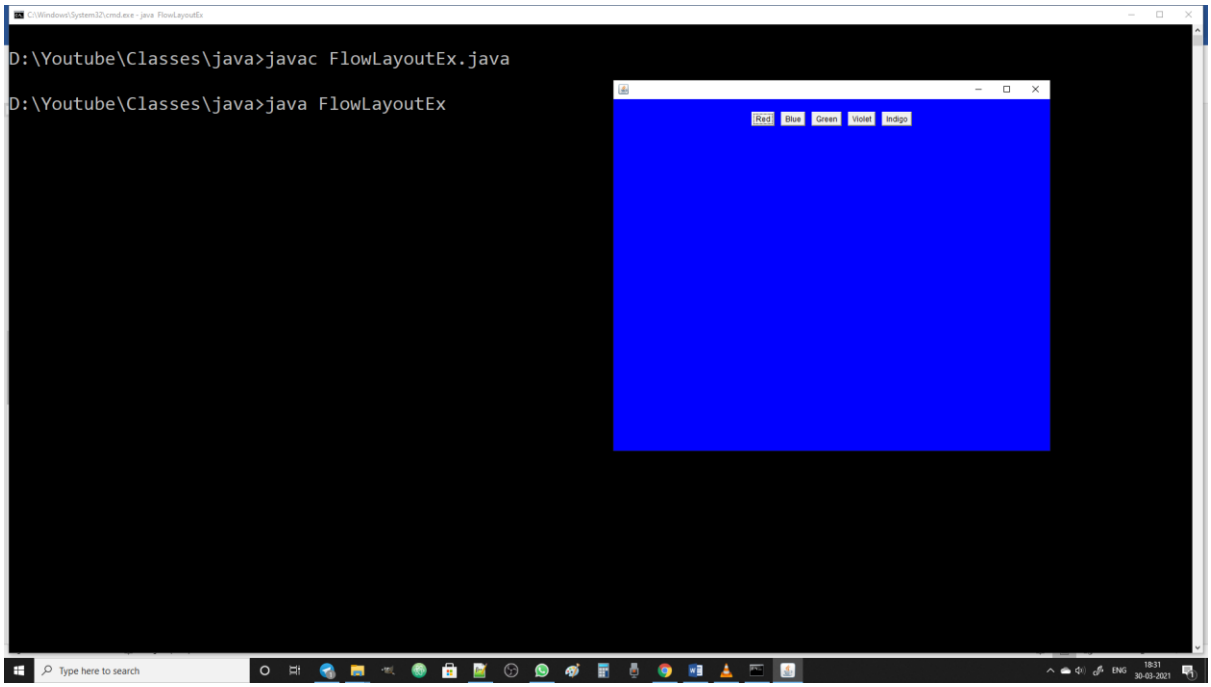
Flow Layout:

A flow layout arranges components in a directional flow, much like lines of text in a paragraph.

Example:

```
//program to demonstrate flowlayout
import java.awt.*;
class FlowLayoutEx extends Frame
{
	public static void main(String ar[])
	{
		FlowLayoutEx f = new FlowLayoutEx();
		f.setSize(300,400);
		f.setVisible(true);
		f.setBackground(Color.blue);

		//setting Layout
		FlowLayout fl = new FlowLayout(FlowLayout.CENTER,10,20);

		//setting layout to the frame
		f.setLayout(fl);

		//create a button
		Button b1 = new Button("Red");

		//add this button to the frame
		f.add(b1);

		Button b2 = new Button("Blue");
		f.add(b2);

		Button b3 = new Button("Green");
		f.add(b3);

		Button b4 = new Button("Violet");
		f.add(b4);

		Button b5 = new Button("Indigo");
		f.add(b5);
	}

}
```

output:



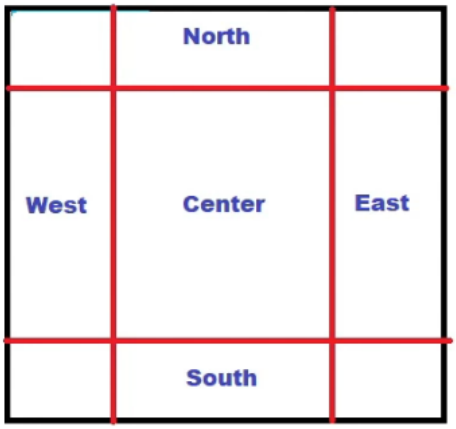
observe that flow layout will arranges the components from top to bottom and left to right by default.

We can set the layout by using setLayout method.

setBackground method is useful to set the background of the container.

Border Layout :

A border layout lays out a container, arranging and resizing its components to fit in five regions: north, south, east, west, and center. Each region may contain no more than one component, and is identified by a corresponding constant: NORTH, SOUTH, EAST, WEST, and CENTER.



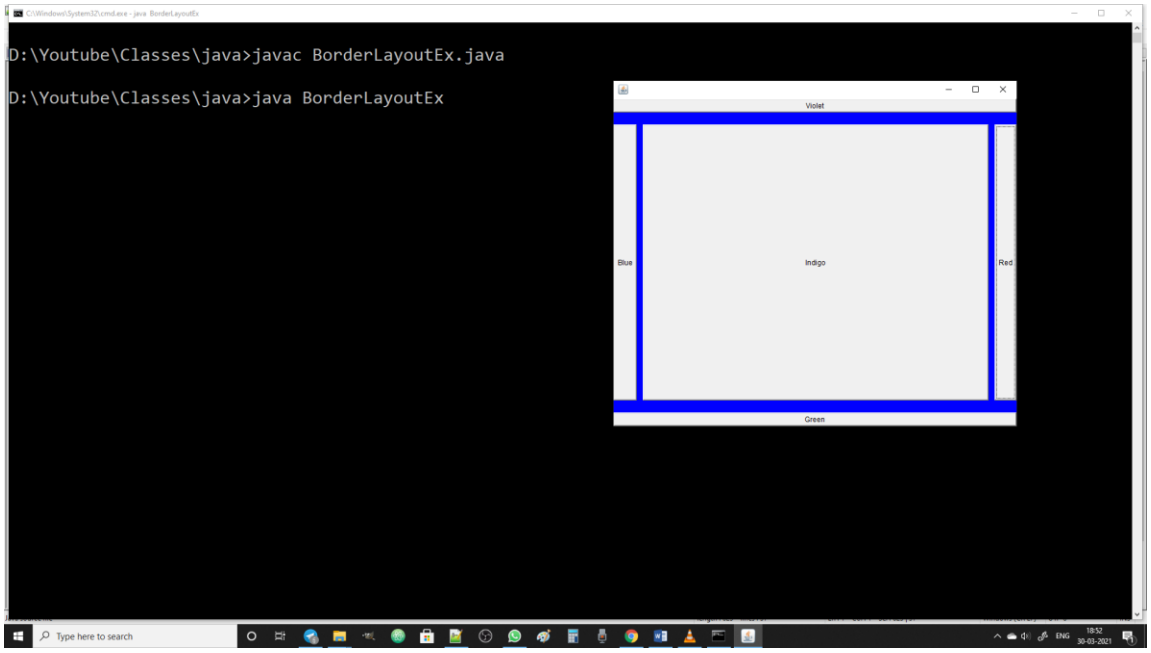
Example:

```
//program to demonstrate border layout
import java.awt.*;
class BorderLayoutEx extends Frame
{
	public static void main(String ar[])
	{
		BorderLayoutEx f = new BorderLayoutEx();
		f.setSize(300,400);
		f.setVisible(true);
		f.setBackground(Color.blue);

		//setting Layout
		BorderLayout bl = new BorderLayout(10,20);

		//setting layout to the frame
		f.setLayout(bl);

		//create a button
		Button b1 = new Button("Red");

		//add this button to the frame
		f.add(b1,BorderLayout.EAST);

		Button b2 = new Button("Blue");
		f.add(b2,BorderLayout.WEST);

		Button b3 = new Button("Green");
		f.add(b3,BorderLayout.SOUTH);

		Button b4 = new Button("Violet");
		f.add(b4,BorderLayout.NORTH);

		Button b5 = new Button("Indigo");
		f.add(b5,BorderLayout.CENTER);
	}

}
```

We can setup horizontal and vertical gaps between the components by passing the values in the BorderLayout constructor.

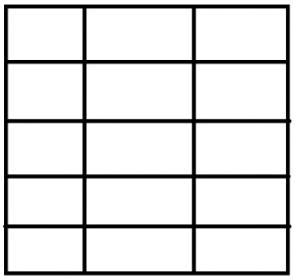
Observe the output below as each button is added in the layout as per alignments in the above program.

Output



GridLayout:

The GridLayout class is a layout manager that lays out a container's components in a rectangular grid. The container is divided into equal-sized rectangles, and one component is placed in each rectangle.



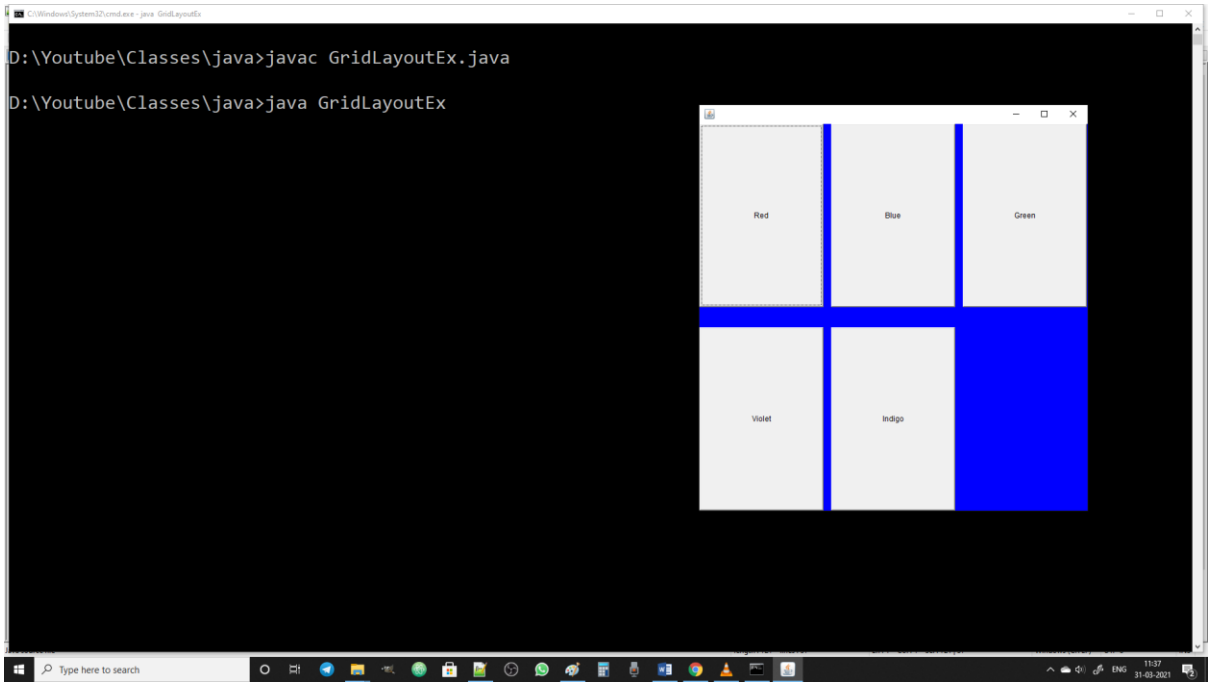
Example

```
//program to demonstrate GridLayout
import java.awt.*;
class GridLayoutEx extends Frame
{
	public static void main(String ar[])
	{
		GridLayoutEx f = new GridLayoutEx();
		f.setSize(300,400);
		f.setVisible(true);
		f.setBackground(Color.blue);

		//setting Layout
		GridLayout gl = new GridLayout(2,3,12,32);

		//setting layout to the frame
```

```
                f.setLayout(gl);

                //create a button
                Button b1 = new Button("Red");

                //add this button to the frame
                f.add(b1);

                Button b2 = new Button("Blue");
                f.add(b2);

                Button b3 = new Button("Green");
                f.add(b3);

                Button b4 = new Button("Violet");
                f.add(b4);

                Button b5 = new Button("Indigo");
                f.add(b5);
        }

}
```

one of the GridLayout's parameterized constructor accepts rows, columns, horizontal gap, vertical gap as arguments so that we can customize our output as per our requirements.

Output:



CardLayout:

A CardLayout object is a layout manager for a container. It treats each component in the container as a card. Only one card is visible at a time, and the container acts as a stack of cards. The first component added to a CardLayout object is the visible component when the container is first displayed.

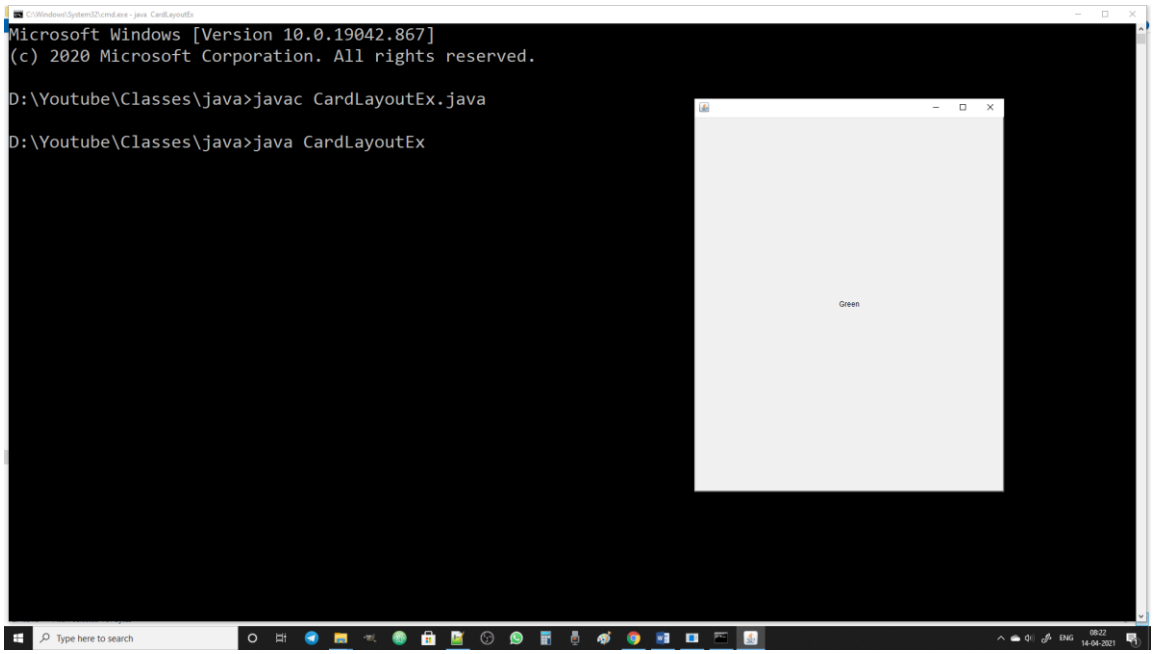
It has two constructors.

CardLayout()
CardLayout(int hgap, int vgap)

Example:

```
//program to demonstrate cardlayout
import java.awt.*;
class CardLayoutEx extends Frame
{
	public static void main(String ar[])
	{
		CardLayoutEx f = new CardLayoutEx();
		f.setSize(300,400);
		f.setVisible(true);
		f.setBackground(Color.blue);

		//setting Layout
		CardLayout cl = new CardLayout();

		//setting layout to the frame
		f.setLayout(cl);

		//create a button
		Button b1 = new Button("Red");

		//add this button to the frame
		f.add("Red",b1);

		Button b2 = new Button("Blue");
		f.add("Blue",b2);

		Button b3 = new Button("Green");
		f.add("Green",b3);

		Button b4 = new Button("Violet");
		f.add("Violet",b4);

		Button b5 = new Button("Indigo");
		f.add("Indigo",b5);

		cl.show(f,"Green");

	}

}
```

Output:



Observe only the card which is active will be shown (Green) and remaining card are hidden behind it.

GridBagLayout :

The GridBagLayout class is a flexible layout manager that aligns components vertically, horizontally or along their baseline without requiring that the components be of the same size. Each GridBagLayout object maintains a dynamic, rectangular grid of cells, with each component occupying one or more cells, called its display area. The GridBagConstraints class specifies constraints for components that are laid out using the GridBagLayout class.

public GridBagConstraints(int gridx,
int gridy,
int gridwidth,
int gridheight,
double weightx,
double weighty,
int anchor,
int fill,
Insets insets,
int ipadx,
int ipady)
Creates a GridBagConstraints object with all of its fields set to the passed-in arguments. Note: Because the use of this constructor hinders readability of source code, this constructor should only be used by automatic source code generation tools.

Parameters:
gridx - The initial gridx value.
gridy - The initial gridy value.
gridwidth - The initial gridwidth value.
gridheight - The initial gridheight value.
weightx - The initial weightx value.
weighty - The initial weighty value.
anchor - The initial anchor value.

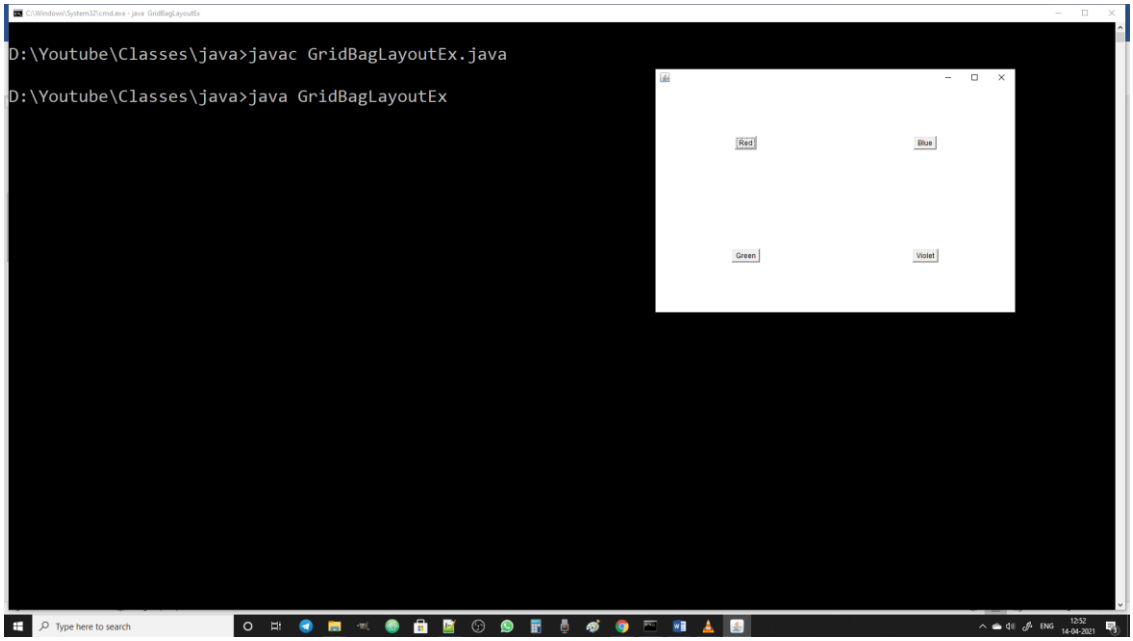
fill - The initial fill value.
insets - The initial insets value.
ipadx - The initial ipadx value.
ipady - The initial ipady value.

Example:

```
//program to demonstrate GridBagLayout
import java.awt.*;

class GridBagLayoutEx extends Frame
{
	public static void main(String ar[])
	{
		GridBagLayoutEx f = new GridBagLayoutEx();
		f.setSize(400,300);
		f.setVisible(true);

		GridBagLayout gl = new GridBagLayout();
		GridBagConstraints cons = new GridBagConstraints();

		f.setLayout(gl);

		cons.gridx=0;
		cons.gridy=0;

		cons.weightx=0.50;
		cons.weighty=0.30;

		Button b1 = new Button("Red");
		gl.setConstraints(b1,cons);
		f.add(b1);

		cons.gridx=1;
		cons.gridy=0;

		Button b2 = new Button("Blue");
		gl.setConstraints(b2,cons);
		f.add(b2);

		cons.gridx=0;
		cons.gridy=1;

		Button b3 = new Button("Green");
		gl.setConstraints(b3,cons);
		f.add(b3);

		cons.gridx=1;
		cons.gridy=1;

		Button b4 = new Button("Violet");
		gl.setConstraints(b4,cons);
		f.add(b4);


	}
```

```
}
```

Output



Now let us discuss about various components in java.awt package.

Button :

This class creates a labeled button. The application can cause some action to happen when the button is pushed.

It contains 2 constructors.

Button()
Constructs a button with an empty string for its label.
Button(String label)
Constructs a button with the specified label.

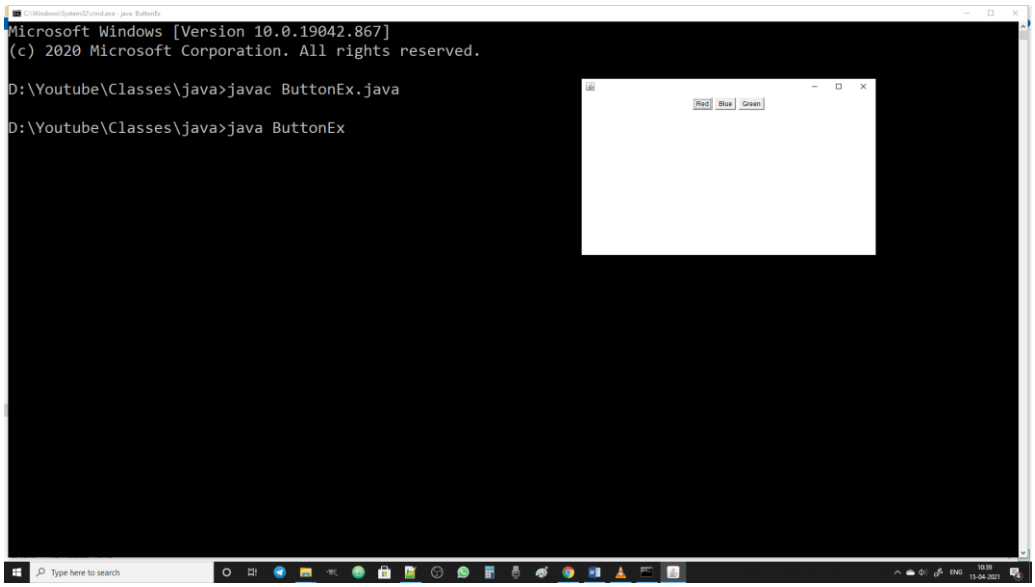
Example

```
//program to demonstrate Button in awt
import java.awt.*;
import java.awt.event.*;

class ButtonEx extends Frame implements ActionListener
{
	ButtonEx()
	{
		FlowLayout fl = new FlowLayout();
		this.setLayout(fl);
```

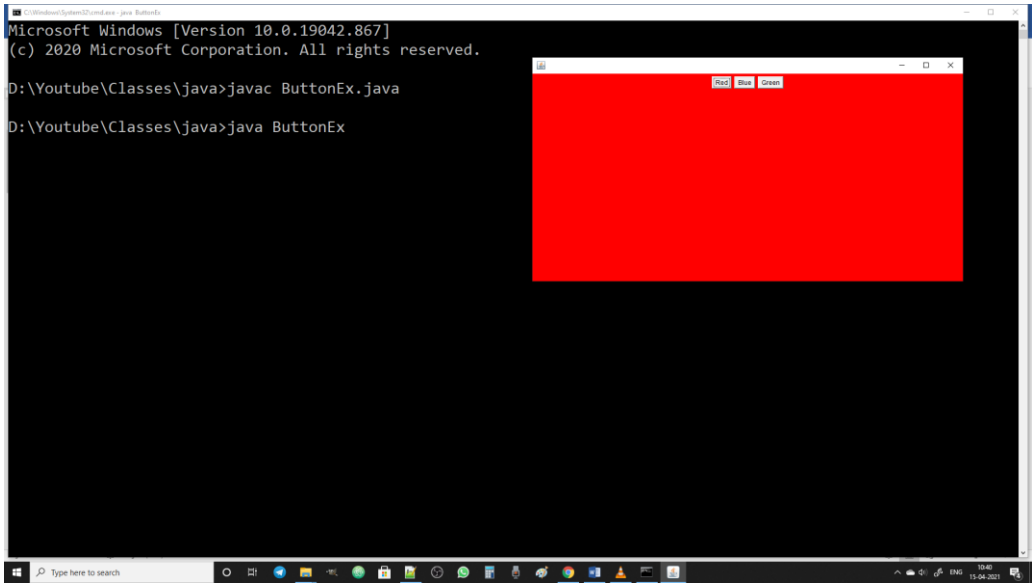
```
		//step-1 : create the component
		Button b1 = new Button("Red");

		//add component to the container - frame
		this.add(b1);

		//add listener to the component
		b1.addActionListener(this);


		Button b2 = new Button("Blue");
		this.add(b2);
		b2.addActionListener(this);


		Button b3 = new Button("Green");
		this.add(b3);
		b3.addActionListener(this);

	}

	public void actionPerformed(ActionEvent e)
	{
		String bname = e.getActionCommand();

		if(bname.equals("Red"))
			this.setBackground(Color.red);

		if(bname.equals("Blue"))
			this.setBackground(Color.blue);


		if(bname.equals("Green"))
			this.setBackground(Color.green);

	}


	public static void main(String ar[])
	{
		ButtonEx f = new ButtonEx();
		f.setSize(400,300);
		f.setVisible(true);



	}

}
```

Observe that the container is set with FlowLayout. Once Buttons are created, we need to add the corresponding listener that senses the click action on it. Related listener of Button is ActionListener. Generally ActionListener listens the click actions on the buttons and whenever it is performed it will execute the actionPerformed method.

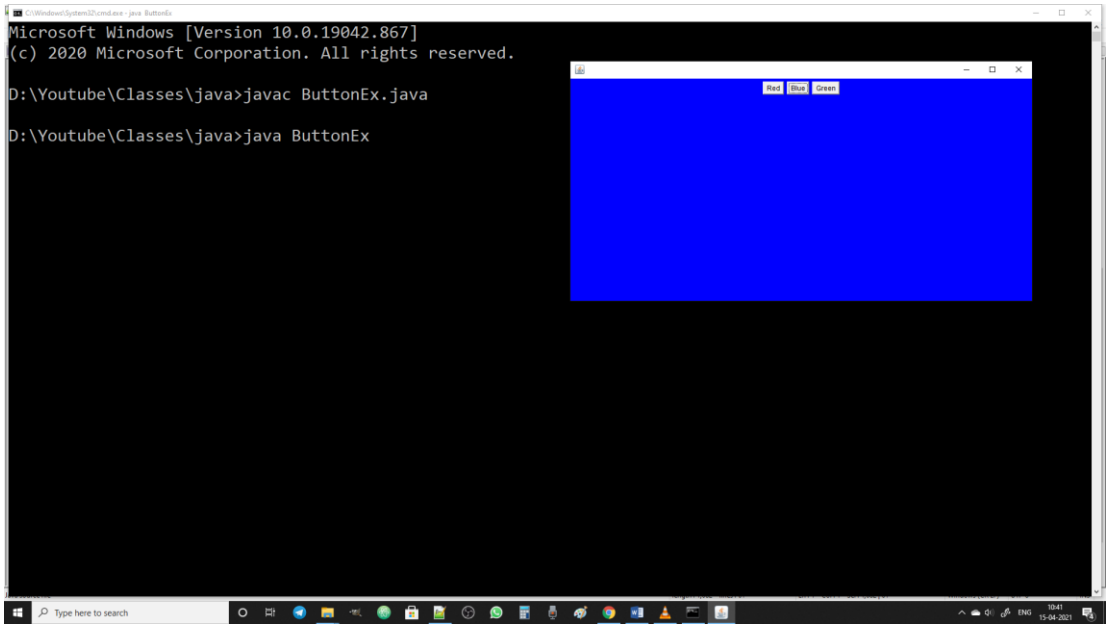
Output



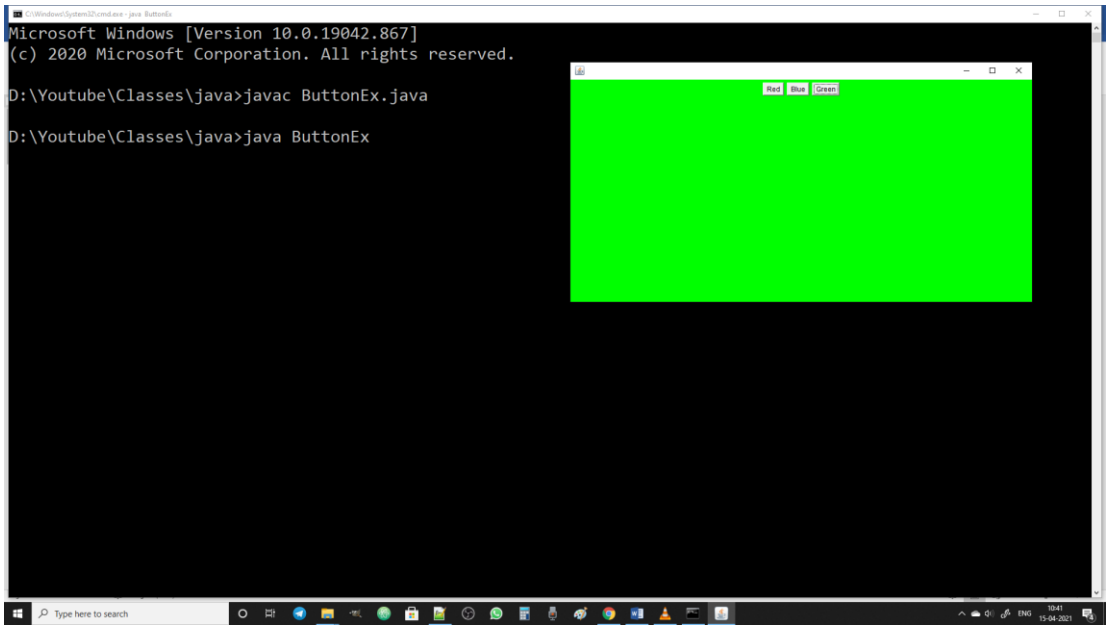
After clicked on Red



After clicked on Blue



after clicked on green



Checkbox:

A check box is a graphical component that can be in either an "on" (true) or "off" (false) state. Clicking on a check box changes its state from "on" to "off," or from "off" to "on."

Constructors available in Checkbox class:

Checkbox() : Creates a check box with an empty string for its label.

Checkbox(String label) : Creates a check box with the specified label.

Checkbox(String label, boolean state) : Creates a check box with the specified label and sets the specified state.

Checkbox(String label, boolean state, CheckboxGroup group)

Constructs a Checkbox with the specified label, set to the specified state, and in the specified check box group.

Checkbox(String label, CheckboxGroup group, boolean state)

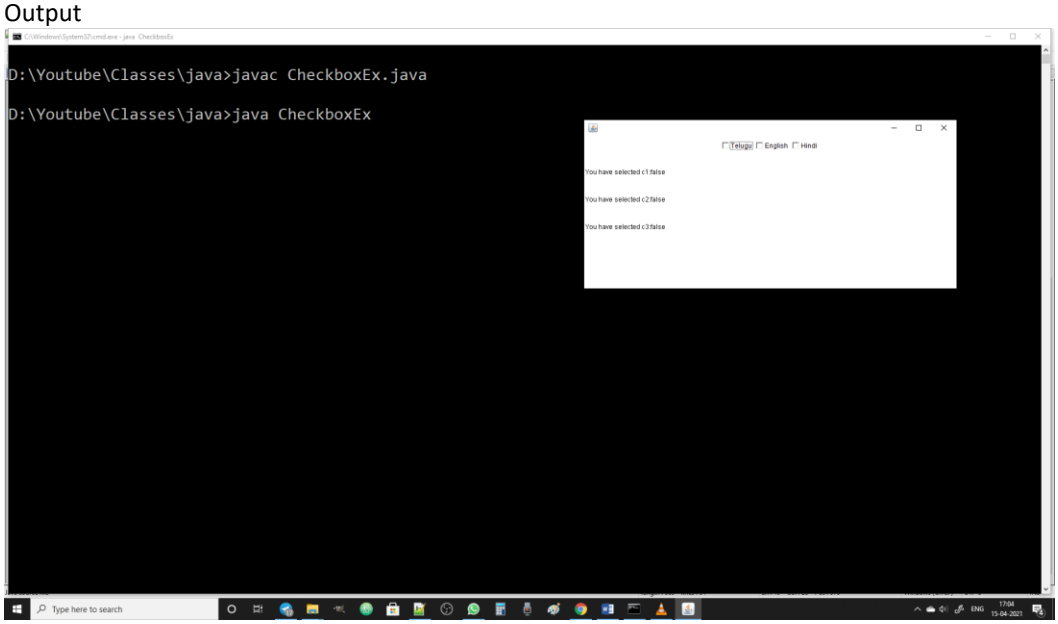
Creates a check box with the specified label, in the specified check box group, and set to the specified state.

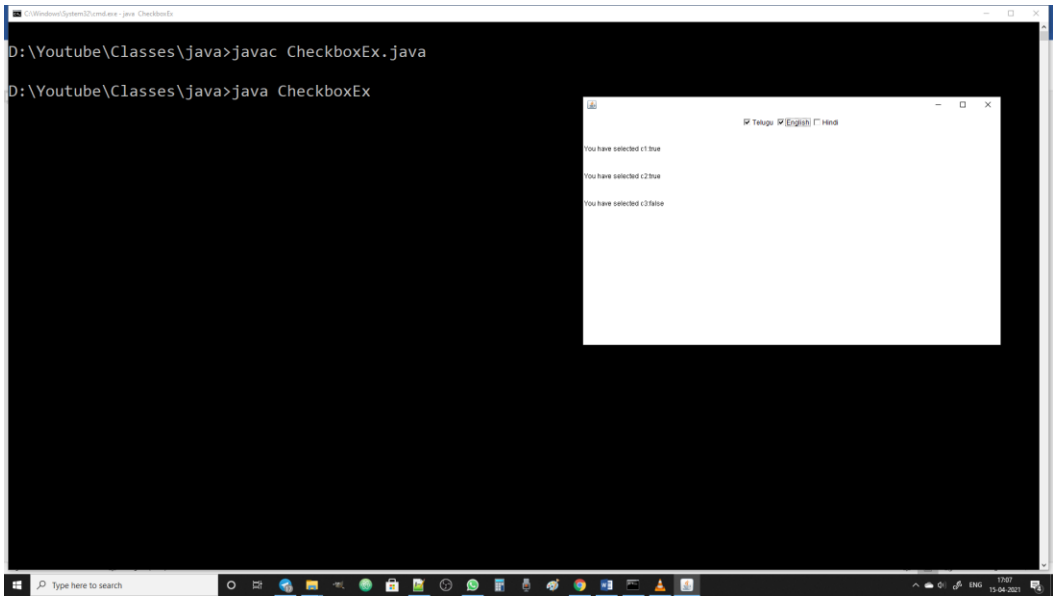
Example

```
//program to demonstrate checkbox in awt
import java.awt.*;
import java.awt.event.*;

class CheckboxEx extends Frame implements ItemListener
{
	Checkbox c1,c2,c3;


	CheckboxEx()
	{
		FlowLayout fl = new FlowLayout();
		this.setLayout(fl);


	  c1=new Checkbox("Telugu",false);
		this.add(c1);
		c1.addItemListener(this);

		c2=new Checkbox("English",false);
		this.add(c2);
		c2.addItemListener(this);

		c3=new Checkbox("Hindi",false);
		this.add(c3);
		c3.addItemListener(this);
	}

	public void paint(Graphics g)
	{
		g.drawString("You have selected c1:"+c1.getState(),10,100);
		g.drawString("You have selected c2:"+c2.getState(),10,150);
		g.drawString("You have selected c3:"+c3.getState(),10,200);

	}

	public void itemStateChanged(ItemEvent e)
	{
```

```
                repaint();

        }

        public static void main(String ar[])
        {

                CheckboxEx f =new CheckboxEx();
                f.setSize(400,300);
                f.setVisible(true);

        }


}
```

Output





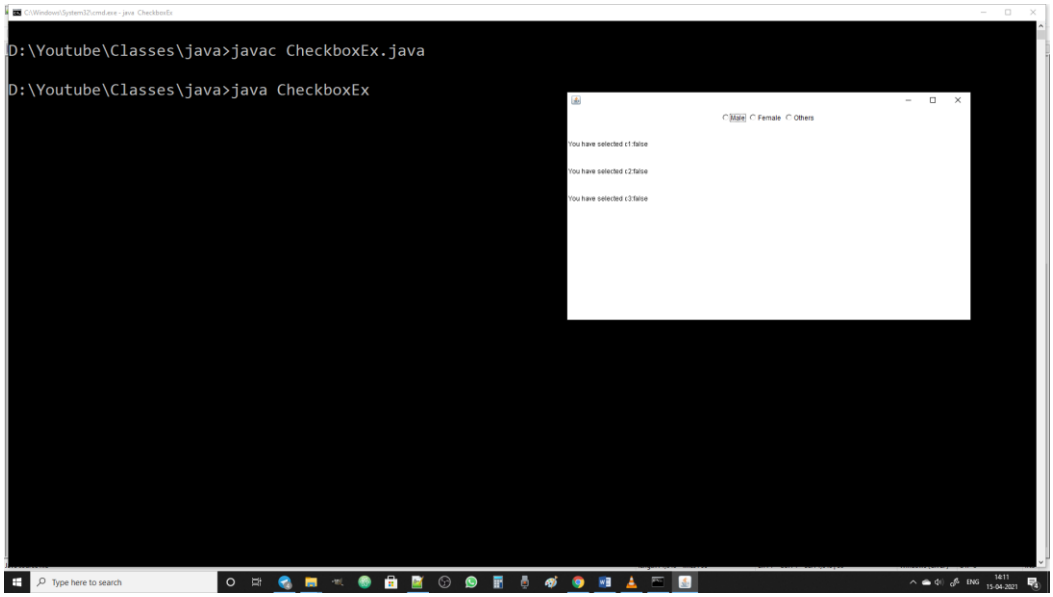
CheckboxGroup

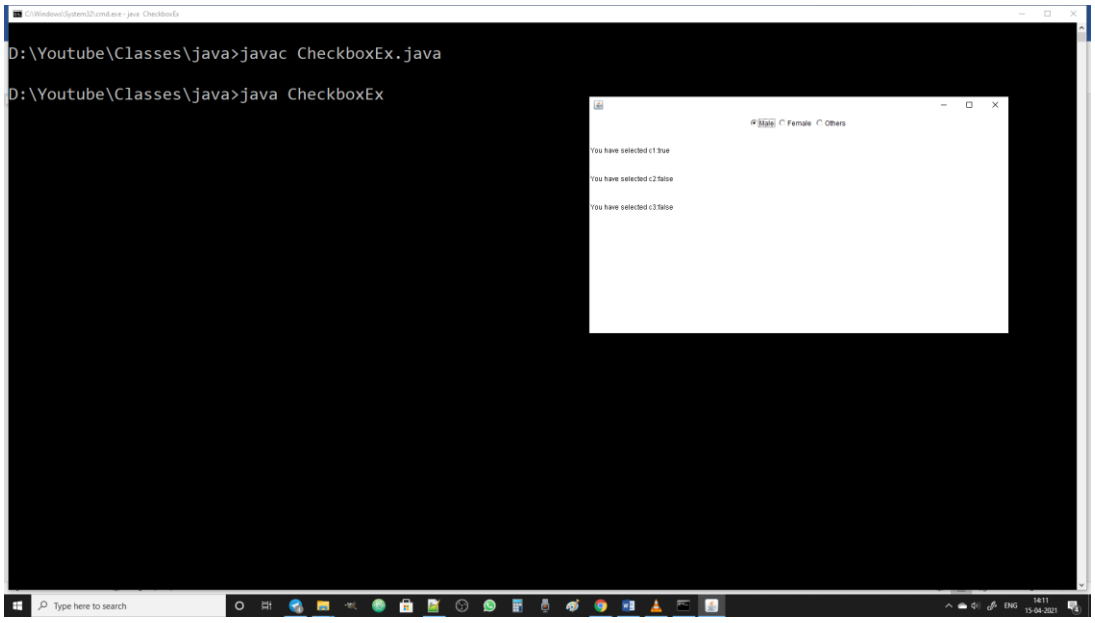
The CheckboxGroup class is used to group together a set of Checkbox buttons.
Exactly one check box button in a CheckboxGroup can be in the "on" state at any given time. Pushing any button sets its state to "on" and forces any other button that is in the "on" state into the "off" state.

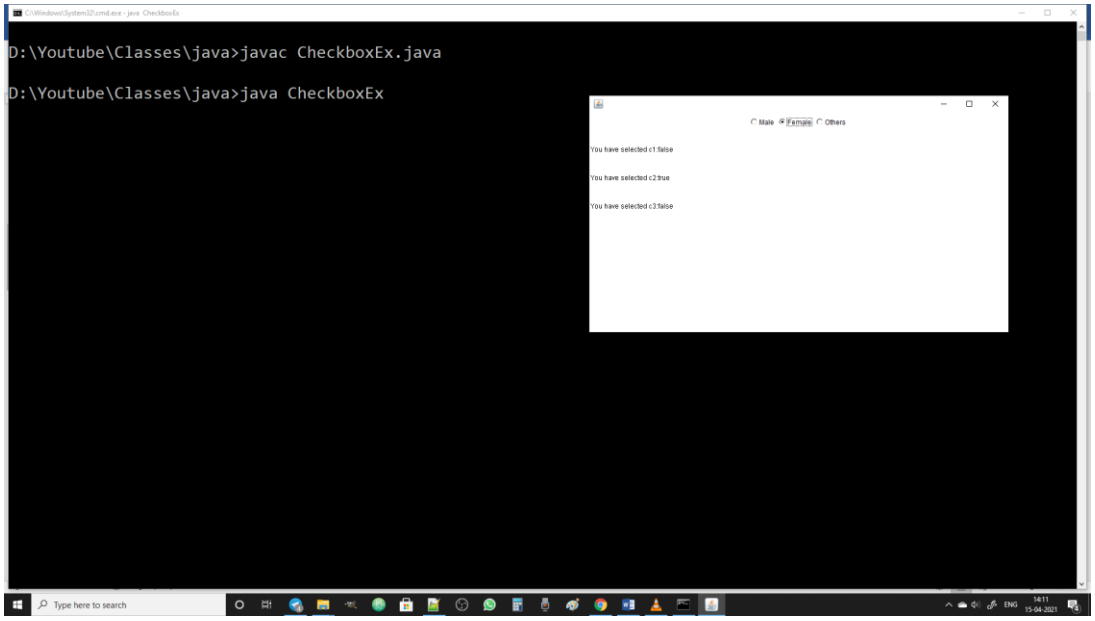
Example:

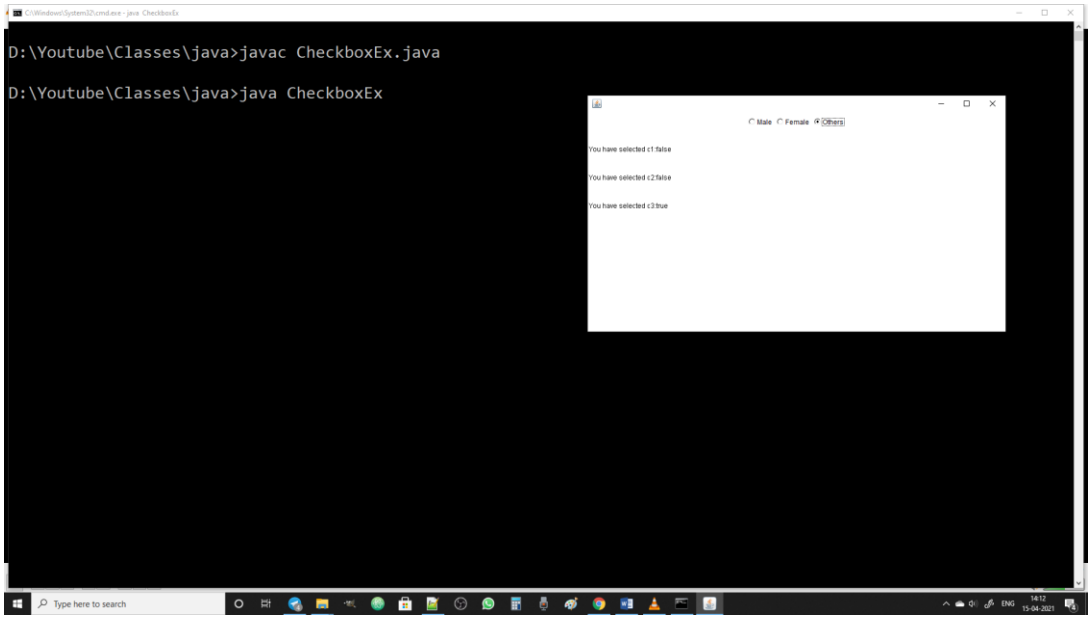
```
//program to demonstrate checkbox in awt
import java.awt.*;
import java.awt.event.*;

class CheckboxEx extends Frame implements ItemListener
{
	Checkbox c1,c2,c3;


	CheckboxEx()
	{
		FlowLayout fl = new FlowLayout();
		this.setLayout(fl);

		CheckboxGroup cbg = new CheckboxGroup();


	   c1=new Checkbox("Male",false,cbg);
		this.add(c1);
		c1.addItemListener(this);

		c2=new Checkbox("Female",false,cbg);
		this.add(c2);
		c2.addItemListener(this);

		c3=new Checkbox("Others",false,cbg);
		this.add(c3);
		c3.addItemListener(this);
	}

	public void paint(Graphics g)
	{
		g.drawString("You have selected c1:"+c1.getState(),10,100);
		g.drawString("You have selected c2:"+c2.getState(),10,150);
		g.drawString("You have selected c3:"+c3.getState(),10,200);

	}

	public void itemStateChanged(ItemEvent e)
	{

		repaint();

	}

	public static void main(String ar[])
	{
```

```
                CheckboxEx f =new CheckboxEx();
                f.setSize(400,300);
                f.setVisible(true);

        }


}
```









Label:

A Label object is a component for placing text in a container. A label displays a single line of read-only text.

Constructors:

Label() : Constructs an empty label.

Label(String text) : Constructs a new label with the specified string of text, left justified.

Label(String text, int alignment) : Constructs a new label that presents the specified string of text with the specified alignment.

TextField :

A TextField object is a text component that allows for the editing of a single line of text.

Constructors

TextField()
Constructs a new text field.
TextField(int columns)
Constructs a new empty text field with the specified number of columns.
TextField(String text)
Constructs a new text field initialized with the specified text.
TextField(String text, int columns)
Constructs a new text field initialized with the specified text to be displayed, and wide enough to hold the specified number of columns.

TextArea:

A TextArea object is a multi-line region that displays text. It can be set to allow editing or to be read-only.

Constructors:

TextArea()
Constructs a new text area with the empty string as text.
TextArea(int rows, int columns)
Constructs a new text area with the specified number of rows and columns and the empty string as text.
TextArea(String text)
Constructs a new text area with the specified text.
TextArea(String text, int rows, int columns)
Constructs a new text area with the specified text, and with the specified number of rows and columns.
TextArea(String text, int rows, int columns, int scrollbars)
Constructs a new text area with the specified text, and with the rows, columns, and scroll bar visibility as specified.

Choice :

The Choice class presents a pop-up menu of choices. The current choice is displayed as the title of the menu.

```
Choice ColorChooser = new Choice();
ColorChooser.add("Green");
ColorChooser.add("Red");
ColorChooser.add("Blue");
```

As the above example shows we can create a choice object and add the options into it by using add method.

List:

The List component presents the user with a scrolling list of text items. The list can be set up so that the user can choose either one item or multiple items. We can create List object and add elements same as choice component.

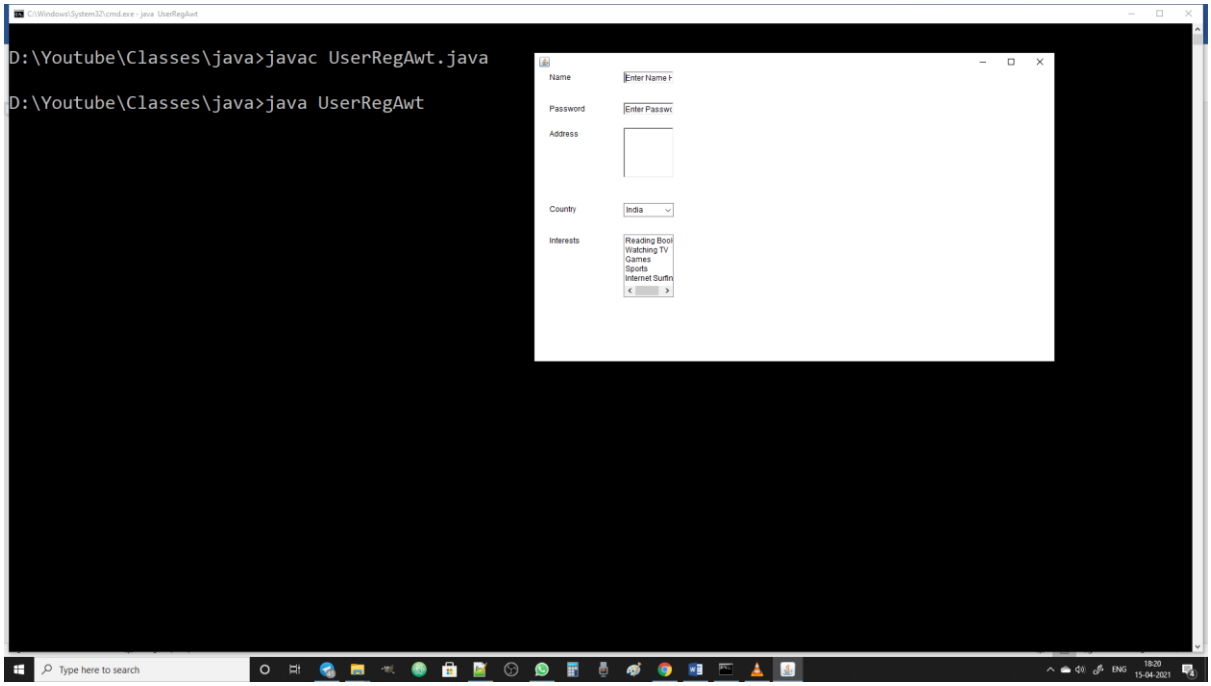
```
List lst = new List(4, false);
lst.add("Mercury");
```

Example

```
//program to demonstrate Label, TextField, TextArea, Choice, List

import java.awt.*;
import java.awt.event.*;

class UserRegAwt extends Frame
{
	Label l1,l2,l3,l4,l5;
	TextField tf1,tf2;
	TextArea ta1;
	Choice ch1;
	List li1;

	UserRegAwt()
	{

		this.setLayout(null);

		l1 = new Label("Name");
		l1.setBounds(30,30,80,20);
		this.add(l1);

		tf1 = new TextField("Enter Name Here",20);
		tf1.setBounds(150,30,80,20);
		this.add(tf1);

		l2 = new Label("Password");
		l2.setBounds(30,80,80,20);
		this.add(l2);


		tf2 = new TextField("Enter Password Here",20);
		tf2.setBounds(150,80,80,20);
		this.add(tf2);

		l3 = new Label("Address");
		l3.setBounds(30,120,80,20);
		this.add(l3);


		ta1 = new TextArea("",50,50,TextArea.SCROLLBARS_NONE);
		ta1.setBounds(150,120,80,80);
		this.add(ta1);


		l4 = new Label("Country");
		l4.setBounds(30,240,80,20);
		this.add(l4);


		ch1 = new Choice();
		ch1.add("India");
		ch1.add("US");
		ch1.add("UK");
		ch1.add("AUS");
```

```
                ch1.setBounds(150,240,80,80);
                this.add(ch1);

                l5 = new Label("Interests");
                l5.setBounds(30,290,80,20);
                this.add(l5);

                li1 = new List(3,true);
                li1.add("Reading Books");
                li1.add("Watching TV");
                li1.add("Games");
                li1.add("Sports");
                li1.add("Internet Surfing");

                li1.setBounds(150,290,80,100);
                this.add(li1);

        }

        public static void main(String ar[])
        {
                UserRegAwt f = new UserRegAwt();
                f.setSize(500,400);
                f.setVisible(true);




        }

}
```

Output:



Now let us see some more examples on awt and event delegation model.

Adapter Classes:

we use Adapters instead of listeners which gives more flexibility for developers in implementing methods. If we use Listener then we must implement all the methods of that interface into subclasses where as if we use adapter classes then we can implement the only methods that we want.

KeyEvents & KeyAdapter Example:

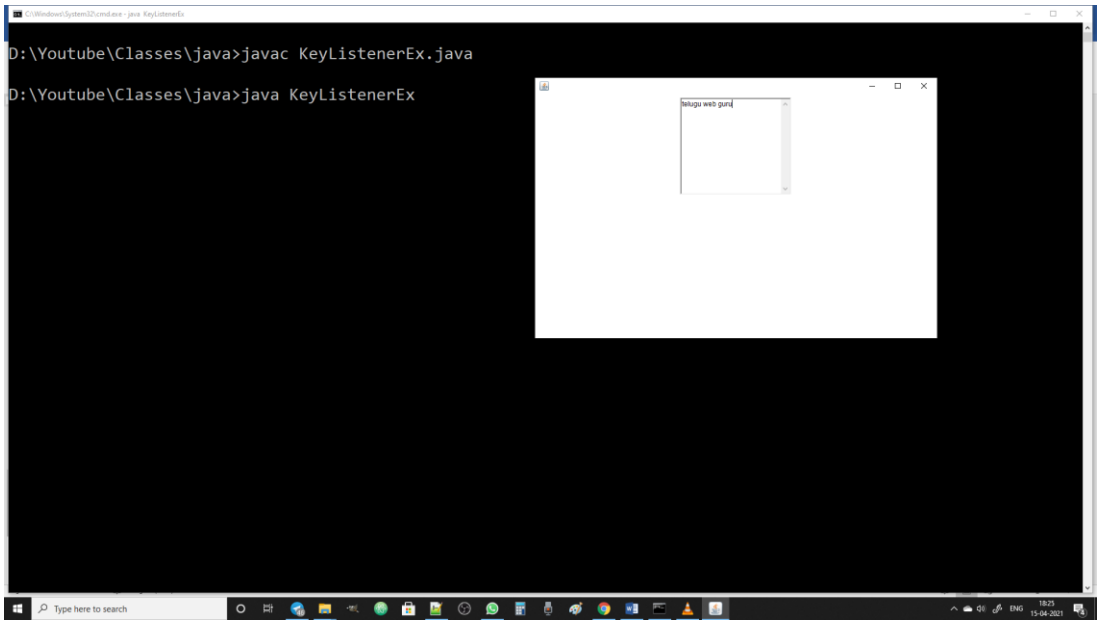
KeyEvents are implemented with the help of KeyListener. We need to implement various method available under KeyListener interface.

But observe that We just implemented keyPressed method instead of implementing all available methods.

```
//program to demonstrate key events, adapters, anonymous inner classes
import java.awt.*;
import java.awt.event.*;

class KeyListenerEx extends Frame implements KeyListener
{
	TextArea ta1;
	String msg="";

	KeyListenerEx()
	{
	  FlowLayout fl = new FlowLayout();
	  this.setLayout(fl);

	  ta1 = new TextArea("",10,25);
	  ta1.addKeyListener(new KeyAdapter(){
		public void keyPressed(KeyEvent e)
	{
		int code = e.getKeyCode();
		String kname = e.getKeyText(code);

		msg=code+" - "+kname;

		repaint();

	}

	});
    this.add(ta1);

	}


	public void paint(Graphics g)
	{
		g.drawString(msg,40,700);
	}
```

```
public static void main(String ar[])
        {
          KeyListenerEx f = new KeyListenerEx();
    f.setSize(500,400);
    f.setVisible(true);


        }

}
```

Output:



Anonymous Inner Classes :

It is an inner class without a name and for which only a single object is created. An anonymous inner class can be useful when making an instance of an object with certain “extras” such as overloading methods of a class or interface, without having to actually subclass a class.

Anonymous inner classes are useful in writing implementation classes for listener interface.

Observe below code

```
  ta1.addKeyListener(new KeyAdapter(){
        public void keyPressed(KeyEvent e)
{
        int code = e.getKeyCode();
        String kname = e.getKeyText(code);

        msg=code+" - "+kname;

        repaint();
```

```
        }

        });
```

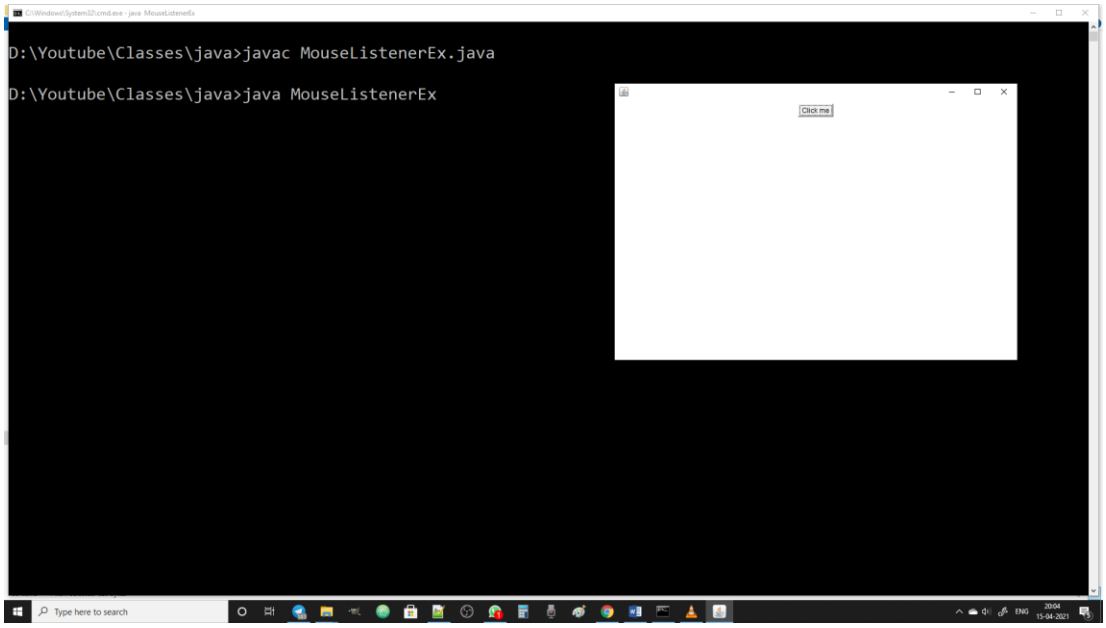
In this observe the class that is implemented in addKeyListener method (Red color text).

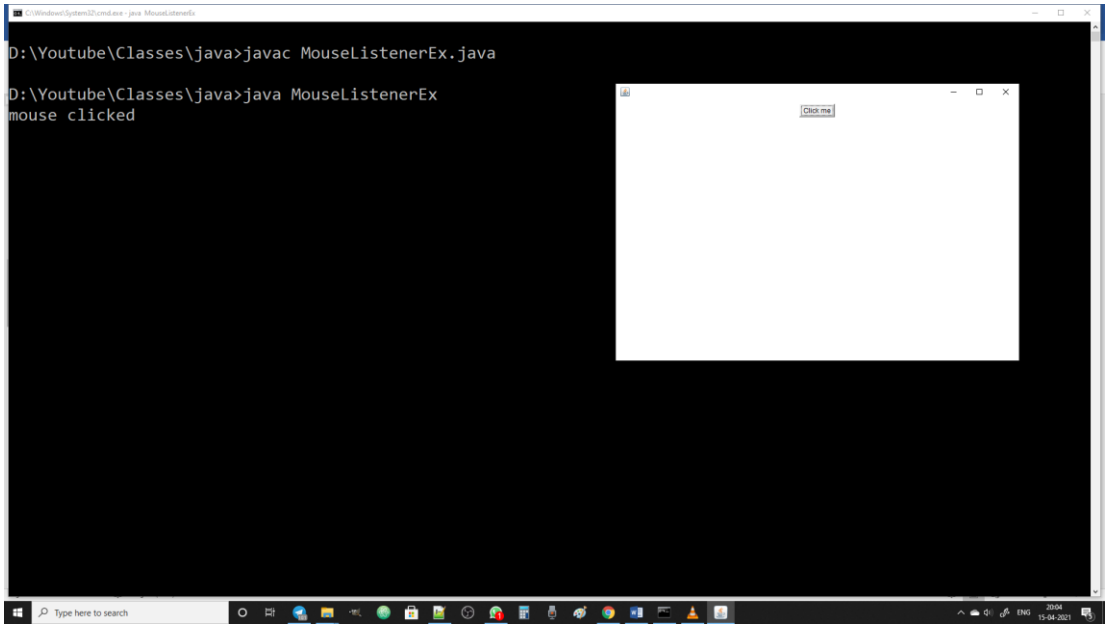
Eventhough this is a class that contains only a method keyPressed, It does not contains any name.

Example Program for Implementing MouseEvent :

```
//program to demonstrate MouseEvents
import java.awt.*;
import java.awt.event.*;

class MouseListenerEx extends Frame
{
        Button b1;

        MouseListenerEx()
        {
         FlowLayout fl = new FlowLayout();
         this.setLayout(fl);

         b1 = new Button("Click me");
         this.add(b1);
         b1.addMouseListener(new MouseAdapter(){
                public void mouseClicked(MouseEvent e)
           {
                 System.out.println("mouse clicked");
           }

         });

        }



        public static void main(String ar[])
        {
                MouseListenerEx f =new MouseListenerEx();
                f.setSize(450,350);
                f.setVisible(true);


        }

}
```

Output:


D:\Youtube\Classes\java>javac MouseListenerEx.java
D:\Youtube\Classes\java>java MouseListenerEx
Click me


D:\Youtube\Classes\java>javac MouseListenerEx.java
D:\Youtube\Classes\java>java MouseListenerEx
mouse clicked
Click me

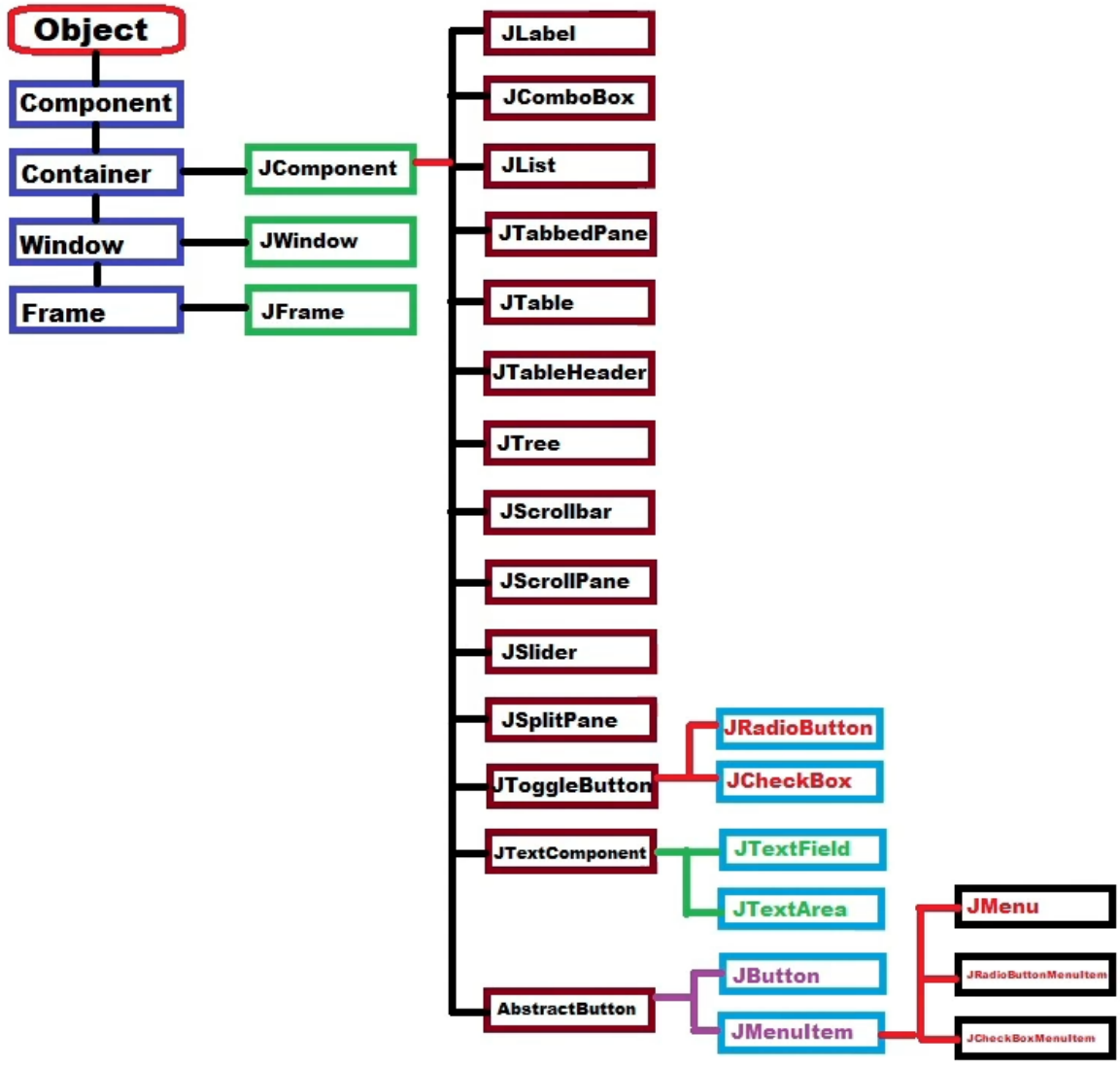
SWINGS IN JAVA

Java Swing tutorial is a part of Java Foundation Classes (JFC) that is used to create window-based applications. It is built on the top of AWT (Abstract Windowing Toolkit) API and entirely written in java.

Generally, awt components are dependent on C language functions in their implementation and C language is platform dependent. Unlike AWT, Java Swing provides platform-independent and lightweight components.

The javax.swing package provides classes for java swing API such as JButton, JTextField, JTextArea, JRadioButton, JCheckbox, JMenu, JColorChooser etc.

The following pic shows class hierarchy related to java swing.



Here JFrame will be used as container.

JFrame:

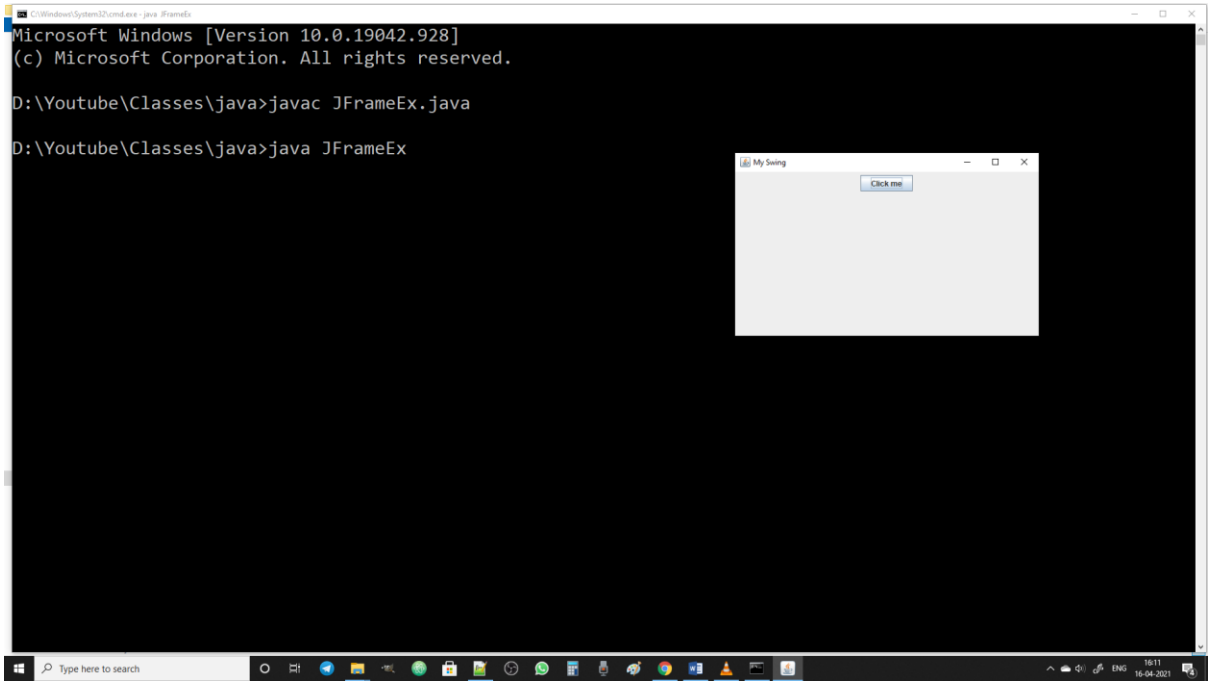
The JFrame class is slightly incompatible with Frame. Like all other JFC/Swing top-level containers, a JFrame contains a JRootPane as its only child. The content pane provided by the root pane should, as a rule, contain all the non-menu components displayed by the JFrame. This is different from the AWT Frame case. As a convenience, the add, remove, and setLayout methods of this class are overridden, so that they delegate calls to the corresponding methods of the ContentPane. For example, you can add a child component to a frame as follows:

```
frame.add(child);
```

Example

```
//program to demonstrate JFrame in java

import javax.swing.*;
import java.awt.*;

class JFrameEx
{
	public static void main(String ar[])
	{
		JFrame f = new JFrame("My Swing");

	   f.setSize(500,300);

		f.setVisible(true);

		f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

		FlowLayout fl = new FlowLayout();
		f.setLayout(fl);

		Container c = f.getContentPane();

		JButton b1 = new JButton("Click me");
		c.add(b1);

	}
}
```

Output



JComponent:

The base class for all Swing components except top-level containers. To use a component that inherits from JComponent, you must place the component in a containment hierarchy whose root is a top-level Swing container. Top-level Swing containers -- such as JFrame, JDialog, and JApplet -- are specialized components that provide a place for other Swing components to paint themselves.

The JComponent class provides:

- The base class for both standard and custom components that use the Swing architecture.
- A "pluggable look and feel" (L&F) that can be specified by the programmer or (optionally) selected by the user at runtime.
- Comprehensive keystroke handling
- Support for tool tips -- short descriptions that pop up when the cursor lingers over a component.
- Support for accessibility. JComponent contains all of the methods in the Accessible interface, but it doesn't actually implement the interface.

JLabel:

A display area for a short text string or an image, or both. A label does not react to input events. As a result, it cannot get the keyboard focus. A label can, however, display a keyboard alternative as a convenience for a nearby component that has a keyboard alternative but can't display it.

It contains following constructors

JLabel() :
Creates a JLabel instance with no image and with an empty string for the title.
JLabel(Icon image) :
Creates a JLabel instance with the specified image.
JLabel(Icon image, int horizontalAlignment)
Creates a JLabel instance with the specified image and horizontal alignment.
JLabel(String text): Creates a JLabel instance with the specified text.

JLabel(String text, Icon icon, int horizontalAlignment) : Creates a JLabel instance with the specified text, image, and horizontal alignment.

JLabel(String text, int horizontalAlignment) : Creates a JLabel instance with the specified text and horizontal alignment.

JButton :

This is swing version of Button

It contains following constructors

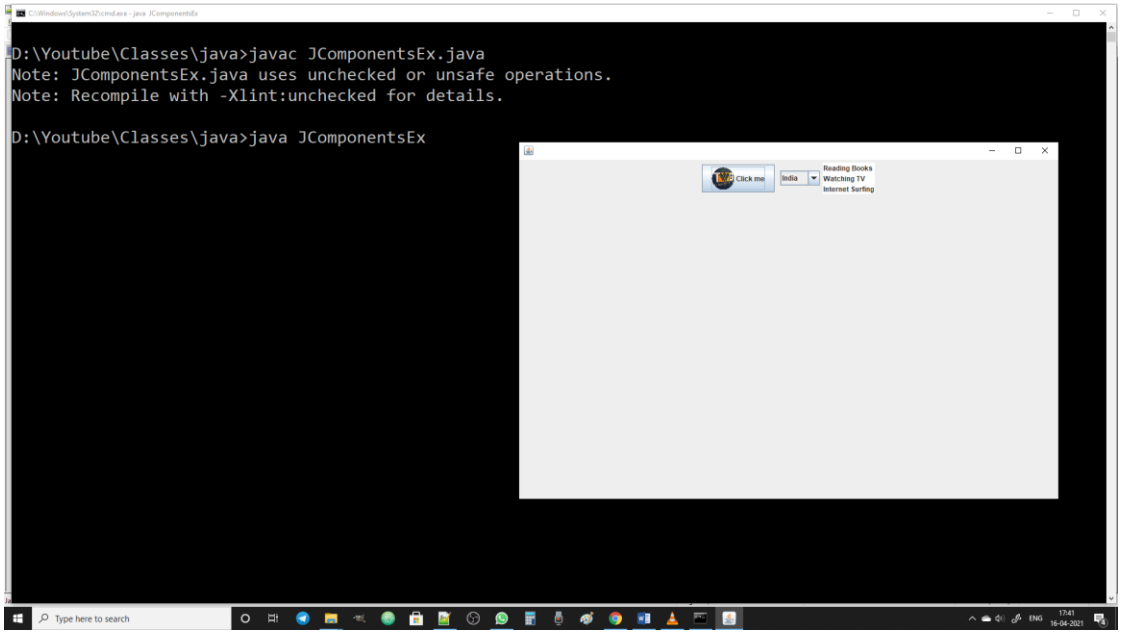
JButton()
Creates a button with no set text or icon.
JButton(Action a)
Creates a button where properties are taken from the Action supplied.
JButton(Icon icon)
Creates a button with an icon.
JButton(String text)
Creates a button with text.
JButton(String text, Icon icon)
Creates a button with initial text and an icon.

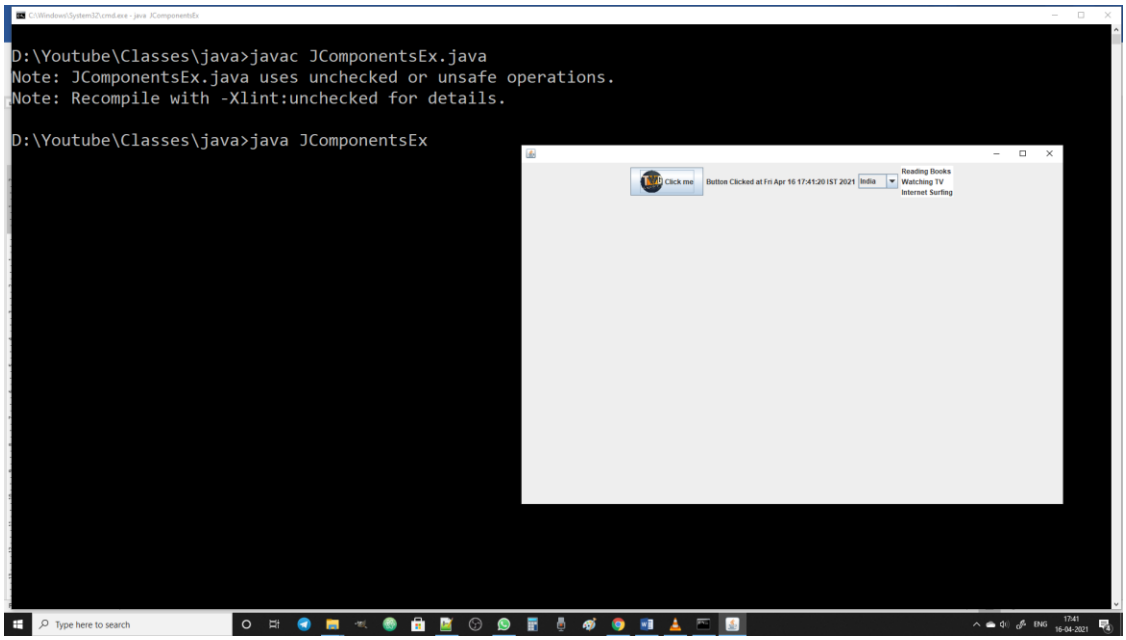
Example

```
//program to demonstrate jcomponents : JLabel, JButton
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
import java.util.*;

class JComponentsEx extends JFrame implements ActionListener
{
	JLabel jl;

	JComponentsEx()
	{
		this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

		FlowLayout fl = new FlowLayout();
		this.setLayout(fl);

		Container c = this.getContentPane();

		//Creating image icon object
		ImageIcon img = new ImageIcon("small.png");


		//JButton
		JButton b1 = new JButton("Click me",img);
		b1.addActionListener(this);

		c.add(b1);

		//JLabel
```

```
            jl = new JLabel();


                c.add(jl);

                //combo box
                String countries[] = {"India","AUS","Europe"};


                JComboBox jc = new JComboBox(countries);

                jc.addActionListener(this);

                c.add(jc);


                //JList
                JList list = new JList();
                String interests[]={"Reading Books","Watching TV","Internet Surfing"};
                list.setListData(interests);



                c.add(list);
        }

        public void actionPerformed(ActionEvent e)
        {
                jl.setText("Button Clicked at "+new Date());
        }

        public static void main(String ar[])
        {
                JComponentsEx f = new JComponentsEx();

                f.setSize(500,400);

                f.setVisible(true);




        }
}
```

Output


D:\Youtube\Classes\java>javac JComponentsEx.java
Note: JComponentsEx.java uses unchecked or unsafe operations.
Note: Recompile with -Xlint:unchecked for details.
D:\Youtube\Classes\java>java JComponentsEx
Click me
India
Reading Books
Watching TV
Internet Surfing


D:\Youtube\Classes\java>javac JComponentsEx.java
Note: JComponentsEx.java uses unchecked or unsafe operations.
Note: Recompile with -Xlint:unchecked for details.
D:\Youtube\Classes\java>java JComponentsEx
Click me
Button Clicked at Fri Apr 16 17:41:20 IST 2021
India
Reading Books
Watching TV
Internet Surfing

JCheckBox :

An implementation of a check box -- an item that can be selected or deselected, and which displays its state to the user. By convention, any number of check boxes in a group can be selected.

Constructors available in this JCheckBox:

JCheckBox ()
Creates an initially unselected check box button with no text, no icon.
JCheckBox (Action a)
Creates a check box where properties are taken from the Action supplied.
JCheckBox (Icon icon)
Creates an initially unselected check box with an icon.
JCheckBox (Icon icon, boolean selected)
Creates a check box with an icon and specifies whether or not it is initially selected.
JCheckBox (String text)
Creates an initially unselected check box with text.
JCheckBox (String text, boolean selected)
Creates a check box with text and specifies whether or not it is initially selected.
JCheckBox (String text, Icon icon)
Creates an initially unselected check box with the specified text and icon.
JCheckBox (String text, Icon icon, boolean selected)
Creates a check box with text and icon, and specifies whether or not it is initially selected.

JRadioButton:

An implementation of a radio button -- an item that can be selected or deselected, and which displays its state to the user. Used with a ButtonGroup object to create a group of buttons in which only one button at a time can be selected.

Constructors available in this JRadioButton:

JRadioButton ()
Creates an initially unselected radio button with no set text.
JRadioButton (Action a)
Creates a radiobutton where properties are taken from the Action supplied.
JRadioButton (Icon icon)
Creates an initially unselected radio button with the specified image but no text.
JRadioButton (Icon icon, boolean selected)
Creates a radio button with the specified image and selection state, but no text.
JRadioButton (String text)
Creates an unselected radio button with the specified text.
JRadioButton (String text, boolean selected)
Creates a radio button with the specified text and selection state.
JRadioButton (String text, Icon icon)
Creates a radio button that has the specified text and image, and that is initially unselected.
JRadioButton (String text, Icon icon, boolean selected)
Creates a radio button that has the specified text, image, and selection state.

JTextField:

JTextField is a lightweight component that allows the editing of a single line of text.

Constructors available in JTextField:

JTextField ()
Constructs a new TextField.
JTextField (Document doc, String text, int columns)
Constructs a new JTextField that uses the given text storage model and the given number of columns.
JTextField (int columns)
Constructs a new empty TextField with the specified number of columns.
JTextField (String text)
Constructs a new TextField initialized with the specified text.
JTextField (String text, int columns)
Constructs a new TextField initialized with the specified text and columns.

JTextArea:

A JTextArea is a multi-line area that displays plain text. It is intended to be a lightweight component that provides source compatibility with the java.awt.TextArea class where it can reasonably do so.

It has the following Constructors:

JTextArea ()
Constructs a new TextArea.
JTextArea (Document doc)
Constructs a new JTextArea with the given document model, and defaults for all of the other arguments (null, 0, 0).
JTextArea (Document doc, String text, int rows, int columns)
Constructs a new JTextArea with the specified number of rows and columns, and the given model.
JTextArea (int rows, int columns)
Constructs a new empty TextArea with the specified number of rows and columns.
JTextArea (String text)
Constructs a new TextArea with the specified text displayed.
JTextArea (String text, int rows, int columns)
Constructs a new TextArea with the specified text and number of rows and columns.

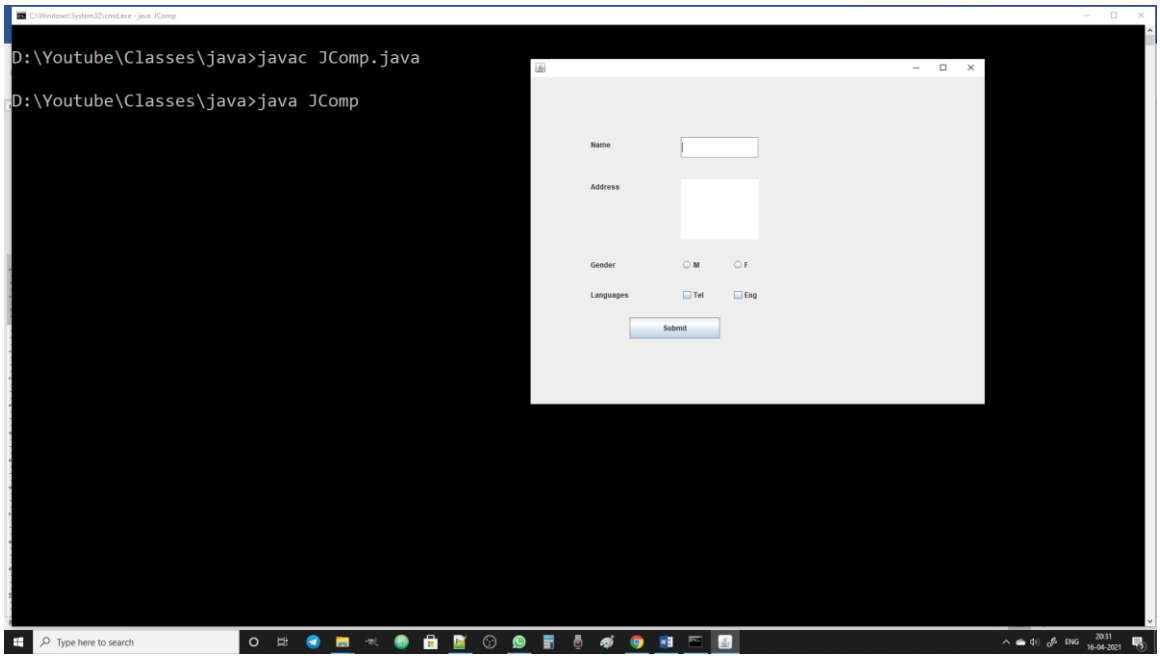
Example:

```
//program to demonstrate JCheckBox, JRadioButton, JTextField, JTextArea, JButton
import javax.swing.*;
import java.awt.*;

class JComp extends JFrame
{
	JComp()
	{
		Container c = this.getContentPane();
		c.setLayout(null);

		JLabel n = new JLabel("Name");
		n.setBounds(100,100,130,25);
		c.add(n);
```

```
        JTextField t = new JTextField();
        t.setBounds(250,100,130,35);
        c.add(t);

        JLabel a = new JLabel("Address");
        a.setBounds(100,170,130,25);
        c.add(a);

        JTextArea addr = new JTextArea();
        addr.setBounds(250,170,130,100);
        c.add(addr);

        JLabel g = new JLabel("Gender");
        g.setBounds(100,300,130,25);
        c.add(g);

        JRadioButton m = new JRadioButton("M");
        m.setBounds(250,300,75,25);
    c.add(m);

        JRadioButton f = new JRadioButton("F");
        f.setBounds(335,300,75,25);
    c.add(f);

        ButtonGroup bg = new ButtonGroup();
        bg.add(m);
        bg.add(f);

        JLabel ln = new JLabel("Languages");
        ln.setBounds(100,350,130,25);
        c.add(ln);

        JCheckBox c1 = new JCheckBox("Tel");
        c1.setBounds(250,350,75,25);
        c.add(c1);

        JCheckBox c2 = new JCheckBox("Eng");
        c2.setBounds(335,350,75,25);
        c.add(c2);

        JButton submit = new JButton("Submit");
        submit.setBounds(165,400,150,35);
        c.add(submit);



    }
    public static void main(String ar[])
    {
        JComp f = new JComp();
        f.setSize(500,400);
        f.setVisible(true);
```

```
                f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);


        }
}
```

Output



JTabbedPane:

A component that lets the user switch between a group of components by clicking on a tab with a given title and/or icon. Tabs/components are added to a TabbedPane object by using the addTab and insertTab methods. A tab is represented by an index corresponding to the position it was added in, where the first tab has an index equal to 0 and the last tab has an index equal to the tab count minus 1.

Constructor Summary:

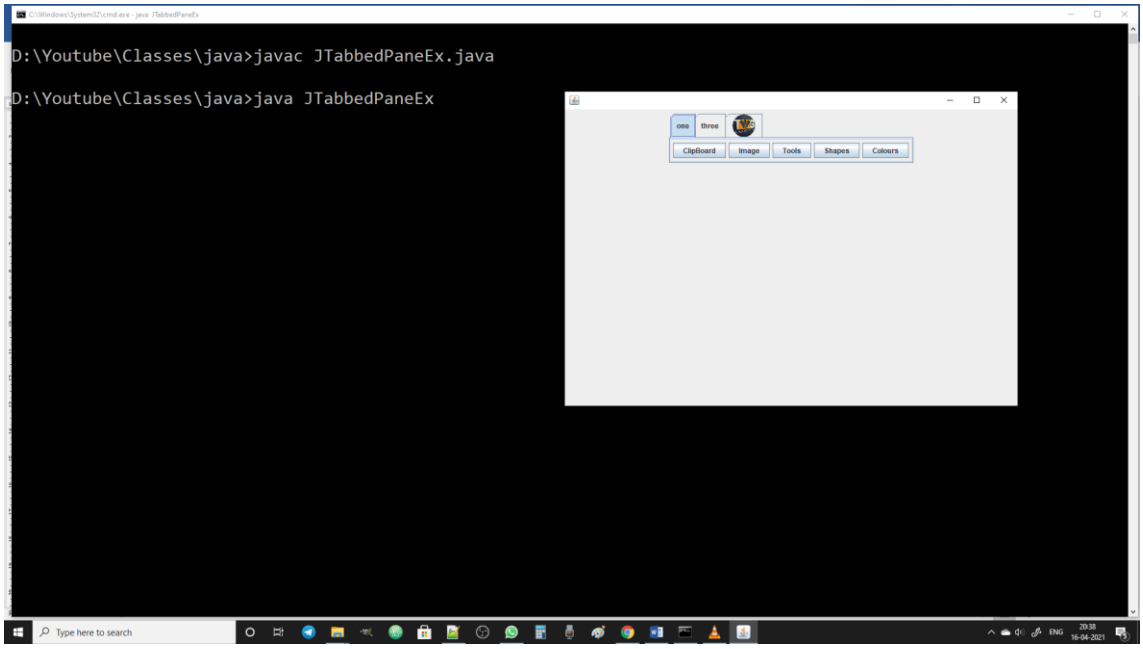
JTabbedPane ()
Creates an empty TabbedPane with a default tab placement of JTabbedPane.TOP.
JTabbedPane (int tabPlacement)
Creates an empty TabbedPane with the specified tab placement of either: JTabbedPane.TOP, JTabbedPane.BOTTOM, JTabbedPane.LEFT, or JTabbedPane.RIGHT.
JTabbedPane (int tabPlacement, int tabLayoutPolicy)
Creates an empty TabbedPane with the specified tab placement and tab layout policy.

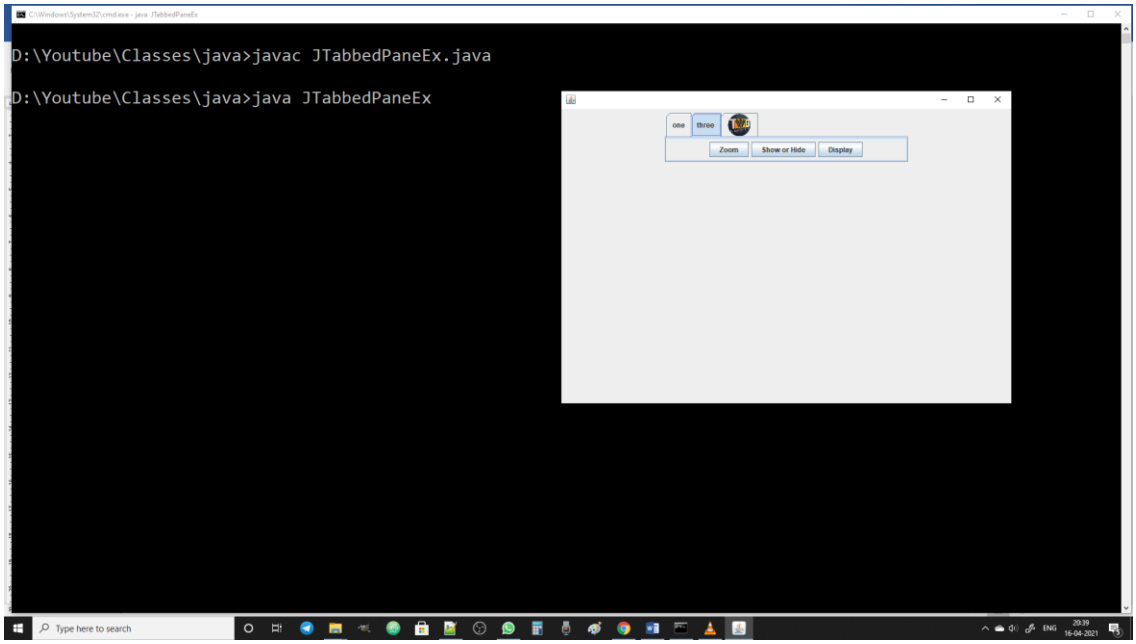
Example:

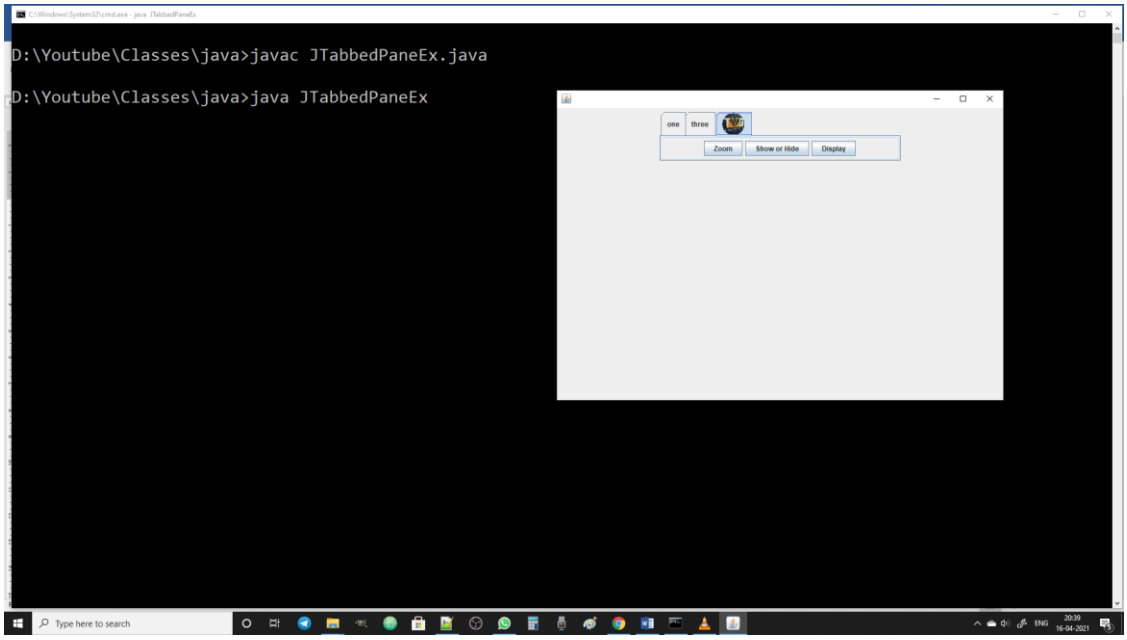
```
//program to demonstrate JTabbedPane
import javax.swing.*;
import java.awt.*;

class JTabbedPaneEx extends JFrame
{
	JTabbedPaneEx()
	{
		FlowLayout fl = new FlowLayout();
		this.setLayout(fl);

		JTabbedPane p = new JTabbedPane(JTabbedPane.TOP);

		p.add("one",new Home());

		ImageIcon img = new ImageIcon("small.png");

		p.addTab("",img,new View());


    p.insertTab("three", null, new View(), "This is view tab", 1);

		this.add(p);

	}
	public static void main(String ar[])
	{
		JTabbedPaneEx f = new JTabbedPaneEx();
		f.setSize(500,400);
		f.setVisible(true);

		f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

	}

}

class Home extends JPanel
{
	Home()
	{
		JButton b1 = new JButton("ClipBoard");
		JButton b2 = new JButton("Image");
		JButton b3 = new JButton("Tools");
		JButton b4 = new JButton("Shapes");
		JButton b5 = new JButton("Colours");

		this.add(b1);
		this.add(b2);
		this.add(b3);
		this.add(b4);
		this.add(b5);

	}
}
```

```
class View extends JPanel
{
	View()
	{
		JButton b1 = new JButton("Zoom");
		JButton b2 = new JButton("Show or Hide");
		JButton b3 = new JButton("Display");

		this.add(b1);
		this.add(b2);
		this.add(b3);

	}
}
```

Output:







JTable :

The JTable is used to display and edit regular two-dimensional tables of cells.

A JTable can be created by using the following constructors.

JTable() – It creates an empty table

JTable(int numRows, int numColumns) –
Constructs a JTable with numRows and numColumns of empty cells using DefaultTableModel.

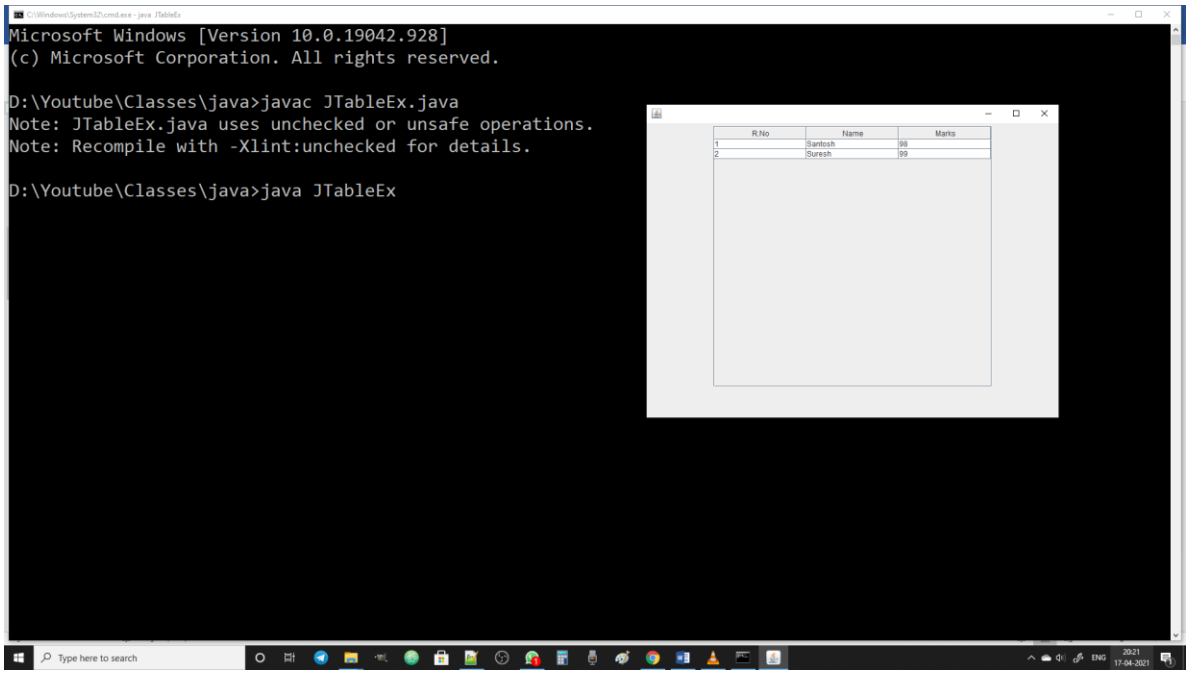
JTable(Object[][] rowData, Object[] columnNames)
Constructs a JTable to display the values in the two dimensional array, rowData, with column names, columnNames.

Example

```
//program to demonstrate JTable
import javax.swing.*;
import java.awt.*;
import java.util.*;

class JTableEx extends JFrame
{
	JTableEx()
	{
		FlowLayout fl = new FlowLayout();
		this.setLayout(fl);

		Container c = this.getContentPane();


		//String rows[][]={{"1","Santosh","98"},{"2","Suresh","99"}};
		//String cols[]={"R.No","Name","Marks"};

		Vector<Vector> rows=new Vector<Vector>();

		Vector rowone=new Vector();
		rowone.add("1");
		rowone.add("Santosh");
		rowone.add("98");

		Vector rowtwo=new Vector();
		rowtwo.add("2");
		rowtwo.add("Suresh");
		rowtwo.add("99");

		rows.add(rowone);
		rows.add(rowtwo);

		Vector cols=new Vector();
		cols.add("R.No");
		cols.add("Name");
		cols.add("Marks");



		JTable t = new JTable(rows,cols);

		JScrollPane jsp = new JScrollPane(t);

		c.add(jsp);
	}
	public static void main(String ar[])
	{
		JTableEx f =new JTableEx();
		f.setSize(500,400);
```

```
                f.setVisible(true);

                f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

        }

}
```

Output



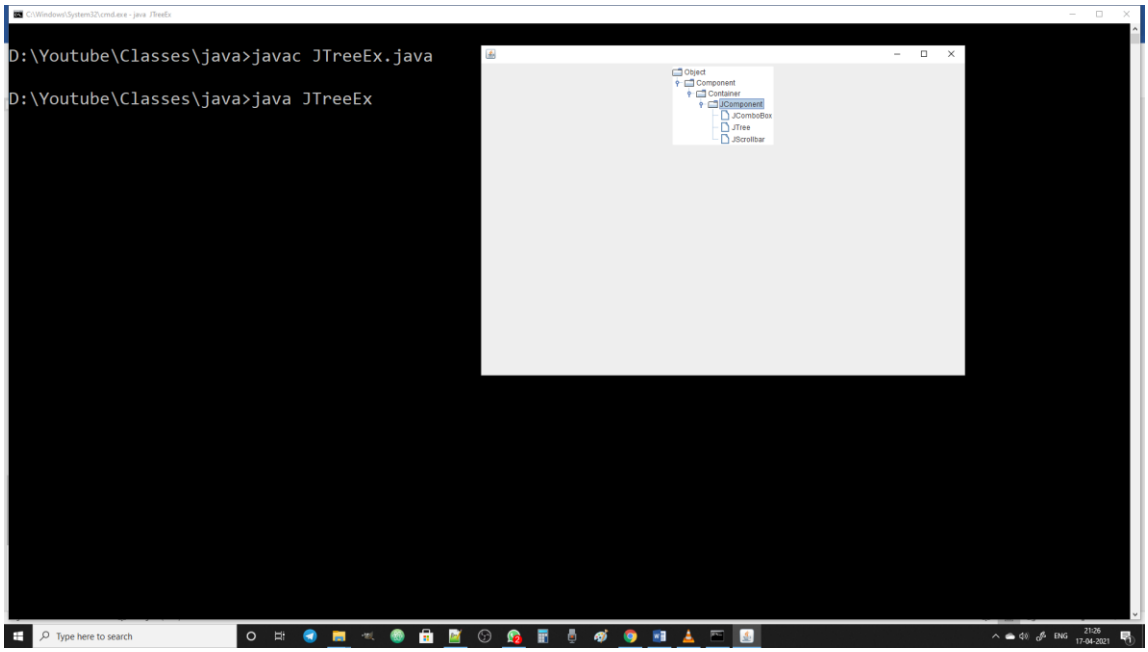
JTree:

A control that displays a set of hierarchical data as an outline. A Tree node can be created by using DefaultMutableTreeNode. Following example shows how to create a tree hierarchy.

Example:

```
//program to demonstrate JTree
import javax.swing.*;
import javax.swing.tree.*;
import java.awt.*;

class JTreeEx extends JFrame
{
	JTreeEx()
	{
		FlowLayout fl = new FlowLayout();
		this.setLayout(fl);

		Container c = this.getContentPane();

		DefaultMutableTreeNode root = new DefaultMutableTreeNode("Object");

		DefaultMutableTreeNode comp = new DefaultMutableTreeNode("Component");
		root.add(comp);

		DefaultMutableTreeNode cont = new DefaultMutableTreeNode("Container");
		comp.add(cont );

		DefaultMutableTreeNode jcomp = new DefaultMutableTreeNode("JComponent");
		cont.add(jcomp);

		DefaultMutableTreeNode jcomb = new DefaultMutableTreeNode("JComboBox");
		jcomp.add(jcomb);

		DefaultMutableTreeNode jtr = new DefaultMutableTreeNode("JTree");
		jcomp.add(jtr);

		DefaultMutableTreeNode jsr = new DefaultMutableTreeNode("JScrollbar");
		jcomp.add(jsr);

		JTree t =new JTree(root);
		c.add(t);

	}
	public static void main(String ar[])
	{
		JTreeEx f = new JTreeEx();
		f.setSize(500,400);
		f.setVisible(true);

		f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

	}

}
```

Output



JSplitPane:

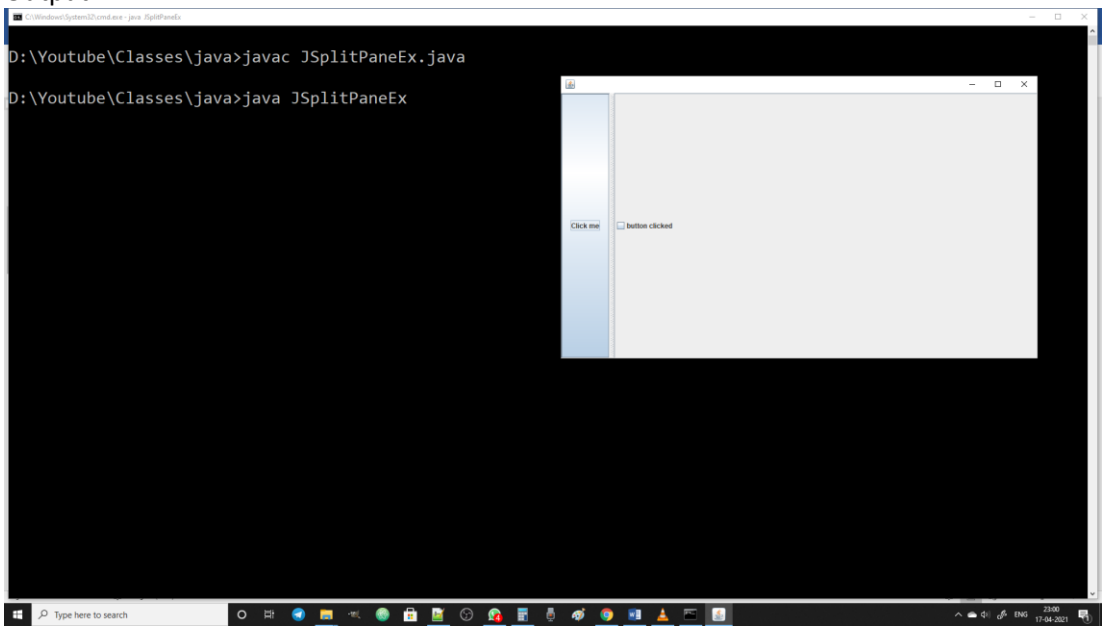
JSplitPane is used to divide two (and only two) Components. The two Components are graphically divided based on the look and feel implementation, and the two Components can then be interactively resized by the user.

Example

```
//program to demonstrate JSplitPane
import javax.swing.*;
import java.awt.*;

class JSplitPaneEx extends JFrame
{
	JSplitPaneEx(){

		Container c = this.getContentPane();

		//FlowLayout fl = new FlowLayout();
		//this.setLayout(fl);


		JButton l = new JButton("Click me");
		JCheckBox r = new JCheckBox("button clicked");

		JSplitPane js = new JSplitPane(JSplitPane.HORIZONTAL_SPLIT,l,r);
		c.add(js);

		}
```

```
	public static void main(String ar[])
	{
		JSplitPaneEx f = new JSplitPaneEx();
		f.setSize(500,400);
		f.setVisible(true);

		f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);



	}

}
```

Output:



JMenu:

An implementation of a menu -- a popup window containing JMenuItems that is displayed when the user selects an item on the JMenuBar. In addition to JMenuItems, a JMenu can also contain JSeparators.

In essence, a menu is a button with an associated JPopupMenu. When the "button" is pressed, the JPopupMenu appears. If the "button" is on the JMenuBar, the menu is a top-level window. If the "button" is another menu item, then the JPopupMenu is "pull-right" menu.

Constructors:

JMenu()
	Constructs a new JMenu with no text.
JMenu(Action a)
	Constructs a menu whose properties are taken from the Action supplied.

JMenu(String s)
Constructs a new JMenu with the supplied string as its text.
JMenu(String s, boolean b)
Constructs a new JMenu with the supplied string as its text and specified as a tear-off menu or not.

JCheckBox:

An implementation of a check box -- an item that can be selected or deselected, and which displays its state to the user. By convention, any number of check boxes in a group can be selected.

Constructors:

JCheckBox ()
Creates an initially unselected check box button with no text, no icon.
JCheckBox (Action a)
Creates a check box where properties are taken from the Action supplied.
JCheckBox (Icon icon)
Creates an initially unselected check box with an icon.
JCheckBox (Icon icon, boolean selected)
Creates a check box with an icon and specifies whether or not it is initially selected.
JCheckBox (String text)
Creates an initially unselected check box with text.
JCheckBox (String text, boolean selected)
Creates a check box with text and specifies whether or not it is initially selected.
JCheckBox (String text, Icon icon)
Creates an initially unselected check box with the specified text and icon.
JCheckBox (String text, Icon icon, boolean selected)
Creates a check box with text and icon, and specifies whether or not it is initially selected.

JRadioButton:

An implementation of a radio button -- an item that can be selected or deselected, and which displays its state to the user. Used with a ButtonGroup object to create a group of buttons in which only one button at a time can be selected.

Buttons can be configured, and to some degree controlled, by Actions. Using an Action with a button has many benefits beyond directly configuring a button.

Constructor Summary:

JRadioButton ()
Creates an initially unselected radio button with no set text.
JRadioButton (Action a)
Creates a radiobutton where properties are taken from the Action supplied.
JRadioButton (Icon icon)
Creates an initially unselected radio button with the specified image but no text.
JRadioButton (Icon icon, boolean selected)
Creates a radio button with the specified image and selection state, but no text.

JRadioButton (String text)
Creates an unselected radio button with the specified text.
JRadioButton (String text, boolean selected)
Creates a radio button with the specified text and selection state.
JRadioButton (String text, Icon icon)
Creates a radio button that has the specified text and image, and that is initially unselected.
JRadioButton (String text, Icon icon, boolean selected)
Creates a radio button that has the specified text, image, and selection state.

JTextField:

JTextField is a lightweight component that allows the editing of a single line of text. JTextField is intended to be source-compatible with java.awt.TextField where it is reasonable to do so. This component has capabilities not found in the java.awt.TextField class. The superclass should be consulted for additional capabilities.

Constructor Summary:

JTextField()
Constructs a new TextField.
JTextField(Document doc, String text, int columns)
Constructs a new JTextField that uses the given text storage model and the given number of columns.
JTextField(int columns)
Constructs a new empty TextField with the specified number of columns.
JTextField(String text)
Constructs a new TextField initialized with the specified text.
JTextField(String text, int columns)
Constructs a new TextField initialized with the specified text and columns.

JTextArea:

A JTextArea is a multi-line area that displays plain text. It is intended to be a lightweight component that provides source compatibility with the java.awt.TextArea class where it can reasonably do so. This component has capabilities not found in the java.awt.TextArea class. The superclass should be consulted for additional capabilities. Alternative multi-line text classes with more capabilities are JTextPane and JEditorPane.

The java.awt.TextArea internally handles scrolling. JTextArea is different in that it doesn't manage scrolling, but implements the swing Scrollable interface. This allows it to be placed inside a JScrollPane if scrolling behavior is desired, and used directly if scrolling is not desired.

Constructor Summary:

JTextArea()
Constructs a new TextArea.
JTextArea(Document doc)
Constructs a new JTextArea with the given document model, and defaults for all of the other arguments (null, 0, 0).
JTextArea(Document doc, String text, int rows, int columns)
Constructs a new JTextArea with the specified number of rows and columns, and the given model.
JTextArea(int rows, int columns)
Constructs a new empty TextArea with the specified number of rows and columns.

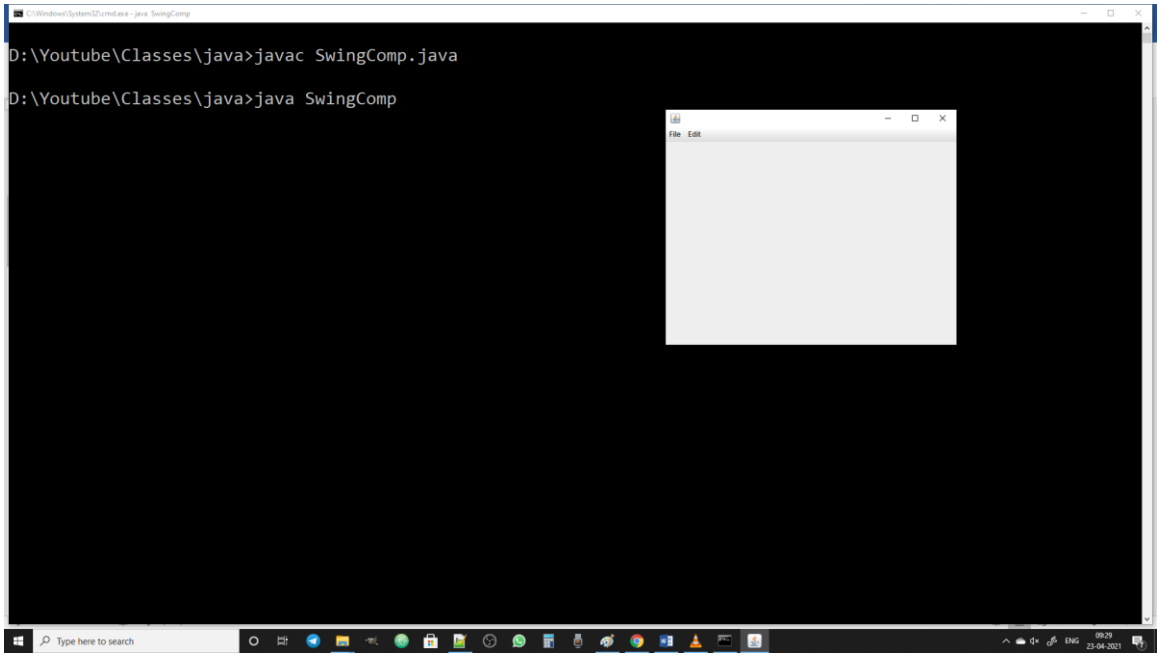
JTextArea(String text)
Constructs a new TextArea with the specified text displayed.
JTextArea(String text, int rows, int columns)
Constructs a new TextArea with the specified text and number of rows and columns.

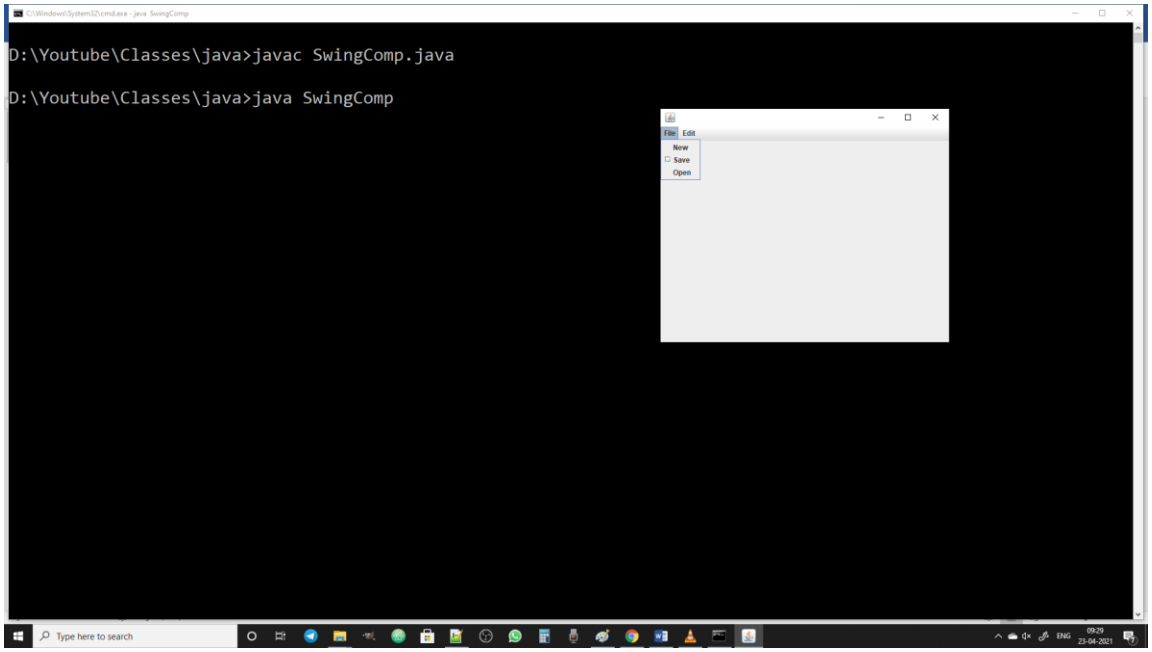
Example:

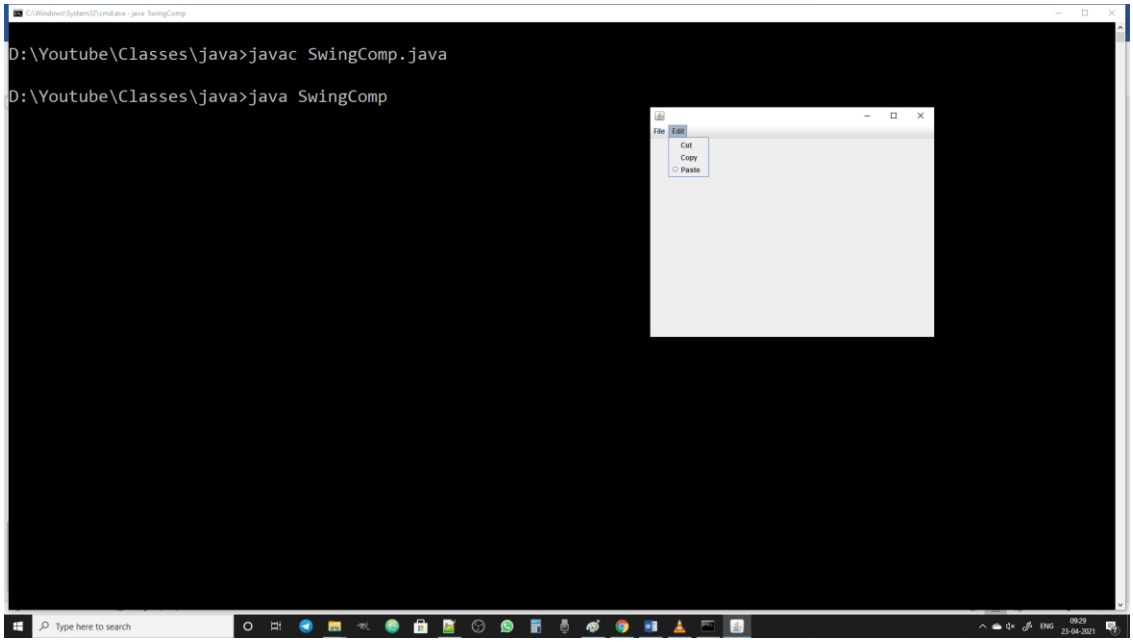
```
//program to demonstrate JMenu
import javax.swing.*;
import java.awt.*;

class SwingComp extends JFrame
{

   SwingComp()
        {
                Container c = this.getContentPane();

                BorderLayout bl = new BorderLayout();


                JMenuBar mb = new JMenuBar();

                JMenu fil = new JMenu("File");
                mb.add(fil);

                JMenu edt = new JMenu("Edit");
                mb.add(edt);

                JMenuItem n = new JMenuItem("New");
                JCheckBoxMenuItem s = new JCheckBoxMenuItem("Save");
                JMenuItem o = new JMenuItem("Open");

                fil.add(n);
                fil.add(s);
                fil.add(o);

                JMenuItem cu = new JMenuItem("Cut");
                JMenuItem co = new JMenuItem("Copy");
                JRadioButtonMenuItem ps = new JRadioButtonMenuItem("Paste");

                edt.add(cu);
                edt.add(co);
                edt.add(ps);

                c.add("North",mb);


        }

        public static void main(String ar[])
        {
          SwingComp f = new SwingComp();
          f.setSize(500,400);
                f.setVisible(true);
```

```
                f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        }
}
```

Output





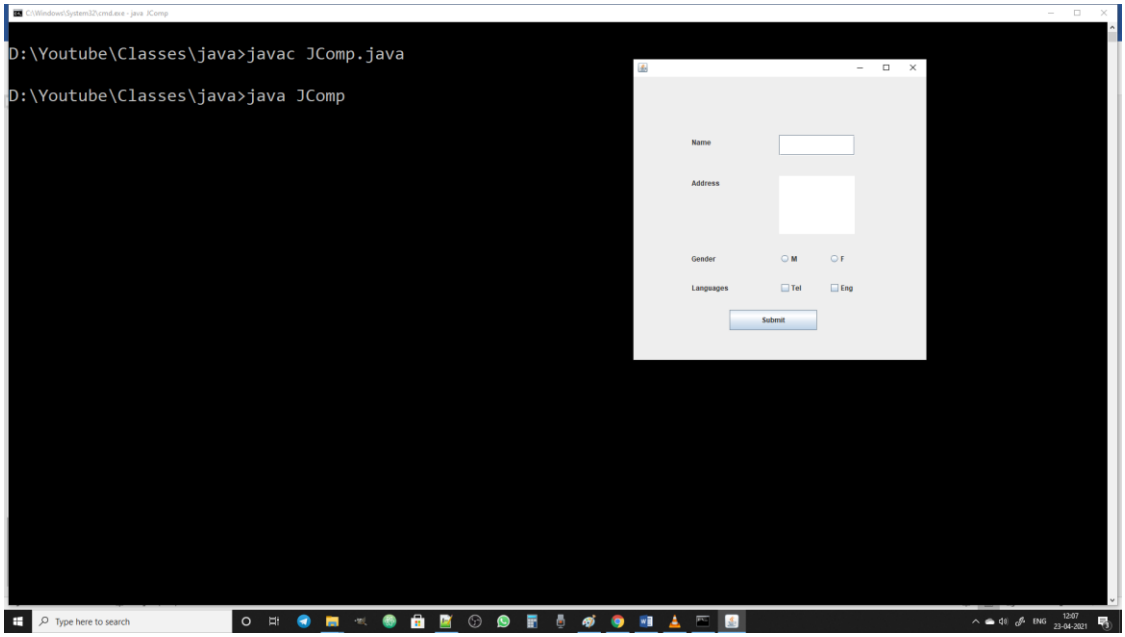


Example2

```
//program to demonstrate JCheckBox, JRadioButton, JTextField, JTextArea, JButton
import javax.swing.*;
import java.awt.*;

class JComp extends JFrame
{
	JComp()
	{
		Container c = this.getContentPane();
		c.setLayout(null);

		JLabel n = new JLabel("Name");
		n.setBounds(100,100,130,25);
		c.add(n);

		JTextField t = new JTextField();
		t.setBounds(250,100,130,35);
		c.add(t);

		JLabel a = new JLabel("Address");
		a.setBounds(100,170,130,25);
		c.add(a);

		JTextArea addr = new JTextArea();
		addr.setBounds(250,170,130,100);
		c.add(addr);

		JLabel g = new JLabel("Gender");
		g.setBounds(100,300,130,25);
		c.add(g);

		JRadioButton m = new JRadioButton("M");
```

```
                m.setBounds(250,300,75,25);
        c.add(m);

                JRadioButton f = new JRadioButton("F");
                f.setBounds(335,300,75,25);
        c.add(f);

                ButtonGroup bg = new ButtonGroup();
                bg.add(m);
                bg.add(f);

                JLabel ln = new JLabel("Languages");
                ln.setBounds(100,350,130,25);
                c.add(ln);

                JCheckBox c1 = new JCheckBox("Tel");
                c1.setBounds(250,350,75,25);
                c.add(c1);

                JCheckBox c2 = new JCheckBox("Eng");
                c2.setBounds(335,350,75,25);
                c.add(c2);

                JButton submit = new JButton("Submit");
                submit.setBounds(165,400,150,35);
                c.add(submit);



        }
        public static void main(String ar[])
        {
                JComp f = new JComp();
                f.setSize(500,400);
                f.setVisible(true);

                f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);


        }

}
```

Output


D:\Youtube\Classes\java>javac JComp.java
D:\Youtube\Classes\java>java JComp
Name
Address
Gender
M
F
Languages
Tel
Eng
Submit
Type here to search

APPLETS IN JAVA

An applet is a small program that is intended not to be run on its own, but rather to be embedded inside another application.

The Applet class must be the superclass of any applet that is to be embedded in a Web page or viewed by the Java Applet Viewer.

We can develop Web based applications using applets.

Applets are special programs that are included in HTML. So an applet is said to be "Java code embedded in a webpage".

The following are the steps to be followed to implement applets.

1) Create a java program
2) Compile the java program and get the .class file (byte code)
3) embedded byte code in html program.

Advantages/Uses of applets

1) We can create dynamic web pages by using Applets.
2) We can create animations and games.

Applet Life Cycle:

Every applet program will run through set of methods which is said to be it's life cycle. The following are the methods that are part of every applet life cycle.

init() – It contains all the initializations. It will be executed only once.
start() - It executes when applets execution started or applet is enabled from disabled state.
stop() - It will be invoked when applet should stop its execution.
destroy() - applet that it is being reclaimed and that it should destroy any resources that it has allocated.

All applets related classes and interfaces are under java.applet package.

How to create an Applet Class?

Applets are created by using the following steps.

1) import the java.applet package

```
import java.applet.*
```

2) Create a class that extends Applet

```
public class MyApp extends Applet
{
    //implement the life cycle methods
}
```

3) write paint method to print the output in the output window

```
public class MyApp extends Applet
{
    //implement the life cycle methods

     public void paint(Graphics g)
     {
         //print the output using Graphics methods like drwaString / drwaLine / drawRect….
     }
}
```

Example

```
//program to demonstrate applets
import java.applet.*;
import java.awt.*;


public class MyApp extends Applet
{
   String msg = "";
   public void init()
          {
                    setBackground(Color.blue);
                    setForeground(Color.white);
                    msg+="init--->";

                    this.add(new Label("Telugu Web Guru"));
                    this.add(new Button("Click me"));

          }
          public void start()
          {
                    msg+="start--->";
          }
          public void stop()
          {
                   msg+="stop--->";
          }
          public void destroy()
          {
                    msg+="destroy--->";

          }

   public void paint(Graphics g)
   {
                       g.drawLine(30,60,500,250);
        g.drawString(msg,100,200);
   }

}
```

Once above java program is compiled we can embed this into our html file using applet tag.

To include applet class in html we generally use the following tag.

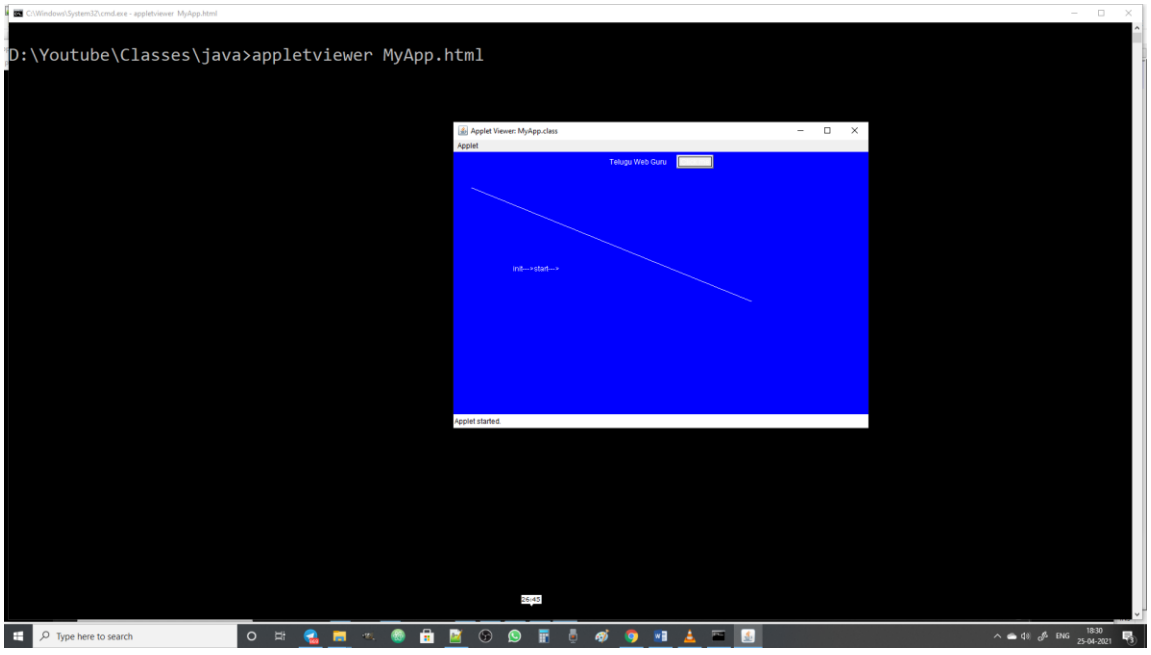
```
<applet
        code = "applet classname"
        codebase= "path"
        width= "width of applet window"
        height= "height of applet window"
        align= "alignment "
        alt= "alternate text to be displayed in case of any errors"

 >

</applet>
```

Example

```
<html>
<applet code="MyApp.class" width="500" height="400">

</applet>
</html>
```

Modern browsers won't support applets because applets are client side programming.

So we can execute applets by using appletviewer also



MISCELLANEOUS TOPICS IN JAVA

Garbage Collection :

Java performs garbage collection automatically. Garbage collection means free up the memory by deallocating unused variables and other resources from the memory.

For this java follows reference count. Whenever an object's reference count reaches to 0 then it will be deallocated from memory as it is dead from that point.

We can invoke the garbage collector manually also by using the following resources.