

J2SE

(Core Java)

Quick Reference



A.R. KISHORE KUMAR

J2SE (Core Java) Quick Reference covers:

- Java Buzz Words
- Data Types
- Control Structures
- Arrays
- Strings
- OOPS in Java
- Static Methods
- Inner Class
- Abstract Class
- Interface
- Packages
- Exceptions
- Generic Types
- lang Package
- util Package
- io Package
- net Package
- Threads
- Abstract Window Toolkit
- Swings
- Applets

Contents

1. Introduction to Java	1
• History of Java	1
• Features of Java (Java Buzz words)	1
• Obtaining Java Environment	2
• Setting up Java Environment	2
2. Programming Structure	3
• Comments	3
• Structure of the Java Program	3
• Escape Sequence	4
• Creating a Source File	4
• Compiling the Source File into a .class file	5
• Executing the Program	6
• The Java Virtual Machine	6
3. Naming Conventions, Data Types and Operators	8
• Naming Conventions	8
• Data Types	8
• Operators	9
4. Control Structures	12
• Java's Selection Statements	12
• Java's Iteration Statements	13
• Java's Jump Statements	15
5. Accepting Input from Keyboard	18
• Accepting a Single Character from the Keyboard	18
• Accepting a String from Keyboard	19
• Accepting an Integer value from Keyboard	19
• Accepting a Float value from Keyboard	19
• Accepting a Double value from Keyboard	19
• Accepting Other Types of Values	19
6. Arrays, Strings & StringBuffer	21
• Arrays	21
• Single Dimensional Arrays	21
• Multi-Dimensional Arrays (2D, 3D, ... arrays)	22
• Creating Strings	23
• Strings	23
• StringBuffer	25
7. Introduction to OOPs	27
• Difference between Procedure Oriented Programming and OOP	27

• Features of OOP	27
• Initializing Instance Variables	31
• Constructor	33
• The keyword 'this'	35
• Garbage Collection	36
• The finalize () method	36
8. Methods & Inner Class	37
• Methods	37
• Instance Methods	37
• Static Methods	38
• Inner Class	39
9. Inheritance	41
• Inheritance	41
• The keyword super	43
10. Polymorphism	45
• Dynamic Polymorphism	45
○ Method Overloading	45
○ Method Overriding	46
• Static Polymorphism	47
• The keyword final	47
• Type Casting	47
11. Abstract Class	50
12. Interface	52
• Types of Inheritance	53
13. Packages	55
• Creating Sub Package	57
• Access Specifier	57
14. Exceptions	60
• throws clause	62
• throw clause	62
• Types of Exceptions	63
15. Wrapper Classes	65
• Character Class	65
• Byte Class	66
• Short Class	67
• Integer Class	67
• Float Class	68
• Long Class	68

• Boolean Class	69
• Double Class	69
• Math Class	70
16. Generic Types	71
• Generic Class	71
• Generic Method	72
• Generic Interface	73
17. The Collection Framework	75
• Collection Object	75
• Retrieving elements from Collection	75
• HashSet Class	76
• LinkedHashSet Class	78
• Stack Class	78
• LinkedList Class	79
• ArrayList Class	80
• Vector Class	82
• HashMap Class	83
• Hashtable Class	84
• Arrays Class	85
• StringTokenizer	86
• Calendar	87
• Date	88
18. Streams and Files	90
• Byte Streams	90
• Character or Text Streams	90
• Serialization of Objects	95
• File Class	97
19. Networking in Java	99
• Socket Programming	99
20. Threads	103
• Creating Thread	104
• MultiTasking using Threads	104
• Multiple Threads acting on Single Object	106
• Thread Synchronization or Thread Safe	107
• Thread Class Methods	108
• Deadlock	108
• Thread Communication	110
• ThreadGroup	112
• Thread States (Life-Cycle of a Thread)	114

21. Abstract Window Toolkit	115
• Creating the Frame	115
• Event Delegation Model	117
• Closing the Frame	117
• Displaying text in the Frame	119
• Drawing in the Frame	120
• Displaying images in the Frame	122
• Component Class Methods	123
• Listeners and Listener Methods	125
• Creating Push Buttons	125
• Checkbox	127
• Radio Button	128
• Choice Menu	130
• List box	131
• Label, TextField, TextArea	133
• Scrollbar Class	134
• Handling Mouse Events	136
• Handling Keyboard Events	138
22. Swings	141
• Creating a Frame	141
• WindowPane	142
• Displaying Text in the Frame	143
• JComponent class Methods	144
• PushButton	145
• Label	146
• Creating Components in Swing	147
• JComboBox Class	149
• JList Class	150
• JTabbedPane	152
• JTable	153
• JMenu Class	155
• Layout Manager	157
○ Flow Layout	158
○ Border Layout	159
○ Card Layout	160
○ Grid Layout	161
○ GridBag Layout	162
23. Applets	165
• Creating an applet	165



Java – James Gosling

James Gosling is a famous Canadian software developer who has been with Sun Microsystems since 1984 and is considered as father of Java programming language. Gosling did the original design of Java and implemented its original compiler and virtual machine.

1. Introduction to Java

History of Java:

- In 1990, Sun Micro Systems Inc. (US) was conceived a project to develop software for consumer electronic devices that could be controlled by a remote. This project was called Stealth Project but later its name was changed to Green Project.
- In January 1991, Project Manager James Gosling and his team members Patrick Naughton, Mike Sheridan, Chris Wrath, and Ed Frank met to discuss about this project.
- Gosling thought C and C++ would be used to develop the project. But the problem he faced with them is that they were system dependent languages. The trouble with C and C++ (and most other languages) is that they are designed to be compiled for a specific target and could not be used on various processors, which the electronic devices might use.
- James Gosling with his team started developing a new language, which was completely system independent. This language was initially called **OAK**. Since this name was registered by some other company, later it was changed to **Java**.
- James Gosling and his team members were consuming a lot of coffee while developing this language. Good quality of coffee was supplied from a place called "Java Island". Hence they fixed the name of the language as Java. The symbol for Java language is cup and saucer.
- Sun formally announced Java at Sun World conference in 1995. On January 23rd 1996, JDK1.0 version was released.

Features of Java (Java buzz words):

- **Simple:** Learning and practicing java is easy because of resemblance with c and C++.
- **Object Oriented Programming Language:** Unlike C++, Java is purely OOP.
- **Distributed:** Java is designed for use on network; it has an extensive library which works in agreement with TCP/IP.
- **Secure:** Java is designed for use on Internet. Java enables the construction of virus-free, tamper free systems.
- **Robust (Strong/ Powerful):** Java programs will not crash because of its exception handling and its memory management features.
- **Interpreted:** Java programs are compiled to generate the byte code. This byte code can be downloaded and interpreted by the interpreter. .class file will have byte code instructions and JVM which contains an interpreter will execute the byte code.
- **Portable:** Java does not have implementation dependent aspects and it yields or gives same result on any machine.
- **Architectural Neutral Language:** Java byte code is not machine dependent, it can run on any machine with any processor and with any OS.
- **High Performance:** Along with interpreter there will be JIT (Just In Time) compiler which enhances the speed of execution.
- **Multithreaded:** Executing different parts of program simultaneously is called multithreading. This is an essential feature to design server side programs.
- **Dynamic:** We can develop programs in Java which dynamically change on Internet (e.g.: Applets).

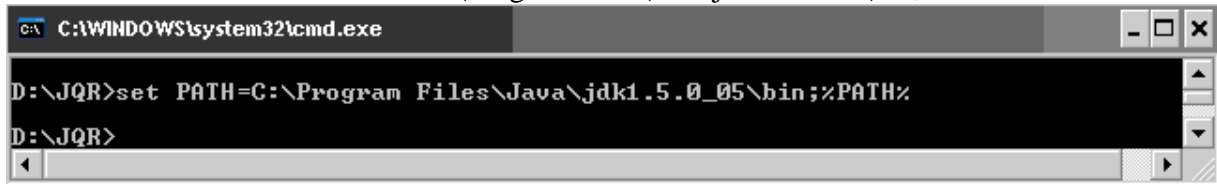
Obtaining the Java Environment:

- We can download the JDK (Java Development Kit) including the compiler and runtime engine from Sun at: <http://java.sun.com/javase>.
- Install JDK after downloading, by default JDK will be installed in
C:\Program Files\Java\jdk1.5.0_05 (Here jdk1.5.0_05 is JDK's version)

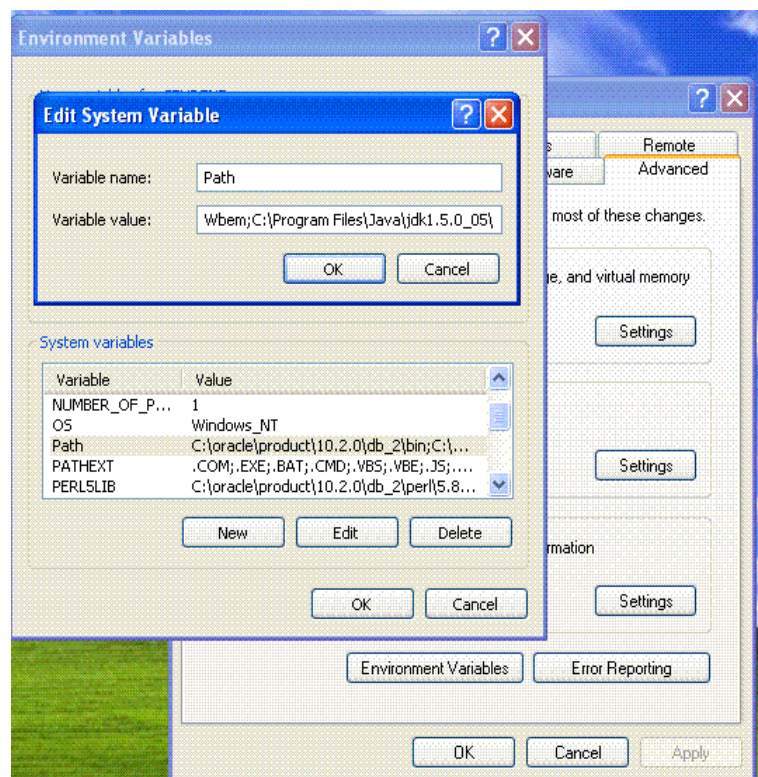
Setting up Java Environment: After installing the JDK, we need to set at least one environment variable in order to be able to compile and run Java programs. A PATH environment variable enables the operating system to find the JDK executables when our working directory is not the JDK's binary directory.

- **Setting environment variables from a command prompt:** If we set the variables from a command prompt, they will only hold for that session. To set the PATH from a command prompt:

```
set PATH=C:\Program Files\Java\jdk1.5.0_05\bin;%PATH%
```



- **Setting environment variables as system variables:** If we set the variables as system variables they will hold continuously.
 - Right-click on *My Computer*
 - Choose *Properties*
 - Select the *Advanced* tab
 - Click the *Environment Variables* button at the bottom
 - In system variables tab, select path (system variable) and click on edit button
 - A window with variable name-path and its value will be displayed.
 - Don't disturb the default path value that is appearing and just append (add) to that path at the end:
;C:\ProgramFiles\Java\jdk1.5.0_05\bin;
 - Finally press OK button.



2. Programming Structure

Comments: Comments are description about the aim and features of the program. Comments increase readability of a program. Three types of comments are there in Java:

- **Single line comments:** These comments start with `//`
e.g.: `// this is comment line`
- **Multi line comments:** These comments start with `/*` and end with `*/`
e.g.: `/* this is comment line*/`
- **Java documentation comments:** These comments start with `/**` and end with `*/`
These comments are useful to create a HTML file called API (application programming Interface) document. This file contains description of all the features of software.

Structure of the Java Program:

- As all other programming languages, Java also has a structure.
- The first line of the C/C++ program contains include statement. For example, `<stdio.h>` is the header file that contains functions, like `printf ()`, `scanf ()` etc. So if we want to use any of these functions, we should include this header file in C/ C++ program.
- Similarly in Java first we need to import the required packages. By default `java.lang.*` is imported. Java has several such packages in its library. A package is a kind of directory that contains a group of related classes and interfaces. A class or interface contains methods.
- Since Java is purely an Object Oriented Programming language, we cannot write a Java program without having at least one class or object. So, it is mandatory to write a class in Java program. We should use `class` keyword for this purpose and then write class name.
- In C/C++, program starts executing from main method similarly in Java, program starts executing from main method. The return type of main method is void because program starts executing from main method and it returns nothing.

Sample Program:

```
//A Simple Java Program
import java.lang.System;
import java.lang.String;
class Sample
{
    public static void main(String args[])
    {
        System.out.print ("Hello world");
    }
}
```

- Since Java is purely an Object Oriented Programming language, without creating an object to a class it is not possible to access methods and members of a class. But main method is also a method inside a class, since program execution starts from main method we need to call main method without creating an object.
- Static methods are the methods, which can be called and executed without creating objects. Since we want to call `main ()` method without using an object, we should declare `main ()`

method as static. JVM calls main () method using its Classname.main () at the time of running the program.

- JVM is a program written by Java Soft people (Java development team) and main () is the method written by us. Since, main () method should be available to the JVM, it should be declared as public. If we don't declare main () method as public, then it doesn't make itself available to JVM and JVM cannot execute it.
- JVM always looks for main () method with String type array as parameter otherwise JVM cannot recognize the main () method, so we must provide String type array as parameter to main () method.
- A class code starts with a { and ends with a }. A class or an object contains variables and methods (functions). We can create any number of variables and methods inside the class. This is our first program, so we had written only one method called main ().
- Our aim of writing this program is just to display a string "Hello world". In Java, print () method is used to display something on the monitor.
- A method should be called by using objectname.methodname (). So, to call print () method, create an object to PrintStream class then call objectname.print () method.
- An alternative is given to create an object to PrintStream Class i.e. System.out. Here, System is the class name and out is a static variable in System class. out is called a field in System class. When we call this field a PrintStream class object will be created internally. So, we can call print() method as: `System.out.print ("Hello world");`
- println () is also a method belonging to PrintStream class. It throws the cursor to the next line after displaying the result.
- In the above Sample program System and String are the classes present in java.lang package.

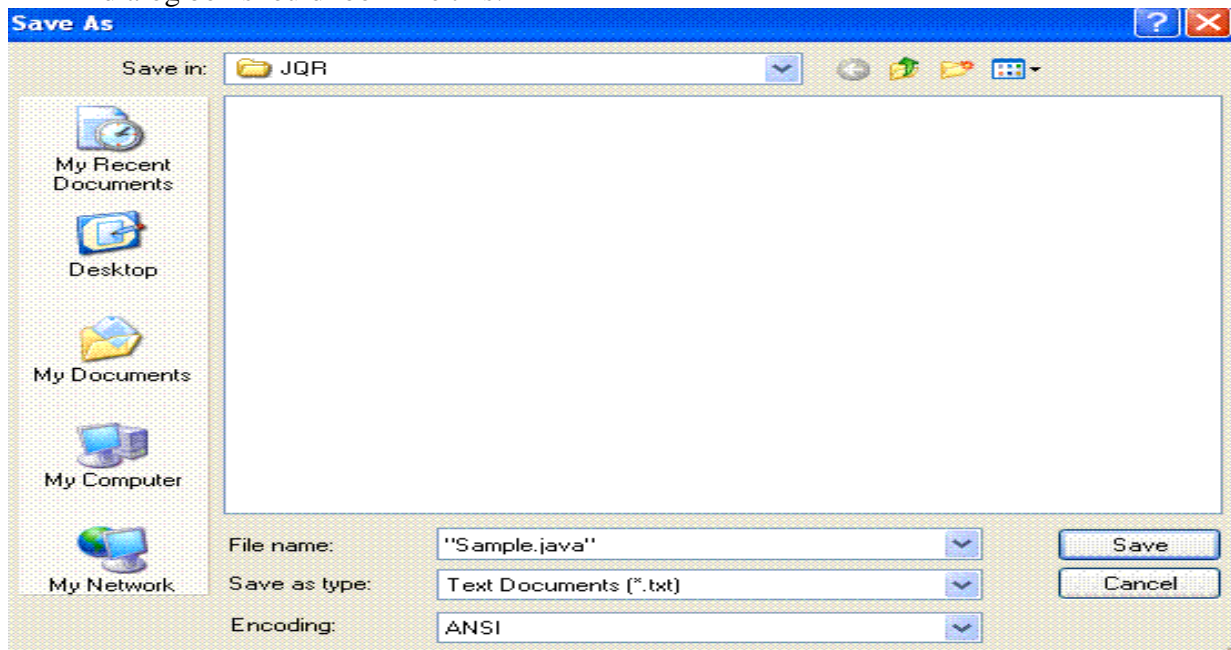
Escape Sequence: Java supports all escape sequence which is supported by C/ C++. A character preceded by a backslash (\) is an escape sequence and has special meaning to the compiler. When an escape sequence is encountered in a print statement, the compiler interprets it accordingly.

Escape Sequence	Description
\t	Insert a tab in the text at this point.
\b	Insert a backspace in the text at this point.
\n	Insert a newline in the text at this point.
\r	Insert a carriage return in the text at this point.
\f	Insert a form feed in the text at this point.
\'	Insert a single quote character in the text at this point.
\"	Insert a double quote character in the text at this point.
\\	Insert a backslash character in the text at this point.

Creating a Source File:

- Type the program in a text editor (i.e. Notepad, WordPad, Microsoft Word or Edit Plus). We can launch the Notepad editor from the **Start** menu by selecting **Programs > Accessories > Notepad**. In a new document, type the above code (i.e. Sample Program).
- Save the program with filename same as Class_name (i.e. Sample.java) in which main method is written. To do this in Notepad, first choose the **File > Save** menu item. Then, in the **Save** dialog box:

- Using the **Save in** combo box, specify the folder (directory) where you'll save your file. In this example, the directory is JQR on the D drive.
- In the **File name** text field, type "Sample.java", including the quotation marks. Then the dialog box should look like this:



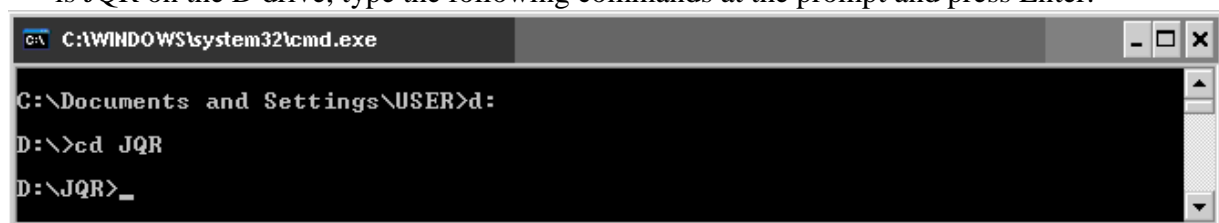
- Now click **Save**, and exit Notepad.

Compiling the Source File into a .class File:

- To Compile the Sample.java program go to DOS prompt. We can do this from the **Start** menu by choosing **Run...** and then entering cmd. The window should look similar to the following figure.



- The prompt shows current directory. To compile Sample.java source file, change current directory to the directory where Sample.java file is located. For example, if source directory is JQR on the D drive, type the following commands at the prompt and press Enter:



Now the prompt should change to D:\JQR>

- At the prompt, type the following command and press Enter.
javac Sample.java

```

C:\WINDOWS\system32\cmd.exe
D:\JQR>javac Sample.java
D:\JQR>

```

- The compiler generates byte code and Sample.class will be created.

Executing the Program (Sample.class):

- To run the program, enter java followed by the class name created at the time of compilation at the command prompt in the same directory as:
java Sample

```

C:\WINDOWS\system32\cmd.exe
D:\JQR>java Sample
Hello world
D:\JQR>

```

- The program interpreted and the output is displayed.

The Java Virtual Machine: Java Virtual Machine (JVM) is the heart of entire Java program execution process. First of all, the .java program is converted into a .class file consisting of byte code instructions by the java compiler at the time of compilation. Remember, this java compiler is outside the JVM. This .class file is given to the JVM. Following figure shows the architecture of Java Virtual Machine.

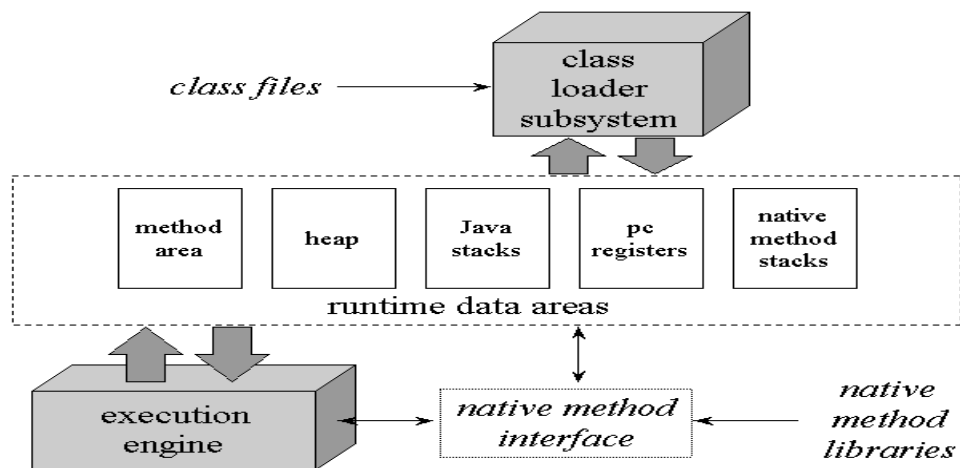


Figure: The internal architecture of the Java virtual machine.

In JVM, there is a module (or program) called class loader sub system, which performs the following instructions:

- First of all, it loads the .class file into memory.
- Then it verifies whether all byte code instructions are proper or not. If it finds any instruction suspicious, the execution is rejected immediately.

- If the byte instructions are proper, then it allocates necessary memory to execute the program. This memory is divided into 5 parts, called run time data areas, which contain the data and results while running the program. These areas are as follows:
 - **Method area:** Method area is the memory block, which stores the class code, code of the variables and code of the methods in the Java program. (Method means functions written in a class).
 - **Heap:** This is the area where objects are created. Whenever JVM loads a class, method and heap areas are immediately created in it.
 - **Java Stacks:** Method code is stored on Method area. But while running a method, it needs some more memory to store the data and results. This memory is allotted on Java Stacks. So, Java Stacks are memory area where Java methods are executed. While executing methods, a separate frame will be created in the Java Stack, where the method is executed. JVM uses a separate thread (or process) to execute each method.
 - **PC (Program Counter) registers:** These are the registers (memory areas), which contain memory address of the instructions of the methods. If there are 3 methods, 3 PC registers will be used to track the instruction of the methods.
 - **Native Method Stacks:** Java methods are executed on Java Stacks. Similarly, native methods (for example C/C++ functions) are executed on Native method stacks. To execute the native methods, generally native method libraries (for example C/C++ header files) are required. These header files are located and connected to JVM by a program, called Native method interface.

Execution Engine contains interpreter and JIT compiler which translates the byte code instructions into machine language which are executed by the microprocessor. Hot spot (loops/iterations) is the area in .class file i.e. executed by JIT compiler. JVM will identify the Hot spots in the .class files and it will give it to JIT compiler where the normal instructions and statements of Java program are executed by the Java interpreter.

3. Naming Conventions, Data Types and Operators

Naming Conventions: Naming conventions specify the rules to be followed by a Java programmer while writing the names of packages, classes, methods etc.

- Package names are written in small letters.
e.g.: java.io, java.lang, java.awt etc
- Each word of class name and interface name starts with a capital
e.g.: Sample, AddTwoNumbers
- Method names start with small letters then each word start with a capital
e.g.: sum (), sumTwoNumbers (), minValue ()
- Variable names also follow the same above method rule
e.g.: sum, count, totalCount
- Constants should be written using all capital letters
e.g.: PI, COUNT
- Keywords are reserved words and are written in small letters.
e.g.: int, short, float, public, void

Data Types: The classification of data item is called data type. Java defines eight simple types of data. byte, short, int, long, char, float, double and boolean. These can be put in four groups:

- **Integer Data Types:** These data types store integer numbers

Data Type	Memory size	Range
Byte	1 byte	-128 to 127
Short	2 bytes	-32768 to 32767
Int	4 bytes	-2147483648 to 2147483647
Long	8 bytes	-9223372036854775808 to 9223372036854775807

e.g.: byte rno = 10;

long x = 150L; L means forcing JVM to allot 8 bytes

- **Float Data Types:** These data types handle floating point numbers

Data Type	Memory size	Range
Float	4 bytes	-3.4e38 to 3.4e38
Double	8 bytes	-1.7e308 to 1.7e308

e.g.: float pi = 3.142f;

double distance = 1.98e8;

- **Character Data Type:** This data type represents a single character. char data type in java uses two bytes of memory also called Unicode system. Unicode is a specification to include alphabets of all international languages into the character set of java.

Data Type	Memory size	Range
Char	2 bytes	0 to 65535

e.g.: char ch = 'x';

- **Boolean Data Type:** can handle truth values either true or false

e.g.: boolean response = true;

Operators: An operator is a symbol that performs an operation. An operator acts on variables called operands.

- **Arithmetic operators:** These operators are used to perform fundamental operations like addition, subtraction, multiplication etc.

Operator	Meaning	Example	Result
+	Addition	3 + 4	7
-	Subtraction	5 - 7	-2
*	Multiplication	5 * 5	25
/	Division (gives quotient)	14 / 7	2
%	Modulus (gives remainder)	20 % 7	6

- **Assignment operator:** This operator (=) is used to store some value into a variable.

Simple Assignment	Compound Assignment
x = x + y	x += y
x = x - y	x -= y
x = x * y	x *= y
x = x / y	x /= y

- **Unary operators:** As the name indicates unary operator's act only on one operand.

Operator	Meaning	Example	Explanation
-	Unary minus	j = -k;	k value is negated and stored into j
++	Increment Operator	b++; ++b;	b value will be incremented by 1 (called as post incrementation) b value will be incremented by 1 (called as pre incrementation)
--	Decrement Operator	b--; --b;	b value will be decremented by 1 (called as post decrementation) b value will be decremented by 1 (called as pre decrementation)

- **Relational operators:** These operators are used for comparison purpose.

Operator	Meaning	Example
==	Equal	x == 3
!=	Not equal	x != 3
<	Less than	x < 3
>	Greater than	x > 3
<=	Less than or equal to	x <= 3

- **Logical operators:** Logical operators are used to construct compound conditions. A compound condition is a combination of several simple conditions.

Operator	Meaning	Example	Explanation
&&	and operator	if(a>b && a>c) System.out.print("yes");	If a value is greater than b and c then only yes is displayed
	or operator	if(a==1 b==1) System.out.print("yes");	If either a value is 1 or b value is 1 then yes is displayed
!	not operator	if(!(a==0)) System.out.print("yes");	If a value is not equal to zero then only yes is displayed

- **Bitwise operators:** These operators act on individual bits (0 and 1) of the operands. They act only on integer data types, i.e. byte, short, long and int.

Operator	Meaning	Explanation
&	Bitwise AND	Multiplies the individual bits of operands
	Bitwise OR	Adds the individual bits of operands
^	Bitwise XOR	Performs Exclusive OR operation
<<	Left shift	Shifts the bits of the number towards left a specified number of positions
>>	Right shift	Shifts the bits of the number towards right a specified number of positions and also preserves the sign bit.
>>>	Zero fill right shift	Shifts the bits of the number towards right a specified number of positions and it stores 0 (Zero) in the sign bit.
~	Bitwise complement	Gives the complement form of a given number by changing 0's as 1's and vice versa.

- **Ternary Operator or Conditional Operator (? :):** This operator is called ternary because it acts on 3 variables. The syntax for this operator is:

Variable = Expression1? Expression2: Expression3;

First Expression1 is evaluated. If it is true, then Expression2 value is stored into variable otherwise Expression3 value is stored into the variable.

e.g.: max = (a>b) ? a: b;

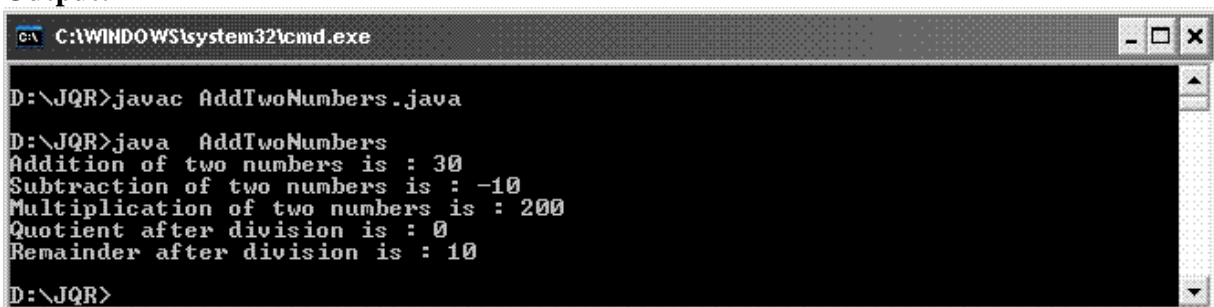
Program 1: Write a program to perform arithmetic operations

//Addition of two numbers

class AddTwoNumbers

```
{
    public static void main(String args[])
    {
        int i=10, j=20;
        System.out.println("Addition of two numbers is : " + (i+j));
        System.out.println("Subtraction of two numbers is : " + (i-j));
        System.out.println("Multiplication of two numbers is : " + (i*j));
        System.out.println("Quotient after division is : " + (i/j) );
        System.out.println("Remainder after division is : " +(i%j) );
    }
}
```

Output:



```
C:\WINDOWS\system32\cmd.exe
D:\JQR>javac AddTwoNumbers.java
D:\JQR>java AddTwoNumbers
Addition of two numbers is : 30
Subtraction of two numbers is : -10
Multiplication of two numbers is : 200
Quotient after division is : 0
Remainder after division is : 10
D:\JQR>
```

Program 2: Write a program to perform Bitwise operations

//Bitwise Operations

class Bits

```
{    public static void main(String args[])
    {    byte x,y;
        x=10;
        y=11;
        System.out.println ("~x="+(~x));
        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<<2="+(x<<2));
        System.out.println ("x>>2="+(x>>2));
        System.out.println ("x>>>2="+(x>>>2));
    }
}
```

Output:

```
C:\WINDOWS\system32\cmd.exe
D:\JQR>javac Bits.java
D:\JQR>java Bits
~x=-11
x & y=10
x | y=11
x ^ y=1
x<<2=40
x>>2=2
x>>>2=2
D:\JQR>_
```

4. Control Statements

Control statements are the statements which alter the flow of execution and provide better control to the programmer on the flow of execution. In Java control statements are categorized into selection control statements, iteration control statements and jump control statements.

- **Java's Selection Statements:** Java supports two selection statements: if and switch. These statements allow us to control the flow of program execution based on condition.

- **if Statement:** if statement performs a task depending on whether a condition is true or false.

Syntax:

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

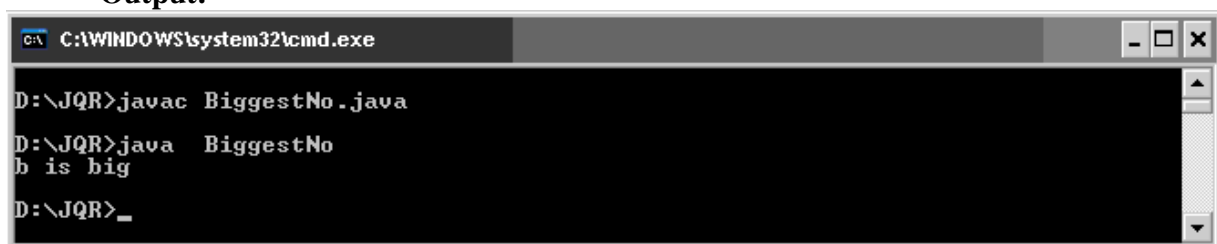
Here, each statement may be a single statement or a compound statement enclosed in curly braces (that is, a block). The condition is any expression that returns a boolean value. The else clause is optional.

Program 1: Write a program to find biggest of three numbers.

//Biggest of three numbers

```
class BiggestNo
{
    public static void main(String args[])
    {
        int a=5,b=7,c=6;
        if ( a > b && a>c)
            System.out.println ("a is big");
        else if ( b > c)
            System.out.println ("b is big");
        else
            System.out.println ("c is big");
    }
}
```

Output:



```
C:\WINDOWS\system32\cmd.exe
D:\JQR>javac BiggestNo.java
D:\JQR>java BiggestNo
b is big
D:\JQR>_
```

- **Switch Statement:** When there are several options and we have to choose only one option from the available ones, we can use switch statement.

Syntax:

```
switch (expression)
{
    case value1: //statement sequence
                break;
    case value2: //statement sequence
```

```

                                break;
                                .....
                                case valueN: //statement sequence
                                    break;
                                default:      //default statement sequence
                                }

```

Here, depending on the value of the expression, a particular corresponding case will be executed.

Program 2: Write a program for using the switch statement to execute a particular task depending on color value.

//To display a color name depending on color value

```

class ColorDemo
{
    public static void main(String args[])
    {
        char color = 'r';
        switch (color)
        {
            case 'r': System.out.println ("red");           break;
            case 'g': System.out.println ("green");         break;
            case 'b': System.out.println ("blue");          break;
            case 'y': System.out.println ("yellow");        break;
            case 'w': System.out.println ("white");          break;
            default: System.out.println ("No Color Selected");
        }
    }
}

```

Output:



```

C:\WINDOWS\system32\cmd.exe
D:\JQR>javac ColorDemo.java
D:\JQR>java ColorDemo
red
D:\JQR>

```

- **Java's Iteration Statements:** Java's iteration statements are for, while and do-while. These statements are used to repeat same set of instructions specified number of times called loops. A loop repeatedly executes the same set of instructions until a termination condition is met.

- **while Loop:** while loop repeats a group of statements as long as condition is true. Once the condition is false, the loop is terminated. In while loop, the condition is tested first; if it is true, then only the statements are executed. while loop is called as entry control loop.

Syntax:

```

while (condition)
{
    statements;
}

```

Program 3: Write a program to generate numbers from 1 to 20.

//Program to generate numbers from 1 to 20.

```
class Natural
{
    public static void main(String args[])
    {
        int i=1;
        while (i <= 20)
        {
            System.out.print (i + "\t");
            i++;
        }
    }
}
```

Output:

```
C:\WINDOWS\system32\cmd.exe
D:\JQR>javac Natural.java
D:\JQR>java Natural
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20
D:\JQR>
```

- **do...while Loop:** do...while loop repeats a group of statements as long as condition is true. In do...while loop, the statements are executed first and then the condition is tested. do...while loop is also called as exit control loop.

Syntax:

```
do
{
    statements;
} while (condition);
```

Program 4: Write a program to generate numbers from 1 to 20.

//Program to generate numbers from 1 to 20.

```
class Natural
{
    public static void main(String args[])
    {
        int i=1;
        do
        {
            System.out.print (i + "\t");
            i++;
        } while (i <= 20);
    }
}
```

Output:

```
C:\WINDOWS\system32\cmd.exe
D:\JQR>javac Natural.java
D:\JQR>java Natural
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20
D:\JQR>
```

- **for Loop:** The for loop is also same as do...while or while loop, but it is more compact syntactically. The for loop executes a group of statements as long as a condition is true.

Syntax: for (expression1; expression2; expression3)
 {
 statements;
 }

Here, expression1 is used to initialize the variables, expression2 is used for condition checking and expression3 is used for increment or decrement variable value.

Program 5: Write a program to generate numbers from 1 to 20.

//Program to generate numbers from 1 to 20.

```
class Natural
{
    public static void main(String args[])
    {
        int i;
        for (i=1; i<=20; i++)
            System.out.print (i + "\t");
    }
}
```

Output:

```
C:\WINDOWS\system32\cmd.exe
D:\JQR>javac Natural.java
D:\JQR>java Natural
1 2 3 4 5 6 7 8 9 10
11 12 13 14 15 16 17 18 19 20
D:\JQR>
```

- **Java's Jump Statements:** Java supports three jump statements: break, continue and return. These statements transfer control to another part of the program.

- **break:**

- break can be used inside a loop to come out of it.
- break can be used inside the switch block to come out of the switch block.
- break can be used in nested blocks to go to the end of a block. Nested blocks represent a block written within another block.

Syntax: break; (or) break label;//here label represents the name of the block.

Program 6: Write a program to use break as a civilized form of goto.

//using break as a civilized form of goto

```
class BreakDemo
{
    public static void main (String args[])
    {
        boolean t = true;
        first:
        {
            second:
            {
                third:
                {
```

```

        System.out.println ("Before the break");
        if (t) break second; // break out of second block
        System.out.println ("This won't execute");
    }
    System.out.println ("This won't execute");
}
System.out.println ("This is after second block");
}
}
}

```

Output:



```
C:\WINDOWS\system32\cmd.exe

D:\JQR>javac BreakDemo.java

D:\JQR>java BreakDemo
Before the break
This is after second block

D:\JQR>
```

- **continue:** This statement is useful to continue the next repetition of a loop/ iteration. When continue is executed, subsequent statements inside the loop are not executed.

Syntax: continue;

Program 7: Write a program to generate numbers from 1 to 20.

```
//Program to generate numbers from 1 to 20.
```

```
class Natural
{
    public static void main (String args[])
    {
        int i=1;
        while (true)
        {
            System.out.print (i + "\t");
            i++;
            if (i <= 20 )
                continue;


            else
                break;

        }

    }

}
```

Output:



```
C:\WINDOWS\system32\cmd.exe

D:\JQR>javac Natural.java

D:\JQR>java Natural

1      2      3      4      5      6      7      8      9      10
11     12     13     14     15     16     17     18     19     20

D:\JQR>
```

- **return statement:**

- return statement is useful to terminate a method and come back to the calling method.
- return statement in main method terminates the application.
- return statement can be used to return some value from a method to a calling method.

Syntax: return;
 (or)
 return value; // value may be of any type

Program 8: Write a program to demonstrate return statement.

```
//Demonstrate return
class ReturnDemo
{
    public static void main(String args[])
    {
        boolean t = true;
        System.out.println ("Before the return");
        if (t)
            return;
        System.out.println ("This won't execute");
    }
}
```

Output:



```
C:\WINDOWS\system32\cmd.exe
D:\JQR>javac ReturnDemo.java
D:\JQR>java ReturnDemo
Before the return
D:\JQR>
```

Note: goto statement is not available in java, because it leads to confusion and forms infinite loops.

5. Accepting Input from Keyboard

A stream represents flow of data from one place to other place. Streams are of two types in java. Input streams which are used to accept or receive data. Output streams are used to display or write data. Streams are represented as classes in java.io package.

- **System.in:** This represents InputStream object, which by default represents standard input device that is keyboard.
- **System.out:** This represents PrintStream object, which by default represents standard output device that is monitor.
- **System.err:** This field also represents PrintStream object, which by default represents monitor. System.out is used to display normal messages and results whereas System.err is used to display error messages.

To accept data from the keyboard:

- Connect the keyboard to an input stream object. Here, we can use InputStreamReader that can read data from the keyboard.

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

- Connect InputStreamReader to BufferedReader, which is another input type of stream. We are using BufferedReader as it has got methods to read data properly, coming from the stream.

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

The above two steps can be combined and rewritten in a single statement as:

```
BufferedReader br = new BufferedReader (new InputStreamReader (System.in));
```

- Now, we can read the data coming from the keyboard using read () and readLine () methods available in BufferedReader class.

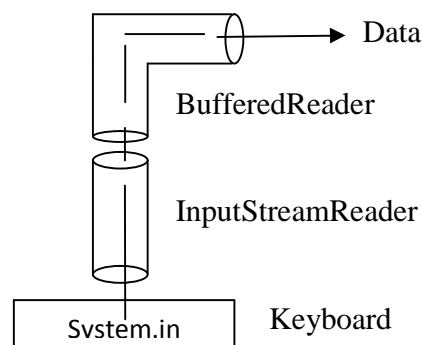


Figure: Reading data from keyboard

Accepting a Single Character from the Keyboard:

- Create a BufferedReader class object (br).
- Then read a single character from the keyboard using read() method as:

```
char ch = (char) br.read();
```
- The read method reads a single character from the keyboard but it returns its ASCII number, which is an integer. Since, this integer number cannot be stored into character type variable ch, we should convert it into char type by writing (char) before the method. int data type is converted into char data type, converting one data type into another data type is called type casting.

Accepting a String from Keyboard:

- Create a BufferedReader class object (br).
- Then read a string from the keyboard using readLine() method as:

```
String str = br.readLine ();
```
- readLine () method accepts a string from keyboard and returns the string into str. In this case, casting is not needed since readLine () is taking a string and returning the same data type.

Accepting an Integer value from Keyboard:

- First, we should accept the integer number from the keyboard as a string, using readLine () as:

```
String str = br.readLine ();
```
- Now, the number is in str, i.e. in form of a string. This should be converted into an int by using parseInt () method, method of Integer class as:

```
int n = Integer.parseInt (str);
```

If needed, the above two statements can be combined and written as:

```
int n = Integer.parseInt (br.readLine() );
```

- parseInt () is a static method in Integer class, so it can be called using class name as Integer.parseInt ().
- We are not using casting to convert String type into int type. The reason is String is a class and int is a fundamental data type. Converting a class type into a fundamental data type is not possible by using casting. It is possible by using the method Integer.parseInt().

Accepting a Float value from Keyboard:

- We can accept a float value from the keyboard with the help of the following statement:

```
float n = Float.parseFloat (br.readLine() );
```
- We are accepting a float value in the form of a string using br.readLine () and then passing the string to Float.parseFloat () to convert it into float. parseFloat () is a static method in Float class.

Accepting a Double value from Keyboard:

- We can accept a double value from the keyboard with the help of the following statement:

```
double n = Double.parseDouble (br.readLine() );
```
- We are accepting a double value in the form of a string using br.readLine () and then passing the string to Double.parseDouble () to convert it into double. parseDouble () is a static method in Double class.

Accepting Other Types of Values:

- To accept a byte value:

```
byte n = Byte.parseByte (br.readLine () );
```
- To accept a short value:

```
short n = Short.parseShort (br.readLine () );
```
- To accept a long value:

```
long n = Long.parseLong (br.readLine () );
```
- To accept a boolean value:

```
boolean x = Boolean.parseBoolean (br.readLine () );
```

If read () / readLine () method could not accept values due to some reason (like insufficient memory or illegal character), then it gives rise to a runtime error which is called by the name IOException, where IO stands for Input/Output and Exception represents runtime error. But we do not know how to handle this exception, in Java we can use throws command to throw the exception without handling it by writing:

throws IOException at the side of the method where read ()/ readLine () is used.

Program 1: Write a program to accept and display student details.

// Accepting and displaying student details.

```
import java.io.*;
class StudentDemo
{
    public static void main(String args[]) throws IOException
    {
        // Create BufferedReader object to accept data
        BufferedReader br =new BufferedReader (new InputStreamReader (System.in));
        //Accept student details
        System.out.print ("Enter roll number: ");
        int rno = Integer.parseInt (br.readLine());
        System.out.print ("Enter Gender (M/F): ");
        char gender = (char)br.read();
        br.skip (2);
        System.out.print ("Enter Student name: ");
        String name = br.readLine ()
        System.out.println ("Roll No.: " + rno);
        System.out.println ("Gender: " + gender);
        System.out.println ("Name: " + name);
    }
}
```

Output:



```
C:\WINDOWS\system32\cmd.exe
D:\JQR>javac StudentDemo.java
D:\JQR>java StudentDemo
Enter roll number: 10
Enter Gender (M/F): M
Enter Student name: Kiran Kumar
Roll No.: 10
Gender: M
Name: Kiran Kumar
D:\JQR>
```

In the above program after accepting gender of the student, br.skip (2) is used. The reason is that we used read () method to accept the gender value and then readLine () is used to accept the name. When we type M for gender and press enter, then it releases a \n code. So at gender column, we are giving two characters M and \n. But, read () method takes only the first character and rejects the next character, i.e. \n, which is trapped by the next readLine () method and name will accept \n. For this purpose, we can use skip () method of BufferedReader, which helps in skipping a specified number of characters. Suppose we take \n as two characters; now to skip them, we can write br.skip (2);

6. Arrays, Strings & StringBuffer

Arrays: An array represents a group of elements of same data type. Arrays are generally categorized into two types:

- Single Dimensional arrays (or 1 Dimensional arrays)
- Multi-Dimensional arrays (or 2 Dimensional arrays, 3 Dimensional arrays, ...)

Single Dimensional Arrays: A one dimensional array or single dimensional array represents a row or a column of elements. For example, the marks obtained by a student in 5 different subjects can be represented by a 1D array.

- We can declare a one dimensional array and directly store elements at the time of its declaration, as: `int marks[] = {50, 60, 55, 67, 70};`
- We can create a 1D array by declaring the array first and then allocate memory for it by using new operator, as: `int marks[]; //declare marks array`
`marks = new int[5]; //allot memory for storing 5 elements`

These two statements also can be written as: `int marks [] = new int [5];`

Program 1: Write a program to accept elements into an array and display the same.

// program to accept elements into an array and display the same.

import java.io.*;

class ArrayDemo1

```
{
    public static void main (String args[]) throws IOException
    {
        //Create a BufferedReader class object (br)
        BufferedReader br = new BufferedReader (new InputStreamReader (System.in));
        System.out.println ("How many elements: ");
        int n = Integer.parseInt (br.readLine ());
        //create a 1D array with size n
        int a[] = new int[n];
        System.out.print ("Enter elements into array : ");
        for (int i = 0; i<n;i++)
            a [i] = Integer.parseInt ( br.readLine ());
        System.out.print ("The entered elements in the array are: ");
        for (int i =0; i < n; i++)
            System.out.print (a[i] + "\t");
    }
}
```

Output:

```
C:\WINDOWS\system32\command.com
D:\JQR>javac ArrayDemo1.java
D:\JQR>java ArrayDemo1
How many elements: 5
Enter elements into array : 10
20
30
40
50
The entered elements in the array are: 10 20 30 40 50
D:\JQR>
```

Multi-Dimensional Arrays (2D, 3D ... arrays): Multi dimensional arrays represent 2D, 3D ... arrays. A two dimensional array is a combination of two or more (1D) one dimensional arrays. A three dimensional array is a combination of two or more (2D) two dimensional arrays.

- **Two Dimensional Arrays (2d array):** A two dimensional array represents several rows and columns of data. To represent a two dimensional array, we should use two pairs of square braces [] [] after the array name. For example, the marks obtained by a group of students in five different subjects can be represented by a 2D array.

- We can declare a two dimensional array and directly store elements at the time of its declaration, as:

```
int marks[ ] [ ] = { {50, 60, 55, 67, 70}, {62, 65, 70, 70, 81}, {72, 66, 77, 80, 69} };
```

- We can create a two dimensional array by declaring the array first and then we can allot memory for it by using new operator as:

```
int marks[ ] [ ]; //declare marks array
```

```
marks = new int[3][5]; //allot memory for storing 15 elements.
```

These two statements also can be written as: `int marks [] [] = new int[3][5];`

Program 2: Write a program to take a 2D array and display its elements in the form of a matrix.

//Displaying a 2D array as a matrix

class Matrix

```
{
    public static void main(String args[])
    {
        //take a 2D array
        int x[ ] [ ] = { {1, 2, 3}, {4, 5, 6} };
        // display the array elements
        for (int i = 0 ; i < 2 ; i++)
        {
            System.out.println ();
            for (int j = 0 ; j < 3 ; j++)
                System.out.print(x[i][j] + "\t");
        }
    }
}
```

Output:



```
C:\WINDOWS\system32\cmd.exe
D:\JQR>javac Matrix.java
D:\JQR>java Matrix
1      2      3
4      5      6
D:\JQR>
```

- **Three Dimensional arrays (3D arrays):** We can consider a three dimensional array as a combination of several two dimensional arrays. To represent a three dimensional array, we should use three pairs of square braces [] [] [] after the array name.

- We can declare a three dimensional array and directly store elements at the time of its declaration, as:

```
int arr[ ] [ ] [ ] = { { {50, 51, 52}, {60, 61, 62} }, { {70, 71, 72}, {80, 81, 82} } };
```

- We can create a three dimensional array by declaring the array first and then we can allot memory for it by using new operator as:

```
int arr[ ][ ] = new int[2][2][3]; //allot memory for storing 15 elements.
```

arrayname.length: If we want to know the size of any array, we can use the property 'length' of an array. In case of 2D, 3D length property gives the number of rows of the array.

Strings: A String represents group of characters. Strings are represented as String objects in java.

Creating Strings:

- We can declare a String variable and directly store a String literal using assignment operator.

```
String str = "Hello";
```
- We can create String object using new operator with some data.

```
String s1 = new String ("Java");
```
- We can create a String by using character array also.

```
char arr[] = { 'p','r','o','g','r','a','m' };
String s2 = new String (arr);
```
- We can create a String by passing array name to it, as:

```
String s3 = new String (str, 2, 3);
```

Here starting from 2nd character a total of 3 characters are copied into String s3.

String Class Methods:

Method	Description
String concat (String str)	Concatenates calling String with str. Note: + also used to do the same
int length ()	Returns length of a String
char charAt (int index)	Returns the character at specified location (from 0)
int compareTo (String str)	Returns a negative value if calling String is less than str, a positive value if calling String is greater than str or 0 if Strings are equal.
boolean equals (String str)	Returns true if calling String equals str. Note: == operator compares the references of the string objects. It does not compare the contents of the objects. equals () method compares the contents. While comparing the strings, equals () method should be used as it yields the correct result.
boolean equalsIgnoreCase (String str)	Same as above but ignores the case
boolean startsWith (String prefix)	Returns true if calling String starts with prefix
boolean endsWith (String suffix)	Returns true if calling String ends with suffix
int indexOf (String str)	Returns first occurrence of str in String.
int lastIndexOf (String str)	Returns last occurrence of str in the String. Note: Both the above methods return negative value, if str not

	found in calling String. Counting starts from 0.
String replace (char oldchar, char newchar)	returns a new String that is obtained by replacing all characters oldchar in String with newchar.
String substring (int beginIndex)	returns a new String consisting of all characters from beginIndex until the end of the String
String substring (int beginIndex, int endIndex)	returns a new String consisting of all characters from beginIndex until the endIndex.
String toLowerCase ()	converts all characters into lowercase
String toUpperCase ()	converts all characters into uppercase
String trim ()	eliminates all leading and trailing spaces

Program 3: Write a program using some important methods of String class.

// program using String class methods

class StrOps

```
{
    public static void main(String args [])
    {
        String str1 = "When it comes to Web programming, Java is #1.";
        String str2 = new String (str1);
        String str3 = "Java strings are powerful.";
        int result, idx;          char ch;
        System.out.println ("Length of str1: " + str1.length ());
        // display str1, one char at a time.
        for(int i=0; i < str1.length(); i++)
            System.out.print (str1.charAt (i));
        System.out.println ();
        if (str1.equals (str2) )
            System.out.println ("str1 equals str2");
        else
            System.out.println ("str1 does not equal str2");
        if (str1.equals (str3) )
            System.out.println ("str1 equals str3");
        else
            System.out.println ("str1 does not equal str3");
        result = str1.compareTo (str3);
        if(result == 0)
            System.out.println ("str1 and str3 are equal");
        else if(result < 0)
            System.out.println ("str1 is less than str3");
        else
            System.out.println ("str1 is greater than str3");
        str2 = "One Two Three One";           // assign a new string to str2
        idx = str2.indexOf ("One");
        System.out.println ("Index of first occurrence of One: " + idx);
        idx = str2.lastIndexOf("One");
        System.out.println ("Index of last occurrence of One: " + idx);
    }
}
```

}

Output:

```

C:\WINDOWS\system32\cmd.exe

D:\JQR>javac StrOps.java
D:\JQR>java StrOps
Length of str1: 46
When it comes to Web programming, Java is #1.
str1 equals str2
str1 does not equal str3
str1 is greater than str3
Index of first occurrence of One: 0
Index of last occurrence of One: 14
D:\JQR>

```

We can divide objects broadly as mutable and immutable objects. Mutable objects are those objects whose contents can be modified. Immutable objects are those objects, once created can not be modified. String objects are immutable. The methods that directly manipulate data of the object are not available in String class.

StringBuffer: StringBuffer objects are mutable, so they can be modified. The methods that directly manipulate data of the object are available in StringBuffer class.

Creating StringBuffer:

- We can create a StringBuffer object by using new operator and pass the string to the object, as:
StringBuffer sb = new StringBuffer ("Kiran");
- We can create a StringBuffer object by first allotting memory to the StringBuffer object using new operator and later storing the String into it as:
StringBuffer sb = new StringBuffer (30);

In general a StringBuffer object will be created with a default capacity of 16 characters. Here, StringBuffer object is created as an empty object with a capacity for storing 30 characters. Even if we declare the capacity as 30, it is possible to store more than 30 characters into StringBuffer. To store characters, we can use append () method as:

Sb.append ("Kiran");

StringBuffer Class Methods:

Method	Description
StringBuffer append (x)	x may be int, float, double, char, String or StringBuffer. It will be appended to calling StringBuffer
StringBuffer insert (int offset, x)	x may be int, float, double, char, String or StringBuffer. It will be inserted into the StringBuffer at offset.
StringBuffer delete (int start, int end)	Removes characters from start to end
StringBuffer reverse ()	Reverses character sequence in the StringBuffer
String toString ()	Converts StringBuffer into a String
int length ()	Returns length of the StringBuffer

Program 4: Write a program using some important methods of StringBuffer class.

// program using StringBuffer class methods

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

```
class Mutable
```

```
{    public static void main(String[] args) throws IOException
    {        // to accept data from keyboard
        BufferedReader br=new BufferedReader (new InputStreamReader (System.in));
        System.out.print ("Enter sur name : ");
        String sur=br.readLine ( );
        System.out.print ("Enter mid name : ");
        String mid=br.readLine ( );
        System.out.print ("Enter last name : ");
        String last=br.readLine ( );
        // create String Buffer object
        StringBuffer sb=new StringBuffer ( );
        // append sur, last to sb
        sb.append (sur);
        sb.append (last);
        // insert mid after sur
        int n=sur.length ( );
        sb.insert (n, mid);
        // display full name
        System.out.println ("Full name = "+sb);
        System.out.println ("In reverse =" +sb.reverse ( ));
    }
}
```

Output:



```
C:\WINDOWS\system32\cmd.exe
D:\JQR>javac Mutable.java
D:\JQR>java Mutable
Enter sur name : Chandra
Enter mid name : Sekhar
Enter last name : Azad
Full name = Chandra Sekhar Azad
In reverse = dazA rahkeS ardnahC
D:\JQR>_
```

7. Introduction to OOPs

Languages like Pascal, C, FORTRAN, and COBOL are called procedure oriented programming languages. Since in these languages, a programmer uses procedures or functions to perform a task. When the programmer wants to write a program, he will first divide the task into separate sub tasks, each of which is expressed as functions/ procedures. This approach is called procedure oriented approach.

The languages like C++ and Java use classes and object in their programs and are called Object Oriented Programming languages. The main task is divided into several modules and these are represented as classes. Each class can perform some tasks for which several methods are written in a class. This approach is called Object Oriented approach.

Difference between Procedure Oriented Programming and OOP:

Procedure Oriented Programming	Object Oriented Programming
1. Main program is divided into small parts depending on the functions.	1. Main program is divided into small object depending on the problem.
2. The Different parts of the program connect with each other by parameter passing & using operating system.	2. Functions of object linked with object using message passing.
3. Every function contains different data.	3. Data & functions of each individual object act like a single unit.
4. Functions get more importance than data in program.	4. Data gets more importance than functions in program.
5. Most of the functions use global data.	5. Each object controls its own data.
6. Same data may be transfer from one function to another	6. Data does not possible transfer from one object to another.
7. There is no perfect way for data hiding.	7. Data hiding possible in OOP which prevent illegal access of function from outside of it. This is one of the best advantages of OOP also.
8. Functions communicate with other functions maintaining as usual rules.	8. One object link with other using the message passing.
9. More data or functions can not be added with program if necessary. For this purpose full program need to be change.	9. More data or functions can be added with program if necessary. For this purpose full program need not to be change.
10. To add new data in program user should be ensure that function allows it.	10. Message passing ensure the permission of accessing member of an object from other object.
11. Top down process is followed for program design.	11. Bottom up process is followed for program design.
12. Example: Pascal, Fortran	12. Example: C++, Java.

Features of OOP:

- **Class:** In object-oriented programming, a class is a programming language construct that is used as a blueprint to create objects. This blueprint includes attributes and methods that the created objects all share. Usually, a class represents a person, place, or thing - it is an abstraction of a concept within a computer program. Fundamentally, it encapsulates the state

and behavior of that which it conceptually represents. It encapsulates state through data placeholders called member variables; it encapsulates behavior through reusable code called methods.

General form of a class:

```
class class_name
{
    Properties (variables);
    Actions (methods);
}
```

e.g.:

```
class Student
{
    //properties -- variables
    int rollNo;
    String name;
    //methods -- actions
    void display ()
    {
        System.out.println ("Student Roll Number is: " + rollNo);
        System.out.println ("Student Name is: " + name);
    }
}
```

Note: Variables inside a class are called as instance variables.

Variables inside a method are called as method variables.

- **Object:** An Object is a real time entity. An object is an instance of a class. Instance means physically happening. An object will have some properties and it can perform some actions. Object contains variables and methods. The objects which exhibit similar properties and actions are grouped under one class. “To give a real world analogy, a house is constructed according to a specification. Here, the specification is a blueprint that represents a class, and the constructed house represents the object”.
 - To access the properties and methods of a class, we must declare a variable of that class type. This variable does not define an object. Instead, it is simply a variable that can refer to an object.
 - We must acquire an actual, physical copy of the object and assign it to that variable. We can do this using **new** operator. The new operator dynamically allocates memory for an object and returns a reference to it. This reference is, more or less, the address in memory of the object allocated by new. This reference is then stored in the variable. Thus, in Java, all class objects must be dynamically allocated.

General form of an Object:

```
Class_name variable_name;           // declare reference to object
variable_name = new Class_name ( ); // allocate an object
```

e.g.:

```
Student s;           // s is reference variable
s = new Student ();  // allocate an object to reference variable s
```

The above two steps can be combined and rewritten in a single statement as:

```
Student s = new Student ();
```

Now we can access the properties and methods of a class by using object with dot operator as:

```
s.rollNo, s.name, s.display ()
```

- **Encapsulation:** Wrapping up of data (variables) and methods into single unit is called Encapsulation. Class is an example for encapsulation. Encapsulation can be described as a protective barrier that prevents the code and data being randomly accessed by other code defined outside the class. Encapsulation is the technique of making the fields in a class private and providing access to the fields via methods. If a field is declared private, it cannot be accessed by anyone outside the class.

e.g.:

```
class Student
{
    private int rollNo;
    private String name;
    //methods -- actions
    void display ()
    {
        System.out.println ("Student Roll Number is: " + rollNo);
        System.out.println ("Student Name is: " + name);
    }
}
```

- **Abstraction:** Providing the essential features without its inner details is called abstraction (or) hiding internal implementation is called Abstraction. We can enhance the internal implementation without effecting outside world. Abstraction provides security. A class contains lot of data and the user does not need the entire data. The advantage of abstraction is that every user will get his own view of the data according to his requirements and will not get confused with unnecessary data. A bank clerk should see the customer details like account number, name and balance amount in the account. He should not be entitled to see the sensitive data like the staff salaries, profit or loss of the bank etc. So such data can be abstracted from the clerks view.

e.g.:

```
class Bank
{
    private int accno;
    private String name;
    private float balance;
    private float profit;
    private float loan;
    void display_to_clerk ()
    {
        System.out.println ("Accno = " + accno);
        System.out.println ("Name = " + name);
        System.out.println ("Balance = " + balance);
    }
}
```

In the preceding class, inspite of several data items, the display_to_clerk () method is able to access and display only the accno, name and balance values. It cannot access profit and loan of the customer. This means the profit and loan data is hidden from the view of the bank clerk.

- **Inheritance:** Acquiring the properties from one class to another class is called inheritance (or) producing new class from already existing class is called inheritance. Reusability of code

is main advantage of inheritance. In Java inheritance is achieved by using extends keyword. The properties with access specifier private cannot be inherited.

e.g.:

```
class Parent
{
    String parentName;
    String familyName;
}
class Child extends Parent
{
    String childName;
    int childAge;
    void printMyName()
    {
        System.out.println ("My name is"+childName+" "+familyName);
    }
}
```

In the above example, the child has inherited its family name from the parent class just by inheriting the class.

- **Polymorphism:** The word polymorphism came from two Greek words 'poly' means 'many' and 'morphos' means 'forms'. Thus, polymorphism represents the ability to assume several different forms. The ability to define more than one function with the same name is called Polymorphism

e.g.:

```
int add (int a, int b)
float add (float a, int b)
float add (int a , float b)
void add (float a)
int add (int a)
```

- **Message Passing:** Calling a method in a class is called message passing. We can call methods of a class by using object with dot operator as:

e.g.: s.display (); ob.add (2, 5); ob.printMyName ();

Program 1: Write a program to display details of student using class and object.

//Program to display the details of a student using class and object

```
class Student
{
    int rollNo;           //properties -- variables
    String name;
    void display ()       //method -- action
    {
        System.out.println ("Student Roll Number is: " + rollNo);
        System.out.println ("Student Name is: " + name);
    }
}
class StudentDemo
{
    public static void main(String args[])
    {
```

```

        //create an object to Student class
        Student s = new Student ();
        //call display () method inside Student class using object s
        s.display ();
    }
}

```

Output:

```

C:\WINDOWS\system32\cmd.exe
D:\JQR>javac StudentDemo.java
D:\JQR>java StudentDemo
Student Roll Number is: 0
Student Name is: null
D:\JQR>_

```

When the programmer does not initialize the instance variables, java compiler will write code and initializes the variables with default values.

Data Type	Default Value
Int	0
Float	0.0
Double	0.0
Char	Space
String	null
Class	null
Boolean	false

Initializing Instance Variables:

- **Type 1:** We can initialize instance variables directly in the class using assignment operator. In this type every object is initialized with the same data.

```

int rollNo = 101;
String name = "Kiran";

```

Program 2: Let us rewrite the Program 1.

//Program to display the details of a student using class and object

```

class Student
{
    int rollNo = 101;
    String name = "Surya";
    void display ()
    {
        System.out.println ("Student Roll Number is: " + rollNo);
        System.out.print ("Student Name is: " + name);
    }
}
class StudentDemo
{
    public static void main(String args[])
    {
        Student s1 = new Student ();
        System.out.println ("First Student Details : " );
    }
}

```

```

        s1.display ();
        Student s2 = new Student ();
        System.out.println ("Second Student Details : " );
        s2.display ();
    }
}

```

Output:


```

C:\WINDOWS\system32\cmd.exe
D:\JQR>javac StudentDemo.java
D:\JQR>java StudentDemo
First Student Details :
Student Roll Number is: 101
Student Name is: Surya
Second Student Details :
Student Roll Number is: 101
Student Name is: Surya
D:\JQR>

```

- **Type 2:** We can initialize one class instance variables in another class using reference variable.

```

s.rollNo = 101;
s.name = "Kiran";

```

Program 3: Let us rewrite the Program 1.

//Program to display the details of a student using class and object
class Student

```

{
    int rollNo;
    String name;
    void display ()
    {
        System.out.println ("Student Roll Number is: " + rollNo);
        System.out.print ("Student Name is: " + name);
    }
}
class StudentDemo
{
    public static void main(String args[])
    {
        Student s1 = new Student ();
        System.out.println ("First Student Details : " );
        s1.rollNo = 101;
        s1.name = "Suresh";
        s1.display ();
        Student s2 = new Student ();
        System.out.println ("Second Student Details : " );
        s2.rollNo = 102;
        s2.name = "Ramesh";
        s2.display ();
    }
}

```

Output:


```

C:\WINDOWS\system32\cmd.exe
D:\JQR>javac StudentDemo.java
D:\JQR>java StudentDemo
First Student Details :
Student Roll Number is: 101
Student Name is: Suresh
Second Student Details :
Student Roll Number is: 102
Student Name is: Ramesh
D:\JQR>

```

In this type of initialization the properties (variables in the class) are not available, if they are declared as private.

Access Specifiers: An access specifier is a key word that represents how to access a member of a class. There are four access specifiers in java.

- **private:** private members of a class are not available outside the class.
- **public:** public members of a class are available anywhere outside the class.
- **protected:** protected members are available outside the class.
- **default:** if no access specifier is used then default specifier is used by java compiler. Default members are available outside the class.

- **Type 3:** We can initialize instance variables using a constructor.

Constructor:

- A constructor is similar to a method that initializes the instance variables of a class.
- A constructor name and classname must be same.
- A constructor may have or may not have parameters. Parameters are local variables to receive data.
- A constructor without any parameters is called default constructor.

e.g.

```

class Student
{
    int rollNo;
    String name;
    Student ()
    {
        rollNo = 101;
        name = "Kiran";
    }
}

```

- A constructor with one or more parameters is called parameterized constructor.

e.g.

```

class Student
{
    int rollNo;
    String name;
    Student (int r, String n)
    {
        rollNo = r;
        name = n;
    }
}

```

- A constructor does not return any value, not even void.

- A constructor is called and executed at the time of creating an object.
- A constructor is called only once per object.
- Default constructor is used to initialize every object with same data where as parameterized constructor is used to initialize each object with different data.
- If no constructor is written in a class then java compiler will provide default values.

Program 4: Write a program to initialize student details using default constructor and display the same.

//Program to initialize student details using default constructor and displaying the same.

```
class Student
{
    int rollNo;
    String name;
    Student ()
    {
        rollNo = 101;
        name = "Suresh";
    }
    void display ()
    {
        System.out.println ("Student Roll Number is: " + rollNo);
        System.out.println ("Student Name is: " + name);
    }
}
class StudentDemo
{
    public static void main(String args[])
    {
        Student s1 = new Student ();
        System.out.println ("s1 object contains: ");
        s1.display ();
        Student s2 = new Student ();
        System.out.println ("s2 object contains: ");
        s2.display ();
    }
}
```

Output:



```
C:\WINDOWS\system32\cmd.exe
D:\JQR>javac StudentDemo.java
D:\JQR>java StudentDemo
s1 object contains:
Student Roll Number is: 101
Student Name is: Suresh
s2 object contains:
Student Roll Number is: 101
Student Name is: Suresh
D:\JQR>
```

Program 5: Write a program to initialize student details using Parameterized constructor and display the same.

//Program to initialize student details using parameterized constructor

```
class Student
{
    int rollNo;
```

```

        String name;
        Student (int r, String n)
        {
            rollNo = r;
            name = n;
        }
        void display ()
        {
            System.out.println ("Student Roll Number is: " + rollNo);
            System.out.println ("Student Name is: " + name);
        }
    }
    class StudentDemo
    {
        public static void main(String args[])
        {
            Student s1 = new Student (101, "Suresh");
            System.out.println ("s1 object contains: ");
            s1.display ();
            Student s2 = new Student (102, "Ramesh");
            System.out.println ("s2 object contains: ");
            s2.display ();
        }
    }
}

```

Output:


```

C:\WINDOWS\system32\cmd.exe
D:\JQR>javac StudentDemo.java
D:\JQR>java StudentDemo
s1 object contains:
Student Roll Number is: 101
Student Name is: Suresh
s2 object contains:
Student Roll Number is: 102
Student Name is: Ramesh
D:\JQR>

```

The keyword ‘this’: There will be situations where a method wants to refer to the object which invoked it. To perform this we use ‘this’ keyword. There are no restrictions to use ‘this’ keyword we can use this inside any method for referring the current object. This keyword is always a reference to the object on which the method was invoked. We can use ‘this’ keyword wherever a reference to an object of the current class type is permitted. ‘this’ is a key word that refers to present class object. It refers to

- Present class instance variables
- Present class methods.
- Present class constructor.

Program 6: Write a program to use ‘this’ to refer the current class parameterized constructor and current class instance variable.

```

//this demo
class Person
{
    String name;

```

```

    Person ( )
    {
        this ("Ravi Sekhar"); // calling present class parameterized constructor
        this.display ( ); // calling present class method
    }
    Person (String name)
    {
        this.name = name; // assigning present class variable with parameter "name"
    }
    void display ( )
    {
        System.out.println ("Person Name is = " + name);
    }
}
class ThisDemo
{
    public static void main(String args[])
    {
        Person p = new Person ( );
    }
}

```

Output:

```

C:\WINDOWS\system32\cmd.exe
D:\JQR>javac ThisDemo.java
D:\JQR>java ThisDemo
Person Name is = Ravi Sekhar
D:\JQR>

```

Garbage Collection: Generally memory is allocated to objects by using ‘new’ operator and deleting an allocated memory is uncommon. This deletion of memory is supported by delete operator in C++ but this deletion of allocated memory works automatically in Java. This automatic deletion of already allocated but unused memory is called as garbage collection. This operation of garbage collection is accomplished by a method named “gc ()”. This method is used for garbage collection.

The finalize() Method: It is possible to define a method that will be called just before an object's final destruction by the garbage collector. This method is called finalize() method. To add a finalizer to a class, simply define the finalize() method. The Java runtime calls that method whenever it is about to recycle an object of that class. Inside the finalize() method specify those actions that must be performed before an object is destroyed. The finalize() method has this general form:

```

protected void finalize ( )
{
    // finalization code here
}

```

Here, the keyword protected is a specifier that prevents access to finalize () by code defined outside its class. This means that you cannot know when or even if finalize () will be executed. For example, if your program ends before garbage collection occurs, finalize () will not execute.

8. Methods & Inner Class

Methods: A method represents a group of statements to perform a task. A method contains two parts: method header (or) method prototype is first part. This part contains method name, method parameters and method returntype.

return_type methodname (param1, param2,)

e.g.: double sum (double d1, double d2), void sum (), float power (float x, int n), etc.

The second part contains method body. This part represents the logic to perform the task in the form of a group of statements.

```
{
    Statements;
}
```

Note: If a method returns a value then return statement should be written in method body.

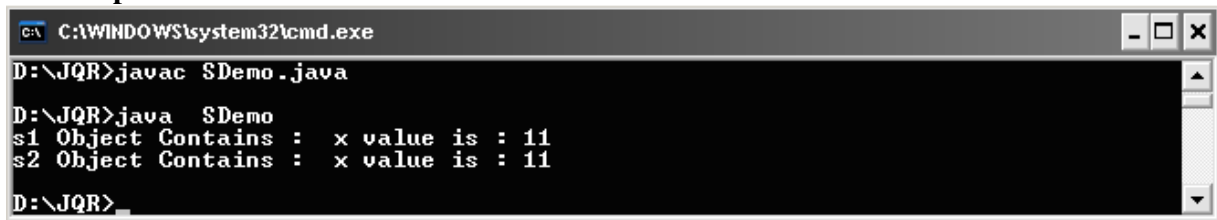
e.g.: return x; return 1; return (x+y-2);

- **Instance Methods:**

- Methods which act upon instance variables of a class are called instance methods.
- To call instance methods use objectname.methodname.
- Instance variable is a variable whose separate copy is available in every object.
- Any modifications to instance variable in one object will not affect the instance variable of other objects. These variables are created on heap.

Program 1: Write a program to access instance variable using instance method.

```
//instance method accessing instance variable
class Sample
{
    int x = 10;
    void display( )
    {
        x++;
        System.out.println (" x value is : " + x);
    }
}
class SDemo
{
    public static void main(String args[])
    {
        Sample s1 = new Sample( );
        System.out.print ("s1 Object Contains : ");
        s1.display ();
        Sample s2 = new Sample( );
        System.out.print ("s2 Object Contains : ");
        s2.display ();
    }
}
```

Output:


```

C:\WINDOWS\system32\cmd.exe
D:\JQR>javac SDemo.java
D:\JQR>java SDemo
s1 Object Contains : x value is : 11
s2 Object Contains : x value is : 11
D:\JQR>

```

Note: Instance methods can read and act upon static variables also.

- **Static Methods:**

- Static methods can read and act upon static variables.
- Static methods cannot read and act upon instance variables.
- Static variable is a variable whose single copy is shared by all the objects.
- Static methods are declared using keyword static.
- Static methods can be called using objectname.methodname (or) classname.methodname.
- From any object, if static variable is modified it affects all the objects. Static variables are stored on method area.

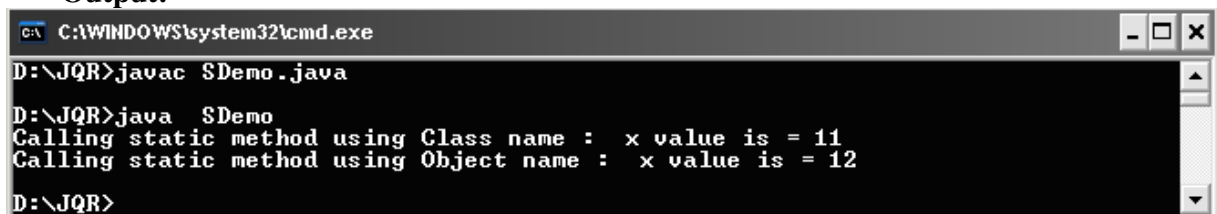
Program 2: Write a program to access static variable using static method.

//static method accessing static variable

```

class Sample
{
    static int x = 10;
    static void display( )
    {
        x++;
        System.out.println (" x value is = " + x);
    }
}
class SDemo
{
    public static void main(String args[])
    {
        System.out.print ("Calling static method using Class name : ");
        Sample.display ();
        Sample s1 = new Sample ( );
        System.out.print ("Calling static method using Object name : ");
        s1.display ();
    }
}

```

Output:


```

C:\WINDOWS\system32\cmd.exe
D:\JQR>javac SDemo.java
D:\JQR>java SDemo
Calling static method using Class name : x value is = 11
Calling static method using Object name : x value is = 12
D:\JQR>

```

Inner Class: A class with in another class is called Inner class. When the programmer wants to restrict the access of entire code of a class, creates an inner class as a private class. The way to access the inner class is through its outer class only.

- Inner class is a safety mechanism.
- Inner class is hidden in outer class from other classes.
- Only inner class can be private.
- An object to Inner class can be created only in its outer class.
- An object to Inner class cannot be created in any other class.
- Outer class object and Inner class objects are created in separate memory locations.
- Outer class members are available to Inner class object.
- Inner class object will have an additional invisible field called 'this\$0' that stores a reference of outer class object.
- Inner class members are referenced as: this.member;
- Outer class members are referred as: Outerclass.this.member;

Program 3: Write a program to access private members of a class using inner class.

```
// inner class demo
class Bank
{
    private double bal, rate;
    Bank (double b, double r)
    {
        bal=b;
        rate = r;
    }
    void display ( )
    {
        Interest in=new Interest ();
        in.calculateInterest ( );
        System.out.println ("New Balance : " + bal);
    }
    private class Interest
    {
        void calculateInterest ( )
        {
            System.out.println ("Balance = " + bal);
            double interest=bal* rate/100;
            System.out.println ("interest = "+interest);
            bal+=interest;
        }
    }
}
class InnerDemo
{
    public static void main (String args[])
    {
        Bank account = new Bank (20000, 5);
        account.display ();
    }
}
```

Output:

```
C:\WINDOWS\system32\cmd.exe
D:\JQR>javac InnerDemo.java
D:\JQR>java InnerDemo
Balance = 20000.0
interest = 1000.0
New Balance : 21000.0
D:\JQR>
```

9. Inheritance

Inheritance: Creating new class from existing class such that the features of existing class are available to the new class is called inheritance. Already existing class is called super class & produced class is called sub class. Using inheritance while creating sub classes a programmer can reuse the super class code without rewriting it.

Syntax: class subclass_name extends superclass_name

e.g.: class Child extends Parent

Program 1: Write a program to create a Person class which contains general details of a person and create a sub class Employ which contains company details of a person. Reuse the general details of the person in its sub class.

// Inheritance Example

```
class Person
{
    String name;
    String permanentAddress;
    int age;
    void set_PermanentDetails (String name, String permanentAddress, int age)
    {
        this.name = name;
        this.permanentAddress = permanentAddress;
        this.age = age;
    }
    void get_PermanentDetails ()
    {
        System.out.println ("Name : " + name);
        System.out.println ("Permanent Address : " + permanentAddress);
        System.out.println ("Age : " + age);
    }
}
class Employ extends Person
{
    int id;
    String companyName;
    String companyAddress;
    Employ (int id, String name, String permanentAddress, int age,
            String companyName, String companyAddress)
    {
        this.id = id;
        set_PermanentDetails (name, permanentAddress, age);
        this.companyName = companyName;
        this.companyAddress = companyAddress;
    }
    void get_EmployDetails ()
    {
        System.out.println ("Employ Id : " + id);
        get_PermanentDetails ();
        System.out.println ("Company Name : "+ companyName);
        System.out.println ("Company Address : "+companyAddress);
    }
}
```



```

}
class InherDemo
{
    public static void main (String args [])
    {
        Employ e1 = new Employ (101, "Suresh Kumar", "18-Madhura Nagar-Tirupati",
                                29, "Centris Software- Chennai", "20-RVS Nagar");
        e1.get_EmployDetails ();
    }
}

```

Output:


```

C:\WINDOWS\system32\cmd.exe
D:\JQR>javac InherDemo.java
D:\JQR>java InherDemo
Employ Id : 101
Name : Suresh Kumar
Permanent Address : 18-Madhura Nagar-Tirupati
Age :29
Company Name : Centris Software- Chennai
Company Address : 20-RVS Nagar
D:\JQR>

```

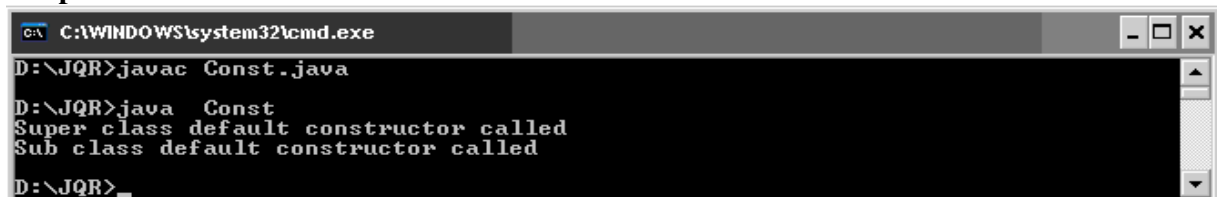
Program 2: Write a program to illustrate the order of calling of default constructor in super and sub class.

// Default constructors in super and sub class

```

class One
{
    One ( )          //super class default constructor
    {
        System.out.println ("Super class default constructor called");
    }
}
class Two extends One
{
    Two ( )          //sub class default constructor
    {
        System.out.println ("Sub class default constructor called");
    }
}
class Const
{
    public static void main (String args[])
    {
        Two t=new Two ( ); //create sub class object
    }
}

```

Output:


```

C:\WINDOWS\system32\cmd.exe
D:\JQR>javac Const.java
D:\JQR>java Const
Super class default constructor called
Sub class default constructor called
D:\JQR>

```

- Super class default constructor is available to sub class by default.
- First super class default constructor is executed then sub class default constructor is executed.
- Super class parameterized constructor is not automatically available to subclass. super is the key word that refers to super class.

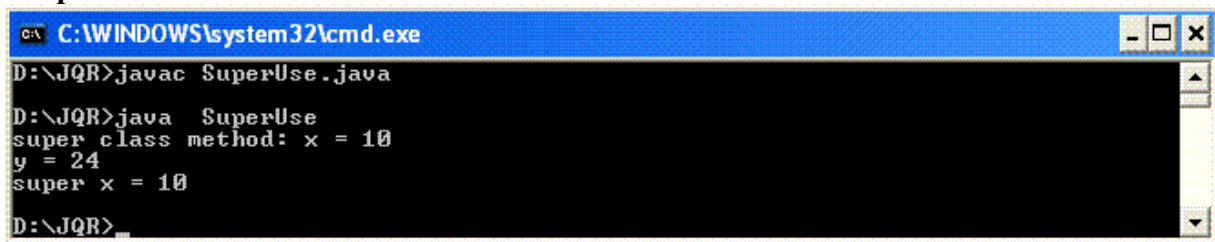
The keyword ‘super’:

- super can be used to refer super class variables as: super.variable
- super can be used to refer super class methods as: super.method ()
- super can be used to refer super class constructor as: super (values)

Program 3: Write a program to access the super class method, super class parameterized constructor and super class instance variable by using super keyword from sub class.

// super refers to super class- constructors, instance variables and methods

```
class A
{
    int x;
    A (int x)
    {
        this.x = x;
    }
    void show( )
    {
        System.out.println("super class method: x = "+x);
    }
}
class B extends A
{
    int y;
    B (int a,int b)
    {
        super(a);    // (or) x=a;
        y=b;
    }
    void show( )
    {
        super.show ();
        System.out.println ("y = "+y);
        System.out.println ("super x = " + super.x);
    }
}
class SuperUse
{
    public static void main(String args[])
    {
        B ob = new B (10, 24);
        ob.show ( );
    }
}
```

Output:A screenshot of a Windows command prompt window. The title bar is blue and contains the text 'C:\WINDOWS\system32\cmd.exe'. The command prompt shows the following sequence of commands and output:
D:\JQR>javac SuperUse.java
D:\JQR>java SuperUse
super class method: x = 10
y = 24
super x = 10
D:\JQR>
The output shows the result of running a Java program that demonstrates the use of the 'super' keyword in a subclass constructor.

- Super key word is used in sub class only.
- The statement calling super class constructor should be the first one in sub class constructor.

10. Polymorphism

Polymorphism came from the two Greek words 'poly' means many and morphos means forms. If the same method has ability to take more than one form to perform several tasks then it is called polymorphism. It is of two types: Dynamic polymorphism and Static polymorphism.

- **Dynamic Polymorphism:** The polymorphism exhibited at run time is called dynamic polymorphism. In this dynamic polymorphism a method call is linked with method body at the time of execution by JVM. Java compiler does not know which method is called at the time of compilation. This is also known as dynamic binding or run time polymorphism. Method overloading and method overriding are examples of Dynamic Polymorphism in Java.
 - **Method Overloading:** Writing two or more methods with the same name, but with a difference in the method signatures is called method over loading. Method signature represents the method name along with the method parameters. In method over loading JVM understands which method is called depending upon the difference in the method signature. The difference may be due to the following:
 - There is a difference in the no. of parameters.


```
void add (int a,int b)
void add (int a,int b,int c)
```
 - There is a difference in the data types of parameters.


```
void add (int a,float b)
void add (double a,double b)
```
 - There is a difference in the sequence of parameters.


```
void swap (int a,char b)
void swap (char a,int b)
```

Program 1: Write a program to create a class which contains two methods with the same name but with different signatures.

```
// overloading of methods ----- Dynamic polymorphism
class Sample
{
    void add(int a,int b)
    {
        System.out.println ("sum of two="+ (a+b));
    }
    void add(int a,int b,int c)
    {
        System.out.println ("sum of three="+ (a+b+c));
    }
}
class OverLoad
{
    public static void main(String[] args)
    {
        Sample s=new Sample ( );
        s.add (20, 25);
        s.add (20, 25, 30);
    }
}
```

Output:

```

C:\WINDOWS\system32\cmd.exe
D:\JQR>javac OverLoad.java
D:\JQR>java OverLoad
sum of two=45
sum of three=75
D:\JQR>

```

- **Method Overriding:** Writing two or more methods in super & sub classes with same name and same signatures is called method overriding. In method overriding JVM executes a method depending on the type of the object.

Program 2: Write a program that contains a super and sub class which contains a method with same name and same method signature, behavior of the method is dynamically decided.

//overriding of methods ----- Dynamic polymorphism

```

class Animal
{
    void move()
    {
        System.out.println ("Animals can move");
    }
}
class Dog extends Animal
{
    void move()
    {
        System.out.println ("Dogs can walk and run");
    }
}
public class OverRide
{
    public static void main(String args[])
    {
        Animal a = new Animal (); // Animal reference and object
        Animal b = new Dog (); // Animal reference but Dog object
        a.move (); // runs the method in Animal class
        b.move (); //Runs the method in Dog class
    }
}

```

Output:

```

C:\WINDOWS\system32\cmd.exe
D:\JQR>javac OverRide.java
D:\JQR>java OverRide
Animals can move
Dogs can walk and run
D:\JQR>

```

Achieving method overloading & method overriding using instance methods is an example of dynamic polymorphism.

- **Static Polymorphism:** The polymorphism exhibited at compile time is called Static polymorphism. Here the compiler knows which method is called at the compilation. This is also called compile time polymorphism or static binding. Achieving method overloading & method overriding using private, static and final methods is an example of Static Polymorphism.

Program 3: Write a program to illustrate static polymorphism.

//Static Polymorphism

```
class Animal
{
    static void move ()
    {
        System.out.println ("Animals can move");
    }
}
class Dog extends Animal
{
    static void move ()
    {
        System.out.println ("Dogs can walk and run");
    }
}
public class StaticPoly
{
    public static void main(String args[])
    {
        Animal.move ();
        Dog.move ();
    }
}
```

Output:



```
C:\WINDOWS\system32\cmd.exe
D:\JQR>javac StaticPoly.java
D:\JQR>java StaticPoly
Animals can move
Dogs can walk and run
D:\JQR>
```

The keyword ‘final’:

- final keyword before a class prevents inheritance.
e.g.: final class A
class B extends A //invalid
- final keyword before a method prevents overriding.
- final keyword before a variable makes that variable as a constant.
e.g.: final double PI = 3.14159; //PI is a constant.

Type Casting: Converting one data type into another data type is called casting. Type cast operator is used to convert one data type into another data type. Data type represents the type of the data stored into a variable. There are two kinds of data types:

- **Primitive Data type:** Primitive data type represents singular values.
e.g.: byte, short, int, long, float, double, char, boolean.

Using casting we can convert a primitive data type into another primitive data type. This is done in two ways, widening and narrowing.

- **Widening:** Converting a lower data type into higher data type is called widening.
byte, short, int, long , float, double
e.g.: char ch = 'a';
int n = (int) ch;
- e.g.:** int n = 12;
float f = (float) n;
- **Narrowing:** Converting a higher data type into lower data type is called narrowing.
e.g.: int i = 65;
char ch = (char) i;
- e.g.:** float f = 12.5;
int i = (int) f;

- **Referenced Data type:** Referenced data type represents multiple values.

e.g.: class, String

Using casting we can convert one class type into another class type if they are related by means of inheritance.

- **Generalization:** Moving back from subclass to super class is called generalization or widening or upcasting.
- **Specialization:** Moving from super class to sub class is called specialization or narrowing or downcasting.

Program 4: Write a program to convert one class type into another class type.

// conversion of one class type into another class type

```
class One
{
    void show1()
    {
        System.out.println ("One's method");
    }
}
class Two extends One
{
    void show2()
    {
        System.out.println ("Two's method");
    }
}
class Ex3
{
    public static void main(String args[])
    {
        /* If super class reference is used to refer to super class object then only super class
        members are available to programmer. */
        One ob1 = new One ();
        ob1.show1 ();
        /* If sub class reference is used to refer to sub class object then super class members as
        well as sub class members are available to the programmer. */
        Two ob2 = new Two();
        ob2.show1();
    }
}
```

```
        ob2.show2();
/* If super class reference is used to refer to sub class object then super class methods are
available, sub class methods are not available unless they override super class methods */
        One ob3 = (One) new Two();          // Generalization
        ob3.show1();
/* It is not possible to access any methods if we use subclass object to refer to super class
as above */
        Two ob4 = (Two) new One();
        ob4.show1();
        ob4.show2();
    // Specialization
        One ob5 = (One) new Two();
        Two ob6 = (Two) ob5;
        ob6.show1();
        ob6.show2();
    }
}
```

Note: Using casting it is not possible to convert a primitive data type into a referenced data type and vice-versa. For this we are using Wrapper classes.

11. Abstract Class

A method with method body is called concrete method. In general any class will have all concrete methods. A method without method body is called abstract method. A class that contains abstract method is called abstract class. It is possible to implement the abstract methods differently in the subclasses of an abstract class. These different implementations will help the programmer to perform different tasks depending on the need of the sub classes. Moreover, the common members of the abstract class are also shared by the sub classes.

The abstract methods and abstract class should be declared using the keyword `abstract`. We cannot create objects to abstract class because it is having incomplete code. Whenever an abstract class is created, subclass should be created to it and the abstract methods should be implemented in the subclasses, then we can create objects to the subclasses.

- An abstract class is a class with zero or more abstract methods
- An abstract class contains instance variables & concrete methods in addition to abstract methods.
- It is not possible to create objects to abstract class.
- But we can create a reference of abstract class type.
- All the abstract methods of the abstract class should be implemented in its sub classes.
- If any method is not implemented, then that sub class should be declared as 'abstract'.
- Abstract class reference can be used to refer to the objects of its sub classes.
- Abstract class references cannot refer to the individual methods of sub classes.
- A class cannot be both 'abstract' & 'final'.

e.g.: `final abstract class A // invalid`

Program 1: Write an example program for abstract class.

// Using abstract methods and classes.

```
abstract class Figure
{
    double dim1;
    double dim2;
    Figure (double a, double b)
    {
        dim1 = a;
        dim2 = b;
    }
    abstract double area ();    // area is now an abstract method
}
class Rectangle extends Figure
{
    Rectangle (double a, double b)
    {
        super (a, b);
    }
    double area ()    // override area for rectangle
    {
        System.out.println ("Inside Area of Rectangle.");
        return dim1 * dim2;
    }
}
```

```
class Triangle extends Figure
{
    Triangle (double a, double b)
    {
        super (a, b);
    }
    double area()          // override area for right triangle
    {
        System.out.println ("Inside Area of Triangle.");
        return dim1 * dim2 / 2;
    }
}
class AbstractAreas
{
    public static void main(String args[])
    {
        // Figure f = new Figure(10, 10); // illegal now
        Rectangle r = new Rectangle(9, 5);
        Triangle t = new Triangle(10, 8);
        System.out.println("Area is " + r.area());
        System.out.println("Area is " + t.area());
    }
}
```

Output:

```
C:\WINDOWS\system32\cmd.exe
D:\JQR>javac AbstractAreas.java
D:\JQR>java AbstractAreas
Inside Area of Rectangle.
Area is 45.0
Inside Area of Triangle.
Area is 40.0
D:\JQR>_
```

12. Interface

A programmer uses an abstract class when there are some common features shared by all the objects. A programmer writes an interface when all the features have different implementations for different objects. Interfaces are written when the programmer wants to leave the implementation to third party vendors. An interface is a specification of method prototypes. All the methods in an interface are abstract methods.

- An interface is a specification of method prototypes.
- An interface contains zero or more abstract methods.
- All the methods of interface are public, abstract by default.
- An interface may contain variables which are by default public static final.
- Once an interface is written any third party vendor can implement it.
- All the methods of the interface should be implemented in its implementation classes.
- If any one of the method is not implemented, then that implementation class should be declared as abstract.
- We cannot create an object to an interface.
- We can create a reference variable to an interface.
- An interface cannot implement another interface.
- An interface can extend another interface.
- A class can implement multiple interfaces.

Program 1: Write an example program for interface
interface Shape

```
{    void area ();
    void volume ();
    double pi = 3.14;
}
class Circle implements Shape
{    double r;
    Circle (double radius)
    {        r = radius;
    }
    public void area ()
    {        System.out.println ("Area of a circle is : " + pi*r*r );
    }
    public void volume ()
    {        System.out.println ("Volume of a circle is : " + 2*pi*r);
    }
}
class Rectangle implements Shape
{    double l,b;
    Rectangle (double length, double breadth)
    {        l = length;
        b = breadth;
```

```

    }
    public void area ()
    {
        System.out.println ("Area of a Rectangle is : " + l*b );
    }
    public void volume ()
    {
        System.out.println ("Volume of a Rectangle is : " + 2*(l+b));
    }
}
class InterfaceDemo
{
    public static void main (String args[])
    {
        Circle ob1 = new Circle (10.2);
        ob1.area ();
        ob1.volume ();
        Rectangle ob2 = new Rectangle (12.6, 23.55);
        ob2.area ();
        ob2.volume ();
    }
}

```

Output:

```

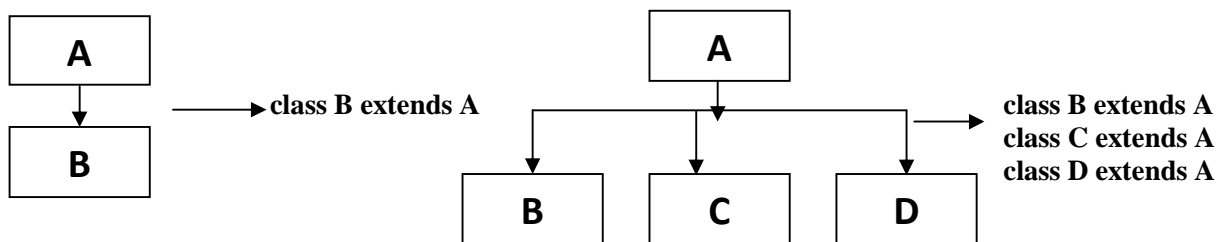
C:\WINDOWS\system32\cmd.exe
D:\JQR>javac InterfaceDemo.java

D:\JQR>java InterfaceDemo
Area of a circle is : 326.68559999999997
Volume of a circle is : 64.056
Area of a Rectangle is : 296.73
Volume of a Rectangle is : 72.3
D:\JQR>

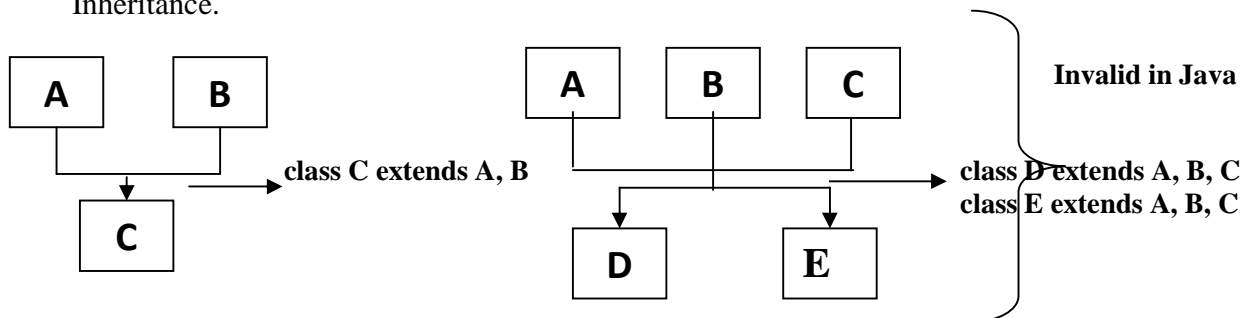
```

Types of inheritance:

- **Single Inheritance:** Producing subclass from a single super class is called single inheritance.



- **Multiple Inheritance:** Producing subclass from more than one super class is called Multiple Inheritance.



Java does not support multiple inheritance. But multiple inheritance can be achieved by using interfaces.

Program 2: Write a program to illustrate how to achieve multiple inheritance using multiple interfaces.

```
//interface Demo
interface Father
{
    double PROPERTY = 10000;
    double HEIGHT = 5.6;
}
interface Mother
{
    double PROPERTY = 30000;
    double HEIGHT = 5.4;
}
class MyClass implements Father, Mother
{
    void show()
    {
        System.out.println("Total property is :" +(Father.PROPERTY+Mother.PROPERTY));
        System.out.println ("Average height is :" + (Father.HEIGHT + Mother.HEIGHT)/2 );
    }
}
class InterfaceDemo
{
    public static void main(String args[])
    {
        MyClass ob1 = new MyClass();
        ob1.show();
    }
}
```

Output:



```
C:\WINDOWS\system32\cmd.exe
D:\JQR>javac InterfaceDemo.java
D:\JQR>java InterfaceDemo
Total property is :40000.0
Average height is :5.5
D:\JQR>
```

13. Packages

A package is a container of classes and interfaces. A package represents a directory that contains related group of classes and interfaces. For example, when we write statements like:

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

Here we are importing classes of java.io package. Here, java is a directory name and io is another sub directory within it. The '*' represents all the classes and interfaces of that io sub directory. We can create our own packages called user-defined packages or extend the available packages. User-defined packages can also be imported into other classes and used exactly in the same way as the Built-in packages. Packages provide reusability.

General form for creating a package:

```
package packagename;
```

e.g.: package pack;

- The first statement in the program must be package statement while creating a package.
- While creating a package except instance variables, declare all the members and the class itself as public then only the public members are available outside the package to other programs.

Program 1: Write a program to create a package pack with Addition class.

```
//creating a package
```

```
package pack;
```

```
public class Addition
```

```
{
    private double d1,d2;
    public Addition(double a,double b)
    {
        d1 = a;
        d2 = b;
    }
    public void sum()
    {
        System.out.println ("Sum of two given numbers is : " + (d1+d2) );
    }
}
```

Compiling the above program:



The -d option tells the Java compiler to create a separate directory and place the .class file in that directory (package). The (.) dot after -d indicates that the package should be created in the current directory. So, our package pack with Addition class is ready.

Program 2: Write a program to use the Addition class of package pack.

```
//Using the package pack
```

```
import pack.Addition;
```

```
class Use
```

```

{
    public static void main(String args[])
    {
        Addition ob1 = new Addition(10,20);
        ob1.sum();
    }
}

```

Output:

```

C:\WINDOWS\system32\cmd.exe
D:\JQR>javac Use.java
D:\JQR>java Use
Sum of two given numbers is : 30.0
D:\JQR>

```

Program 3: Write a program to add one more class Subtraction to the same package pack.

//Adding one more class to package pack:

```

package pack;
public class Subtraction
{
    private double d1,d2;
    public Subtraction(double a, double b)
    {
        d1 = a;
        d2 = b;
    }
    public void difference()
    {
        System.out.println ("Sum of two given numbers is : " + (d1 - d2) );
    }
}

```

Compiling the above program:

```

C:\WINDOWS\system32\cmd.exe
D:\JQR>javac -d . Subtraction.java
D:\JQR>

```

Program 4: Write a program to access all the classes in the package pack.

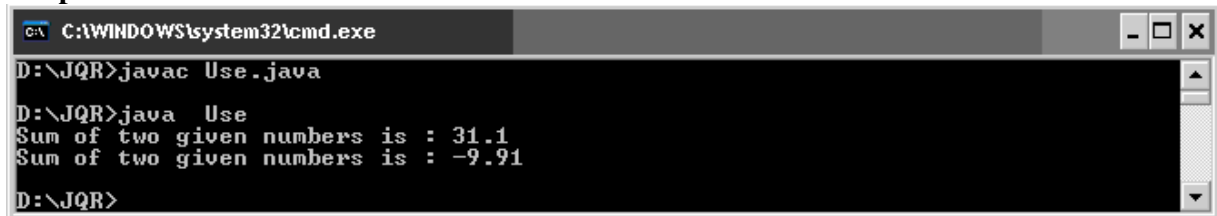
//To import all the classes and interfaces in a class using import pack.*;

```

import pack.*;
class Use
{
    public static void main(String args[])
    {
        Addition ob1 = new Addition(10.5,20.6);
        ob1.sum();
        Subtraction ob2 = new Subtraction(30.2,40.11);
        ob2.difference();
    }
}

```

In this case, please be sure that any of the Addition.java and Subtraction.java programs will not exist in the current directory. Delete them from the current directory as they cause confusion for the Java compiler. The compiler looks for byte code in Addition.java and Subtraction.java files and there it gets no byte code and hence it flags some errors.

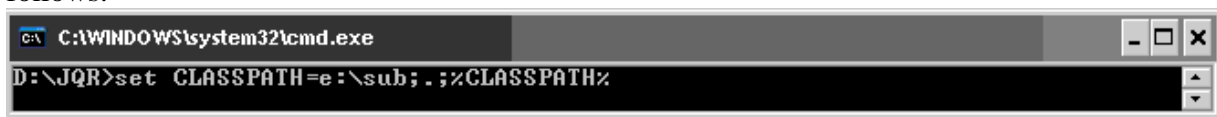
Output:


```

C:\WINDOWS\system32\cmd.exe
D:\JQR>javac Use.java
D:\JQR>java Use
Sum of two given numbers is : 31.1
Sum of two given numbers is : -9.91
D:\JQR>

```

If the package pack is available in different directory, in that case the compiler should be given information regarding the package location by mentioning the directory name of the package in the classpath. The CLASSPATH is an environment variable that tells the Java compiler where to look for class files to import. If our package exists in e:\sub then we need to set class path as follows:



```

C:\WINDOWS\system32\cmd.exe
D:\JQR>set CLASSPATH=e:\sub;.;%CLASSPATH%
D:\JQR>

```

We are setting the classpath to **e:\sub** directory and current directory (.) and %CLASSPATH% means retain the already available classpath as it is.

Creating Sub package in a package: We can create sub package in a package in the format:

package packagename.subpackagename;

e.g.: package pack1.pack2;

Here, we are creating pack2 subpackage which is created inside pack1 package. To use the classes and interfaces of pack2, we can write import statement as:

import pack1.pack2;

Program 5: Program to show how to create a subpackage in a package.

//Creating a subpackage in a package

package pack1.pack2;

public class Sample

{ public void show ()

{

System.out.println ("Hello Java Learners");

}

}

Compiling the above program:



```

C:\WINDOWS\system32\cmd.exe
D:\JQR>javac -d . Sample.java
D:\JQR>

```

Access Specifier: Specifies the scope of the data members, class and methods.

- private members of the class are available with in the class only. The scope of private members of the class is "CLASS SCOPE".

- public members of the class are available anywhere . The scope of public members of the class is "GLOBAL SCOPE".
- default members of the class are available with in the class, outside the class and in its sub class of same package. It is not available outside the package. So the scope of default members of the class is "PACKAGE SCOPE".
- protected members of the class are available with in the class, outside the class and in its sub class of same package and also available to subclasses in different package also.

Class Member Access	private	No Modifier	protected	public
Same class	Yes	Yes	Yes	Yes
Same package subclass	No	Yes	Yes	Yes
Same package non-subclass	No	Yes	Yes	Yes
Different package subclass	No	No	Yes	Yes
Different package non-subclass	No	No	No	Yes

Program 6: Write a program to create class A with different access specifiers.

```
//create a package same
package same;
public class A
{
    private int a=1;
    public int b = 2;
    protected int c = 3;
    int d = 4;
}
```


Compiling the above program:



```
C:\WINDOWS\system32\cmd.exe
D:\JQR>javac -d . A.java
D:\JQR>
```

Program 7: Write a program for creating class B in the same package.

```
//class B of same package
package same;
import same.A;
public class B
{
    public static void main(String args[])
    {
        A obj = new A();
        System.out.println(obj.a);
        System.out.println(obj.b);
        System.out.println(obj.c);
        System.out.println(obj.d);
    }
}
```

Compiling the above program:

```
C:\WINDOWS\system32\cmd.exe
D:\JQR>javac -d . B.java
B.java:7: a has private access in same.A
        System.out.println(obj.a);
                        ^
1 error
D:\JQR>
```

Program 8: Write a program for creating class C of another package.

package another;

import same.A;

public class C extends A

{ public static void main(String args[])

{ C obj = new C();

System.out.println(obj.a);

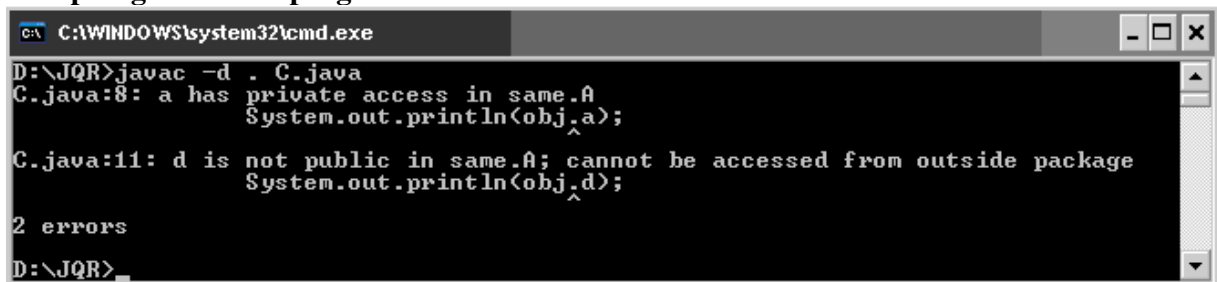
System.out.println(obj.b);

System.out.println(obj.c);

System.out.println(obj.d);

}

}

Compiling the above program:

```
C:\WINDOWS\system32\cmd.exe
D:\JQR>javac -d . C.java
C.java:8: a has private access in same.A
        System.out.println(obj.a);
                        ^
C.java:11: d is not public in same.A; cannot be accessed from outside package
        System.out.println(obj.d);
                        ^
2 errors
D:\JQR>
```

14. Exceptions

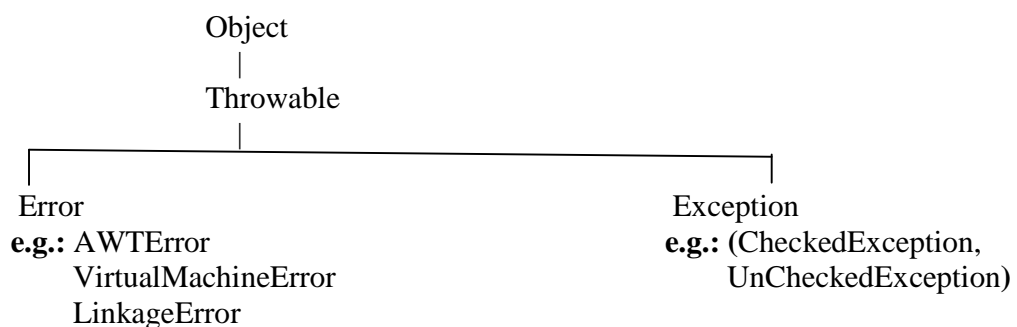
An error in a program is called bug. Removing errors from program is called debugging. There are basically three types of errors in the Java program:

- **Compile time errors:** Errors which occur due to syntax or format is called compile time errors. These errors are detected by java compiler at compilation time. Desk checking is solution for compile-time errors.
- **Runtime errors:** These are the errors that represent computer inefficiency. Insufficient memory to store data or inability of the microprocessor to execute some statement is examples to runtime errors. Runtime errors are detected by JVM at runtime.
- **Logical errors:** These are the errors that occur due to bad logic in the program. These errors are rectified by comparing the outputs of the program manually.

Exception: An abnormal event in a program is called Exception.

- Exception may occur at compile time or at runtime.
- Exceptions which occur at compile time are called **Checked exceptions**.
e.g.: ClassNotFoundException, NoSuchMethodException, NoSuchFieldException etc.
- Exceptions which occur at run time are called **Unchecked exceptions**.
eg: ArrayIndexOutOfBoundsException, ArithmeticException, NumberFormatException etc.

Exception Handling: Exceptions are represented as classes in java.



An exception can be handled by the programmer where as an error cannot be handled by the programmer. When there is an exception the programmer should do the following tasks:

- If the programmer suspects any exception in program statements, he should write them inside try block.

```

try
{
    statements;
}
  
```

- When there is an exception in try block JVM will not terminate the program abnormally. JVM stores exception details in an exception stack and then JVM jumps into catch block. The programmer should display exception details and any message to the user in catch block.

```

        catch ( ExceptionClass obj)
        {
            statements;
        }

```

- Programmer should close all the files and databases by writing them inside finally block. Finally block is executed whether there is an exception or not.

```

        finally
        {
            statements;
        }

```

- Performing above tasks is called Exception Handling.

Program 1: Write a program which tells the use of try, catch and finally block.

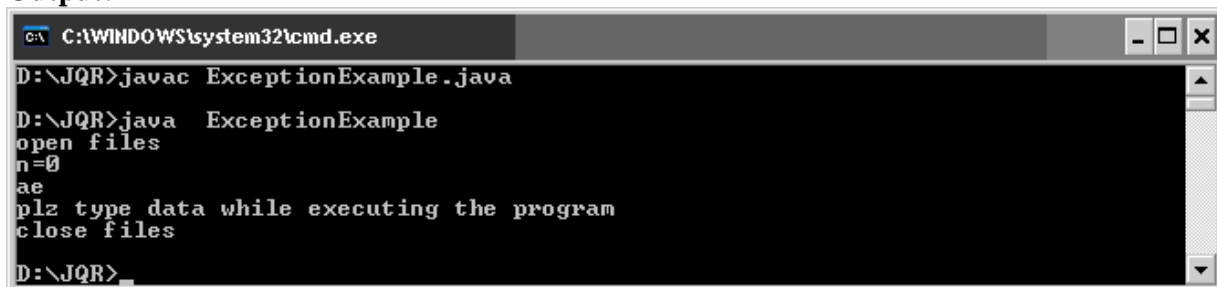
// Exception example

```

class ExceptionExample
{
    public static void main(String args[])
    {
        try
        {
            System.out.println ("open files");
            int n=args.length;
            System.out.println ("n="+n);
            int a=45/n;
            System.out.println ("a="+a);
            int b[]={ 10,19,12,13 };
            b[50]=100;
        }
        catch (ArithmeticException ae)
        {
            System.out.println ("ae");
            System.out.println ("plz type data while executing the program");
        }
        catch (ArrayIndexOutOfBoundsException aie)
        {
            System.out.println ("aie");
            System.out.println ("please see that array index is not within the range");
        }
        finally
        {
            System.out.println ("close files");
        }
    }
}

```

Output:



```

C:\WINDOWS\system32\cmd.exe
D:\JQR>javac ExceptionExample.java
D:\JQR>java ExceptionExample
open files
n=0
ae
plz type data while executing the program
close files
D:\JQR>

```

- Even though multiple exceptions are found in the program, only one exception is raised at a time.
- We can handle multiple exceptions by writing multiple catch blocks.
- A single try block can be followed by several catch blocks.
- Catch block does not always exit without a try, but a try block exit without a catch block.
- Finally block is always executed whether there is an exception or not.

throws Clause: throws clause is useful to escape from handling an exception. throws clause is useful to throw out any exception without handling it.

Program 2: Write a program which shows the use of throws clause.

// not handling the exception

import java.io.*;

class Sample

```
{
    void accept() throws IOException
    {
        BufferedReader br=new BufferedReader (new InputStreamReader(System.in));
        System.out.print ("enter ur name: ");
        String name=br.readLine ( );
        System.out.println ("Hai "+name);
    }
}
```

class ExceptionNotHandle

```
{
    public static void main (String args[])throws IOException
    {
        Sample s=new Sample ( );
        s.accept ( );
    }
}
```

Output:



```
C:\WINDOWS\system32\cmd.exe
D:\JQR>javac ExceptionNotHandle.java
D:\JQR>java ExceptionNotHandle
enter ur name: Ravi Chandra
Hai Ravi Chandra
D:\JQR>
```

throw Clause: throw clause can be used to throw out user defined exceptions. It is useful to create an exception object and throw it out of try block

Program 3: Write a program which shows the use of throw clause.

//Throw Example

class ThrowDemo

```
{
    static void Demo()
    {
        try
        {
            System.out.println ("inside method");
            throw new NullPointerException("my data");
        }
    }
}
```

```

    }
    catch (NullPointerException ne)
    {
        System.out.println ("ne");
    }
}
public static void main(String args[])
{
    ThrowDemo.Demo ( );
}
}

```

Output:

```

C:\WINDOWS\system32\cmd.exe
D:\JQR>javac ThrowDemo.java
D:\JQR>java ThrowDemo
inside method
ne
D:\JQR>

```

Types of Exceptions:

- **Built-in exceptions:** These are the exceptions which are already available in java.
e.g.: ArithmeticException, ArrayIndexOutOfBoundsException, NullPointerException, StringIndexOutOfBoundsException, NoSuchMethodException, InterruptedException, ClassNotFoundException, FileNotFoundException, NumberFormatException, RuntimeException etc.
- **User-defined exceptions:** - These are the exceptions created by the programmer.

Creating user defined exceptions:

- Write user exception class extending Exception class.
e.g.: class MyException extends Exception
- Write a default constructor in the user exception class
e.g.: MyException () { }
- Write a parameterized constructor with String as a parameter, from there call the parameterized constructor of Exception class.
e.g.: MyException (String str)

```

{
    super (str);
}

```
- Whenever required create user exception object and throw it using throw statement.
Ex: - throw me;

Program 4: Write a program to throw a user defined exception.

```

// user defined exception
class MyException extends Exception
{
    int accno[] = {1001,1002,1003,1004,1005};
}

```

```

String name[] = {"Hari","Siva","Bhanu","Rama","Chandu"};
double bal[] = {2500,3500,1500,1000,6000};
MyException()
{
}
MyException(String str)
{
    super(str);
}
public static void main(String args[])
{
    try
    {
        MyException me = new MyException("");
        System.out.println("AccNo \t Name \t Balance ");
        for(int i=0;i<5;i++)
        {
            System.out.println(me.accno[i]+ "\t" + me.name[i] + "\t" +
                               me.bal[i] );

            if( me.bal[i] < 2000 )
            {
                MyException me1 = new MyException
                    ("Insufficient Balance");

                throw me1;
            }
        }
    }
    catch(MyException e)
    {
        e.printStackTrace();
    }
}

```

Output:

```

C:\WINDOWS\system32\cmd.exe
D:\JQR>java MyException
AccNo    Name    Balance
1001     Hari    2500.0
1002     Siva    3500.0
1003     Bhanu    1500.0
MyException: Insufficient Balance
        at MyException.main(MyException.java:26)
D:\JQR>

```

15. Wrapper Classes

Wrapper Classes are used to convert primitive data types into objects. A wrapper class is a class whose object wraps the primitive data type. Wrapper classes are available in java.lang package. Different applications on internet send data or receive data in the form of objects. The classes in java.util package act upon objects only.

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

Character Class: The Character class wraps a value of the primitive type char in an object. An object of type character contains a single field whose type is char. We can create Character class object as: `Character obj = new Character (ch);` // where ch is a character.

Methods of Character Class:

Method	Description
char charValue ()	returns the char value of the invoking object.
int compareTo (Character obj)	This method is useful to compare the contents of two Character class objects.
static boolean isDigit (char ch)	returns true if ch is a digit (0 to 9) otherwise returns false.
static boolean isLetter (char ch)	returns true if ch is a letter (A to Z or a to z)
static boolean isUpperCase (char ch)	returns true if ch is an uppercase letter (A to Z)
static boolean isLowerCase (char ch)	returns true if ch is a lower case letter (a to z)
static boolean isSpaceChar (char ch)	returns true if ch is coming from SPACEBAR
static boolean isWhiteSpace(char ch)	returns true if ch is coming from TAB, ENTER, BackSpace
static char toUpperCase (char ch)	converts ch into uppercase
static char toLowerCase (char ch)	converts ch into lowercase

Program 1: Write a program which shows the use of Character class methods.

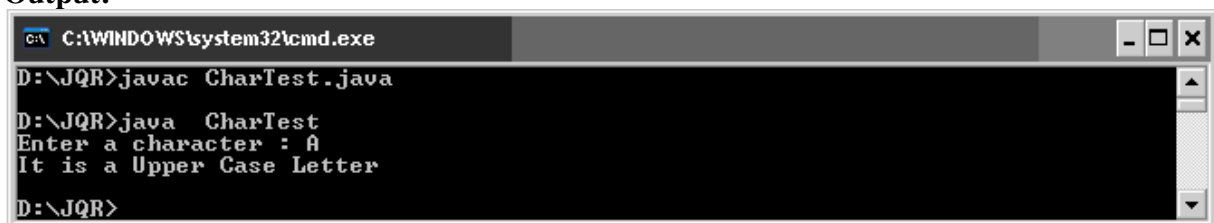
```
//Testing a char
import java.io.*;
class CharTest
{
    public static void main(String args[]) throws IOException
    {
        BufferedReader br = new BufferedReader (new InputStreamReader (System.in));
        System.out.print ("Enter a character : " );
        char ch = (char) br.read();
    }
}
```



```

        if (Character.isDigit (ch) )
            System.out.println ("It is a digit");
        else if (Character.isUpperCase (ch))
            System.out.println ("It is a Upper Case Letter");
        else if (Character.isLowerCase (ch) )
            System.out.println ("It is a Lower Case Letter");
        else if ( Character.isSpaceChar (ch))
            System.out.println ("It is a Space bar");
        else if ( Character.isWhitespace (ch) )
            System.out.println ("Dont know what is this character");
    }
}

```

Output:


```

C:\WINDOWS\system32\cmd.exe
D:\JQR>javac CharTest.java
D:\JQR>java CharTest
Enter a character : A
It is a Upper Case Letter
D:\JQR>

```

Byte Class: The Byte class wraps a value of primitive type 'byte' in an object. An object of type Byte contains a single field whose type is byte.

Constructors:

- Byte (byte num)
e.g.: Byte b1 = new Byte (98);
- Byte (String str)
e.g.: String str = "98";
Byte b2 = new Byte (str);

Methods of Byte Class:

Method	Description
byte byteValue ()	returns the value of invoking object as a byte.
int compareTo (Byte b)	This method is useful to compare the contents of two Byte class objects.
static byte parseByte(String str)	returns byte equivalent of the number contained in the string specified by 'str'.
String toString ()	returns a String that contains the decimal equivalent of the invoking object.
static Byte valueOf (String str)	returns a Byte object that contains the value specified by the String 'str'.

Program 2: Write a program which shows the use of Byte class methods.

//Creating and comparing Byte Objects

```

import java.io.*;
class Bytes
{
    public static void main(String args[]) throws IOException

```

```

{
    BufferedReader br = new BufferedReader (new InputStreamReader (System.in));
    System.out.print ("Enter a byte number: ");
    String str = br.readLine ();
    //convert str into Byte object
    Byte b1 = new Byte (str);
    System.out.print ("Enter a another byte number: ");
    str = br.readLine ();
    //convert str into Byte obj
    Byte b2 = Byte.valueOf (str);
    //compare b1 and b2
    int n = b1.compareTo(b2);
    if (n==0)
        System.out.println ("Both are Same");
    else if(n>0)
        System.out.println (b1 + "is bigger");
    else
        System.out.println (b2 + " is bigger");
}
}

```

Output:

```

C:\WINDOWS\system32\cmd.exe
D:\JQR>javac Bytes.java
D:\JQR>java Bytes
Enter a byte number: 10
Enter a another byte number: 35
35 is bigger
D:\JQR>

```

Short Class: Short class wraps a value of primitive data type 'short' in its object. Short class object contains a short type field that stores a short number.

Constructors:

- Short (short num)
- Short (String str)

Methods:

Method	Description
int compareTo (Short obj)	This method compares the numerical value of two Short class objects and returns 0,-ve, +ve value.
Boolean equals (Object obj)	This method compares the Short object with any other object obj and returns true if both have same content.
static short parseShort (String str)	This method returns int equivalent of the String str.
String toString ()	This method returns a String form of the Short object
static Short valueOf (String str)	This method converts a String str that contains a short number into Short class object and returns that object.

Integer Class: Integer class wraps a value of the primitive type 'int' in an object. An object of type Integer contains a single field whose type is int.

Constructors:

- Integer (int num)
- Integer (String str)

Methods:

Method	Description
int intValue ()	returns the value of the invoking object as an int.
int compareTo (Integer obj)	compares the numerical value of the invoking object with that of 'obj' returns zero or -ve value or +ve value.
static int parseInt (String str)	returns int equivalent of the String str.
String toString ()	returns a String form of the invoking object.
static Integer valueOf (String str)	returns an Integer object that contains the value shown by str.
static String toBinaryString (int i)	returns a String representation of the integer argument in base 2.
static String toHexString (int i)	returns a String representation of the integer argument in base 16.
static String toOctalString (int i)	returns a String representation of the integer argument in base 8.

Float Class: Float class wraps a value of primitive type float in an object. An object of type float contains a single field whose type is float.

Constructors:

- Float (float num)
- Float (String str)

Methods:

Method	Description
float floatValue ()	returns the value of the invoking object as a float
double doubleValue ()	returns the value of the invoking object as a double
int compareTo (Float f)	Compares the numerical value of the invoking object with that of 'f' returns zero or +ve or -ve value.
static float parseFloat (String str)	returns the float equivalent of the String str.
String toString ()	returns the String equivalent of invoking object
static Float valueOf (String str)	returns the Float object with the value specified by String str.

Long Class: The Long class contains a primitive long type data. The object of Long class contains a field where we can store a long value.

Constructors: Long has two constructors.

- Long (long num): Long object can be created as: Long obj = new Long (123000);
- Long (String str): String str = "12300044";
Long obj = new Long (str);

Methods:

Method	Description
int compareTo(Long obj)	This method compares the numerical value of two Long class objects and returns),-ve,+ve value.
static long parseLong(String str)	This method returns long equivalent of the String str.
String toString()	This method converts Long object into String object and returns the String object.
Static Long valueOf(String str)	This method converts a string str that contains some long number into Long object and returns that object.

Boolean class: The Boolean class object contains a primitive 'boolean' type data. The object of Boolean class contains a field where we can store a boolean value.

Constructors:

- Boolean obj = new Boolean (true);
- String str ="false";
Boolean obj = new Boolean (str);

Methods:

Method	Description
int compareTo(Boolean obj)	This method compares the numerical value of two Boolean class objects and returns 0,-ve,+ve value.
static boolean parseBoolean(String str)	This method returns boolean equivalent of the String str.
String toString()	This method converts Boolean object into a String object and returns the String object
static Boolean valueOf(String str)	This method converts a String str that contains a boolean value into Boolean object and returns that object.

Double Class: Double class wraps a value of primitive type Double in an Object.

Constructors:

- Double (double num)
- Double (String str)

Methods:

Method	Description
double doubleValue()	returns the value of the invoking object as a double
float floatValue()	returns the value of the invoking object as a float
int compareTo(Double d)	This method compares the numerical value of two Double class objects and returns 0,-ve,+ve value.
static double parseDouble(String str)	returns the double equivalent of the String str.
String toString()	This method converts Double object into a String object and returns the String object
static Double valueOf(String str)	returns the Double object with the value specified by String str.

Math class: The class Math contains methods for performing basic numeric operations.

Methods:

Method	Description
static double sin(double arg)	returns the sine value of the arg. arg is in radians.
static double cos(double arg)	returns the cosine value of the arg.
static double tan(double arg)	returns the tangent value of the arg.
static double log(double arg)	returns the natural logarithm value of arg.
static double pow(double x, double n)	returns x to the power of n value.
static double sqrt(double arg)	returns the square root of arg.
static double abs(double arg)	returns the absolute value of arg.
static double ceil(double arg)	returns the smallest integer which is greater or equal to arg.
static double floor(double arg)	returns the greatest integer which is lower or equal to arg.
static double min(arg1,arg2)	returns the minimum of arg1 and arg2.
static double max(arg1,arg2)	returns the maximum of arg1 and arg2.
static long round(arg)	returns the rounded value of arg.
static double random()	returns a random number between 0 and 1.
static double toRadians(double angle)	converts angle in degrees into radians.
static double toDegrees(double angle)	converts angle in radians into degrees.

Program 3: Write a program to print random numbers using Math class.

//Generating random numbers

class Rand

```
{
    public static void main(String args[]) throws Exception
    {
        while(true)
        {
            double d = 10 * Math.random();
            int i = (int) d;
            System.out.print ("\t" + i);
            if ( i == 0)
                System.exit (0);
        }
    }
}
```

Output:

```
C:\WINDOWS\system32\cmd.exe
D:\JQR>javac Rand.java
D:\JQR>java Rand
8 1 3 7 0
D:\JQR>
```

16. Generic Types

- Generic type represents classes, interfaces and methods in a type safe manner.
- Generic types can act on any type of data.
- All Generic types are subclasses of Object class, it acts on Objects only.
- Generic types act on advanced data type only.
- It is not possible to create an object to Generic type itself.
- Using generic types, we can avoid casting in many cases.

Generic Class: When we create a class with an instance variable to store an Integer object, it can be used to store Integer type data only. We cannot use that instance variable to store a Float class object or a String type Object. To store different types of data into a class, we have to write the same class again and again by changing the data type of the variables. This can be avoided using a generic class. A generic class represents a class that is type-safe. This means a generic class can act upon any data type. Generic classes and generic interfaces are also called ‘parameterized types’ because they use a parameter that determines which data type they should work upon.

Program 1: Write a program that has a class which stores any type of data.

//Example for Generic Class

```
class MyClass<T>
{
    T obj;
    MyClass (T obj)
    {
        this.obj = obj;
    }
    T getObj ()
    {
        return obj;
    }
}
class Gen1
{
    public static void main(String args[])
    {
        Integer i1 = new Integer (10);
        MyClass<Integer> obj1 = new MyClass<Integer>(i1);
        System.out.println ("U stored : " + obj1.getObj() );

        Double d1 = new Double(30.66);
        MyClass<Double> obj2 = new MyClass<Double>(d1);
        System.out.println ("U Stored : " + obj2.getObj() );

        MyClass<String> obj3 = new MyClass<String>("Suresh Kumar");
        System.out.println ("U Stored : " + obj3.getObj() );
    }
}
```

Output:

```

C:\WINDOWS\system32\cmd.exe
D:\JQR>javac Gen1.java
D:\JQR>java Gen1
U stored : 10
U Stored : 30.66
U Stored : Suresh Kumar
D:\JQR>

```

Generic Method: We can make a method alone as generic method by writing the generic parameter before the method return type as:

```

<T> returntype methodname ()
{
    Method code;
}

```

e.g.: `<T> void display_data ()`

```

{
    Method body;
}

```

Program 2: Write a program with generic method which displays any type of data.

//Generic method example

```

class MyClass
{
    <T>void display_data (T arr[])
    {
        for (int i=0;i<arr.length; i++)
            System.out.print ("\t" + arr[i]);
        System.out.println ();
    }
}

class Gen2
{
    public static void main(String args[])
    {
        MyClass obj = new MyClass ( );
        Integer a[] = { 1,2,3,4,5,6};
        System.out.print ("Reading Integer Objects: ");
        obj.display_data (a);

        Float b[] = { 1.1f,2.2f,3.4f};
        System.out.print ("Reading Float Objects: ");
        obj.display_data (b);

        String c[] = {"Subash","Chandra","Bose"};
        System.out.print ("Reading String Objects: ");
        obj.display_data (c);
    }
}

```

Output:

```

C:\WINDOWS\system32\cmd.exe
D:\JQR>javac Gen2.java
D:\JQR>java Gen2
Reading Integer Objects: 1 2 3 4 5 6
Reading Float Objects: 1.1 2.2 3.4
Reading String Objects: Subash Chandra Bose
D:\JQR>

```

Generic Interface: It is possible to develop an interface using generic type concept. The general form of generic interface looks like:

```

interface interface_name <T>
{
    //method that accepts any object
    return_type method_name ( T object_name );
}

```

Here, T represents any data type which is used in the interface. We can write an implementation class for the above interface as:

```

class class_name <T> implements interface_name <T>
{
    public return_type method_name ( T object_name )
    {
        //provide body of the method
    }
}

```

Program 3: Write an example program for generic interface.

```

//A generic interface
interface inter<T>
{
    void displayData (T obj);
}
class AnyClass<T> implements inter<T>
{
    public void displayData (T t1)
    {
        System.out.println ("Entered value is : " + t1);
    }
}
class Gen3
{
    public static void main (String args[])
    {
        AnyClass<Integer> ob1 = new AnyClass<Integer>();
        ob1.displayData (new Integer (10) );
        AnyClass<String> ob2 = new AnyClass<String>();
        ob2.displayData (new String ("Hari") );
    }
}

```


Output:

```
C:\WINDOWS\system32\cmd.exe
D:\JQR>javac Gen3.java
D:\JQR>java Gen3
Entered value is : 10
Entered value is : Hari
D:\JQR>
```

17. The Collection Framework

In order to handle group of objects we can use array of objects. If we have a class called Employ with members name and id, if we want to store details of 10 Employees, create an array of object to hold 10 Employ details.

```
Employ ob [] = new Employ [10];
```

- We cannot store different class objects into same array.
- Inserting element at the end of array is easy but at the middle is difficult.
- After retriving the elements from the array, in order to process the elements we dont have any methods

Collection Object:

- A collection object is an object which can store group of other objects.
- A collection object has a class called Collection class or Container class.
- All the collection classes are available in the package called 'java.util' (util stands for utility).
- Group of collection classes is called a Collection Framework.
- A collection object does not store the physical copies of other objects; it stores references of other objects.
- All the collection classes in java.util package are the implementation classes of different interfaces.

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<T>	HashMap<K,V> Hashtable<K,V>

- **Set:** A Set represents a group of elements (objects) arranged just like an array. The set will grow dynamically when the elements are stored into it. A set will not allow duplicate elements.
- **List:** Lists are like sets but allow duplicate values to be stored.
- **Queue:** A Queue represents arrangement of elements in FIFO (First In First Out) order. This means that an element that is stored as a first element into the queue will be removed first from the queue.
- **Map:** Maps store elements in the form of key value pairs. If the key is provided its corresponding value can be obtained.

Retrieving Elements from Collections: Following are the ways to retrieve any element from a collection object:

- Using Iterator interface.
- Using ListIterator interface.
- Using Enumeration interface.

Iterator Interface: Iterator is an interface that contains methods to retrieve the elements one by one from a collection object. It retrieves element only in forward direction. It has 3 methods:

Method	Description
boolean hasNext()	This method returns true if the iterator has more elements.
element next()	This method returns the next element in the iterator.
void remove()	This method removes the last element from the collection returned by the iterator.

ListIterator Interface: ListIterator is an interface that contains methods to retrieve the elements from a collection object, both in forward and reverse directions. It can retrieve the elements in forward and backward direction. It has the following important methods:

Method	Description
boolean hasNext()	This method returns true if the ListIterator has more elements when traversing the list in forward direction.
element next()	This method returns the next element.
void remove()	This method removes the last element that was returned by the next () or previous () methods.
boolean hasPrevious()	This method returns true if the ListIterator has more elements when traversing the list in reverse direction.
element previous()	This method returns the previous element in the list.

Enumeration Interface: This interface is useful to retrieve elements one by one like Iterator. It has 2 methods.

Method	Description
boolean hasMoreElements()	This method tests Enumeration has any more elements.
element nextElement()	This returns the next element that is available in Enumeration.

HashSet Class: HashSet represents a set of elements (objects). It does not guarantee the order of elements. Also it does not allow the duplicate elements to be stored.

- We can write the HashSet class as: `class HashSet<T>`
- We can create the object as: `HashSet<String> hs = new HashSet<String> ();`

The following constructors are available in HashSet:

- `HashSet();`
- `HashSet (int capacity);` Here capacity represents how many elements can be stored into the HashSet initially. This capacity may increase automatically when more number of elements is being stored.

HashSet Class Methods:

Method	Description
boolean add(obj)	This method adds an element obj to the HashSet. It returns true if the element is added to the HashSet, else it returns false. If the same

	element is already available in the HashSet, then the present element is not added.
boolean remove(obj)	This method removes the element obj from the HashSet, if it is present. It returns true if the element is removed successfully otherwise false.
void clear()	This removes all the elements from the HashSet
boolean contains(obj)	This returns true if the HashSet contains the specified element obj.
boolean isEmpty()	This returns true if the HashSet contains no elements.
int size()	This returns the number of elements present in the HashSet.

Program 1: Write a program which shows the use of HashSet and Iterator.

```
//HashSet Demo
import java.util.*;
class HS
{
    public static void main(String args[])
    {
        //create a HashSet to store Strings
        HashSet <String> hs = new HashSet<String> ();
        //Store some String elements
        hs.add ("India");
        hs.add ("America");
        hs.add ("Japan");
        hs.add ("China");
        hs.add ("America");
        //view the HashSet
        System.out.println ("HashSet = " + hs);
        //add an Iterator to hs
        Iterator it = hs.iterator ();
        //display element by element using Iterator
        System.out.println ("Elements Using Iterator: ");
        while (it.hasNext() )
        {
            String s = (String) it.next ();
            System.out.println(s);
        }
    }
}
```

Output:



```
C:\WINDOWS\system32\cmd.exe
D:\JQR>javac HS.java
D:\JQR>java HS
HashSet = [America, China, Japan, India]
Elements Using Iterator:
America
China
Japan
India
D:\JQR>
```

LinkedHashSet Class: This is a subclass of HashSet class and does not contain any additional members on its own. LinkedHashSet internally uses a linked list to store the elements. It is a generic class that has the declaration:

```
class LinkedHashSet<T>
```

Stack Class: A stack represents a group of elements stored in LIFO (Last In First Out) order. This means that the element which is stored as a last element into the stack will be the first element to be removed from the stack. Inserting the elements (Objects) into the stack is called push operation and removing the elements from stack is called pop operation. Searching for an element in stack is called peep operation. Insertion and deletion of elements take place only from one side of the stack, called top of the stack. We can write a Stack class as:

```
class Stack<E>
```

e.g.: Stack<Integer> obj = new Stack<Integer> ();

Stack Class Methods:

Method	Description
boolean empty()	this method tests whether the stack is empty or not. If the stack is empty then true is returned otherwise false.
element peek()	this method returns the top most object from the stack without removing it.
element pop()	this method pops the top-most element from the stack and returns it.
element push(element obj)	this method pushes an element obj onto the top of the stack and returns that element.
int search(Object obj)	This method returns the position of an element obj from the top of the stack. If the element (object) is not found in the stack then it returns -1.

Program 2: Write a program to perform different operations on a stack.

//pushing, popping, searching elements in a stack

```
import java.util.*;
```

```
class StackDemo
```

```
{
```

```
    public static void main(String args[])
```

```
    {
```

```
        //create an empty stack to contain Integer objects
```

```
        Stack<Integer> st = new Stack<Integer>();
```

```
        st.push (new Integer(10) );
```

```
        st.push (new Integer(20) );
```

```
        st.push (new Integer(30) );
```

```
        st.push (new Integer(40) );
```

```
        st.push (new Integer(50) );
```

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

```
        System.out.println ("Element at top of the stack is : " + st.peek() );
```

```
        System.out.println ("Removing element at the TOP of the stack : " + st.pop());
```

```
        System.out.println ("The new stack is : " + st);
```

```
    }
```

```
}
```

Output:

```

C:\WINDOWS\system32\cmd.exe
D:\JQR>javac StackDemo.java
D:\JQR>java StackDemo
[10, 20, 30, 40, 50]
Element at top of the stack is : 50
Removing element at the TOP of the stack : 50
The new stack is : [10, 20, 30, 40]
D:\JQR>

```

LinkedList Class: A linked list contains a group of elements in the form of nodes. Each node will have three fields- the data field contains data and the link fields contain references to previous and next nodes. A linked list is written in the form of:

```
class LinkedList<E>
```

we can create an empty linked list for storing String type elements (objects) as:

```
LinkedList <String> ll = new LinkedList<String> ();
```

LinkedList Class methods:

Method	Description
boolean add (element obj)	This method adds an element to the linked list. It returns true if the element is added successfully.
void add(int position, element obj)	This method inserts an element obj into the linked list at a specified position.
void addFirst(element obj)	This method adds the element obj at the first position of the linked list.
void addLast(element obj)	This method adds the element obj at the last position of the linked list.
element removeFirst ()	This method removes the first element from the linked list and returns it.
element removeLast ()	This method removes the last element from the linked list and returns it.
element remove (int position)	This method removes an element at the specified position in the linked list.
void clear ()	This method removes all the elements from the linked list.
element get (int position)	This method returns the element at the specified position in the linked list.
element getFirst ()	This method returns the first element from the list.
element getLast ()	This method returns the last element from the list.
element set(int position, element obj)	This method replaces the element at the specified position in the list with the specified element obj.
int size ()	Returns number of elements in the linked list.
int indexOf (Object obj)	This method returns the index of the first occurrence of the specified element in the list, or -1 if the list does not contain the element.
int lastIndexOf (Object obj)	This method returns the index of the last occurrence of the specified element in the list, or -1 if the list does not contain the element.

Object[] toArray()	This method converts the linked list into an array of Object class type. All the elements of the linked list will be stored into the array in the same sequence.
--------------------	--

Note: In case of LinkedList counting starts from 0 and we start counting from 1.

Program 3: Write a program that shows the use of LinkedList class.

```
import java.util.*;
//Linked List
class LinkedDemo
{
    public static void main(String args[])
    {
        LinkedList <String> ll = new LinkedList<String>();
        ll.add ("Asia");
        ll.add ("North America");
        ll.add ("South America");
        ll.add ("Africa");
        ll.addFirst ("Europe");
        ll.add (1,"Australia");
        ll.add (2,"Antarctica");
        System.out.println ("Elements in Linked List is : " + ll);
        System.out.println ("Size of the Linked List is : " + ll.size() );
    }
}
```

Output:

```
C:\WINDOWS\system32\cmd.exe
D:\JQR>javac LinkedDemo.java
D:\JQR>java LinkedDemo
Elements in Linked List is : [Europe, Australia, Antarctica, Asia, North America, South America, Africa]
Size of the Linked List is : 7
D:\JQR>
```

ArrayList Class: An ArrayList is like an array, which can grow in memory dynamically. ArrayList is not synchronized. This means that when more than one thread acts simultaneously on the ArrayList object, the results may be incorrect in some cases.

ArrayList class can be written as: `class ArrayList <E>`

We can create an object to ArrayList as: `ArrayList <String> arl = new ArrayList<String> ();`

ArrayList Class Methods:

Method	Description
boolean add (element obj)	This method appends the specified element to the end of the ArrayList. If the element is added successfully then the method returns true.
void add(int position, element obj)	This method inserts the specified element at the specified position in the ArrayList.
element remove(int position)	This method removes the element at the specified position in the ArrayList and returns it.
boolean remove (Object)	This method removes the first occurrence of the specified element

obj)	obj from the ArrayList, if it is present.
void clear ()	This method removes all the elements from the ArrayList.
element set(int position, element obj)	This method replaces an element at the specified position in the ArrayList with the specified element obj.
boolean contains (Object obj)	This method returns true if the ArrayList contains the specified element obj.
element get (int position)	This method returns the element available at the specified position in the ArrayList.
int size ()	Returns number of elements in the ArrayList.
int indexOf (Object obj)	This method returns the index of the first occurrence of the specified element in the list, or -1 if the list does not contain the element.
int lastIndexOf (Object obj)	This method returns the index of the last occurrence of the specified element in the list, or -1 if the list does not contain the element.
Object[] toArray ()	This method converts the ArrayList into an array of Object class type. All the elements of the ArrayList will be stored into the array in the same sequence.

Program 4: Write a program that shows the use of ArrayList class.

```
import java.util.*;
//ArrayList Demo
class ArrayListDemo
{
    public static void main(String args[])
    {
        ArrayList <String> al = new ArrayList<String>();
        al.add ("Asia");
        al.add ("North America");
        al.add ("South America");
        al.add ("Africa");
        al.add ("Europe");
        al.add (1,"Australia");
        al.add (2,"Antarctica");
        System.out.print ("Size of the Array List is: " + al.size ());
        System.out.print ("\nRetrieving elements in ArrayList using Iterator :");
        Iterator it = al.iterator ();
        while (it.hasNext () )
            System.out.print (it.next () + "\t");
    }
}
```

Output:

```
C:\WINDOWS\system32\cmd.exe
D:\JQR>javac ArrayListDemo.java
D:\JQR>java ArrayListDemo
Size of the Array List is: 7
Retrieving elements in ArrayList using Iterator :Asia    Australia    Antarcti
ca    North America    South America    Africa    Europe
D:\JQR>
```


Vector Class: Similar to ArrayList, but Vector is synchronized. It means even if several threads act on Vector object simultaneously, the results will be reliable.

Vector class can be written as: `class Vector <E>`

We can create an object to Vector as: `Vector <String> v = new Vector<String> ();`

Vector Class Methods:

Method	Description
<code>boolean add(element obj)</code>	This method appends the specified element to the end of the Vector. If the element is added successfully then the method returns true.
<code>void add (int position, element obj)</code>	This method inserts the specified element at the specified position in the Vector.
<code>element remove (int position)</code>	This method removes the element at the specified position in the Vector and returns it.
<code>boolean remove (Object obj)</code>	This method removes the first occurrence of the specified element obj from the Vector, if it is present.
<code>void clear ()</code>	This method removes all the elements from the Vector.
<code>element set (int position, element obj)</code>	This method replaces an element at the specified position in the Vector with the specified element obj.
<code>boolean contains (Object obj)</code>	This method returns true if the Vector contains the specified element obj.
<code>element get (int position)</code>	This method returns the element available at the specified position in the Vector.
<code>int size ()</code>	Returns number of elements in the Vector.
<code>int indexOf (Object obj)</code>	This method returns the index of the first occurrence of the specified element in the Vector, or -1 if the Vector does not contain the element.
<code>int lastIndexOf (Object obj)</code>	This method returns the index of the last occurrence of the specified element in the Vector, or -1 if the Vector does not contain the element.
<code>Object[] toArray ()</code>	This method converts the Vector into an array of Object class type. All the elements of the Vector will be stored into the array in the same sequence.
<code>int capacity ()</code>	This method returns the current capacity of the Vector.

Program 5: Write a program that shows the use of Vector class.

```
import java.util.*;
//Vector Demo
class VectorDemo
{
    public static void main(String args[])
    {
        Vector <Integer> v = new Vector<Integer> ();
        int x[] = {10,20,30,40,50};
        //When x[i] is stored into v below, x[i] values are converted into Integer Objects
        //and stored into v. This is auto boxing.
        for (int i = 0; i<x.length; i++)
            v.add(x[i]);
        System.out.println ("Getting Vector elements using get () method: ");
    }
}
```

```

        for (int i = 0; i < v.size(); i++)
            System.out.print (v.get (i) + "\t");
        System.out.println ("\nRetrieving elements in Vector using ListIterator :");
        ListIterator lit = v.listIterator ();
        while (lit.hasNext () )
            System.out.print (lit.next () + "\t");
        System.out.println ("\nRetrieving elements in reverse order using ListIterator :");
        while (lit.hasPrevious () )
            System.out.print (lit.previous () + "\t");
    }
}

```

Output:

```

C:\WINDOWS\system32\cmd.exe
D:\JQR>java VectorDemo
Getting Vector elements using get (<) method:
10 20 30 40 50
Retrieving elements in Vector using ListIterator :
10 20 30 40 50
Retrieving elements in reverse order using ListIterator :
50 40 30 20 10
D:\JQR>

```

HashMap Class: HashMap is a collection that stores elements in the form of key-value pairs. If key is provided later its corresponding value can be easily retrieved from the HashMap. Keys should be unique. HashMap is not synchronized and hence while using multiple threads on HashMap object, we get unreliable results.

We can write HashMap class as: `class HashMap<K, V>`

For example to store a String as key and an integer object as its value, we can create the HashMap as: `HashMap<String, Integer> hm = new HashMap<String, Integer> ();`

The default initial capacity of this HashMap will be taken as 16 and the load factor as 0.75. Load factor represents at what level the HashMap capacity should be doubled. For example, the product of capacity and load factor = $16 * 0.75 = 12$. This represents that after storing 12th key-value pair into the HashMap, its capacity will become 32.

HashMap Class Methods:

Method	Description
value put (key, value)	This method stores key-value pair into the HashMap.
value get (Object key)	This method returns the corresponding value when key is given. If the key does not have a value associated with it, then it returns null.
Set<K> keyset()	This method, when applied on a HashMap converts it into a set where only keys will be stored.
Collection<V> values()	This method, when applied on a HashMap object returns all the values of the HashMap into a Collection object.
value remove (Object key)	This method removes the key and corresponding value from the HashMap.
void clear ()	This method removes all the key-value pairs from the map.
boolean isEmpty ()	This method returns true if there are no key-value pairs in the HashMap.
int size ()	This method returns number of key-value pairs in the HashMap.

Program 6: Write a program that shows the use of HashMap class.

```
//HashMap Demo
import java.util.*;
class HashMapDemo
{
    public static void main(String args[])
    {
        HashMap<Integer, String> hm = new HashMap<Integer, String> ();
        hm.put (new Integer (101),"Naresh");
        hm.put (new Integer (102),"Rajesh");
        hm.put (new Integer (103),"Suresh");
        hm.put (new Integer (104),"Mahesh");
        hm.put (new Integer (105),"Ramesh");
        Set<Integer> set = new HashSet<Integer>();
        set = hm.keySet();
        System.out.println (set);
    }
}
```

Output:

```
C:\WINDOWS\system32\cmd.exe
D:\JQR>javac HashMapDemo.java
D:\JQR>java HashMapDemo
[102, 103, 101, 104, 105]
D:\JQR>
```

Hashtable Class: Hashtable is a collection that stores elements in the form of key-value pairs. If key is provided later its corresponding value can be easily retrieved from the Hashtable. Keys should be unique. Hashtable is synchronized and hence while using multiple threads on Hashtable object, we get reliable results.

We can write Hashtable class as: `class Hashtable<K,V>`

For example to store a String as key and an integer object as its value, we can create the Hashtable as: `Hashtable<String, Integer> ht = new Hashtable<String, Integer> ();`

The default initial capacity of this Hashtable will be taken as 11 and the load factor as 0.75. Load factor represents at what level the Hashtable capacity should be doubled. For example, the product of capacity and load factor = $11 * 0.75 = 8.25$. This represents that after storing 8th key-value pair into the Hashtable, its capacity will become 22.

Hashtable Class Methods:

Method	Description
value put(key, value)	This method stores key-value pair into the Hashtable.
value get(Object key)	This method returns the corresponding value when key is given. If the key does not have a value associated with it, then it returns null.
Set<K> keyset()	This method, when applied on a Hashtable converts it into a set where only keys will be stored.
Collection <V> values()	This method, when applied on a Hashtable object returns all the values of the Hashtable into a Collection object.
value remove(Object key)	This method removes the key and corresponding value from the Hashtable.

void clear()	This method removes all the key-value pairs from the Hashtable.
boolean isEmpty()	This method returns true if there are no key-value pairs in the Hashtable.
int size()	This method returns number of key-value pairs in the Hashtable.

Program 7: Write a program that shows the use of Hashtable class.

```
//Hashtable Demo
import java.util.*;
class HashtableDemo
{
    public static void main(String args[])
    {
        Hashtable<Integer, String> ht = new Hashtable<Integer, String> ();
        ht.put (new Integer (101),"Naresh");
        ht.put (new Integer (102),"Rajesh");
        ht.put (new Integer (103),"Suresh");
        ht.put (new Integer (104),"Mahesh");
        ht.put (new Integer (105),"Ramesh");
        Enumeration e = ht.keys ();
        while ( e.hasMoreElements () )
        {
            Integer i1 = (Integer) e.nextElement ();
            System.out.println (i1 + "\t" + ht.get (i1));
        }
    }
}
```

Output:

```
C:\WINDOWS\system32\cmd.exe
D:\JQR>javac HashtableDemo.java
D:\JQR>java HashtableDemo
105      Ramesh
104      Mahesh
103      Suresh
102      Rajesh
101      Naresh
D:\JQR>
```

Arrays Class: Arrays class provides methods to perform certain operations on any single dimensional array. All the methods of the Arrays class are static, so they can be called in the form of Arrays.methodname ().

Arrays Class Methods:

Method	Description
static void sort (array)	This method sorts all the elements of an array into ascending order. This method internally uses QuickSort algorithm.
static void sort (array, int start, int end)	This method sorts the elements in the range from start to end within an array into ascending order.

static int binarySearch (array, element)	This method searches for an element in the array and returns its position number. If the element is not found in the array, it returns a negative value. Note that this method acts only on an array which is sorted in ascending order. This method internally uses BinarySearch algorithm.
static boolean equals (array1, array2)	This method returns true if two arrays, that is array1 and array2 are equal, otherwise false.
static array copyOf (source-array, int n)	This method copies n elements from the source-array into another array and returns the array.
static void fill (array, value)	This method fills the array with the specified value. It means that all the elements in the array will receive that value.

Program 8: Write a program to sort given numbers using sort () method of Arrays Class.

```
import java.util.*;
//Arrays Demo
class ArraysDemo
{
    public static void main(String args[])
    {
        int x[] = {40,50,10,30,20};
        Arrays.sort( x );
        for (int i=0;i<x.length;i++)
            System.out.print(x[i] + "\t");
    }
}
```

Output:

```
C:\WINDOWS\system32\cmd.exe
D:\JQR>javac ArraysDemo.java
D:\JQR>java ArraysDemo
10    20    30    40    50
D:\JQR>
```

StringTokenizer: The StringTokenizer class is useful to break a String into small pieces called tokens. We can create an object to StringTokenizer as:

```
StringTokenizer st = new StringTokenizer (str, "delimiter");
```

StringTokenizer Class Methods:

Method	Description
String nextToken()	Returns the next token from the StringTokenizer
boolean hasMoreTokens()	Returns true if token is available and returns false if not available
int countTokens()	Returns the number of tokens available.

Program 9: Write a program that shows the use of StringTokenizer object.

```
//cutting the String into tokens
import java.util.*;
class STDemo
{
```

```

public static void main(String args[])
{
    //take a String
    String str = "Java is an OOP Language";
    //brake wherever a space is found
    StringTokenizer st = new StringTokenizer (str, " ");
    //retrieve tokens and display
    System.out.println ("The tokens are: ");
    while ( st.hasMoreTokens () )
    {
        String s = st.nextToken ();
        System.out.println (s );
    }
}

```

Output:

```

C:\WINDOWS\system32\cmd.exe
D:\JQR>javac STDemo.java
D:\JQR>java STDemo
The tokens are:
Java
is
an
OOP
Language
D:\JQR>

```

Calendar: This class is useful to handle date and time. We can create an object to Calendar class as:

```
Calendar cl = Calendar.getInstance ();
```

Calendar Class Methods:

Method	Description
int get(Constant)	This method returns the value of the given Calendar constant. Examples of Constants are Calendar.DATE, Calendar.MONTH, Calendar.YEAR, Calendar.MINUTE, Calendar.SECOND, Calendar.Hour
void set(int field, int value)	This method sets the given field in Calendar Object to the given value. For example, cl.set(Calendar.DATE,15);
String toString()	This method returns the String representation of the Calendar object.
boolean equals(Object obj)	This method compares the Calendar object with another object obj and returns true if they are same, otherwise false.

Program 10: Write a program to display System time and date.

//To display system time and date

```
import java.util.*;
```

```
class Cal
```

```
{
    public static void main(String args[])
    {
        Calendar cl = Calendar.getInstance ();
    }
}
```

```

        //Retrieve Date
        int dd = cl.get (Calendar.DATE);
        int mm = cl.get (Calendar.MONTH);
        ++mm;
        int yy = cl.get (Calendar.YEAR);
        System.out.println ("Current Date is : " + dd + "-" + mm + "-" + yy );
        //Retrieve Time
        int hh = cl.get (Calendar.HOUR);
        int mi = cl.get (Calendar.MINUTE);
        int ss = cl.get (Calendar.SECOND);
        System.out.println ("Current Time is : " + hh + ":" + mi + ":" +ss);
    }
}

```

Output:

```

C:\WINDOWS\system32\cmd.exe
D:\JQR>javac Cal.java
D:\JQR>java Cal
Current Date is : 9-10-2011
Current Time is : 4:31:42
D:\JQR>

```

Date Class: Date Class is also useful to handle date and time. Once Date class object is created, it should be formatted using the following methods of DateFormat class of java.text package. We can create an object to Date class as: `Date dd = new Date ();`
Once Date class object is created, it should be formatted using the methods of DateFormat class of java.text package.

DateFormat class Methods:

- `DateFormat fmt = DateFormat.getDateInstance(formatconst, region);`
This method is useful to store format information for date value into DateFormat object fmt.
- `DateFormat fmt = DateFormat.getTimeInstance(formatconst, region);`
This method is useful to store format information for time value into DateFormat object fmt.
- `DateFormat fmt = DateFormat.getDateTimeInstance(formatconst, formatconst, region);`
This method is useful to store format information for date value into DateFormat object fmt.

Formatconst	Example (region=Locale.UK)
DateFormat.FULL	03 september 2007 19:43:14 O'Clock GMT + 05:30
DateFormat.LONG	03 september 2007 19:43:14 GMT + 05:30
DateFormat.MEDIUM	03-sep-07 19:43:14
DateFormat.SHORT	03/09/07 19:43

Program 11: Write a program that shows the use of Date class.

```

//Display System date and time using Date class
import java.util.*;
import java.text.*;
class MyDate
{

```

```
public static void main(String args[])
{
    Date d = new Date ();
    DateFormat fmt = DateFormat.getDateInstance (DateFormat.MEDIUM,
                                                    DateFormat.SHORT, Locale.UK);

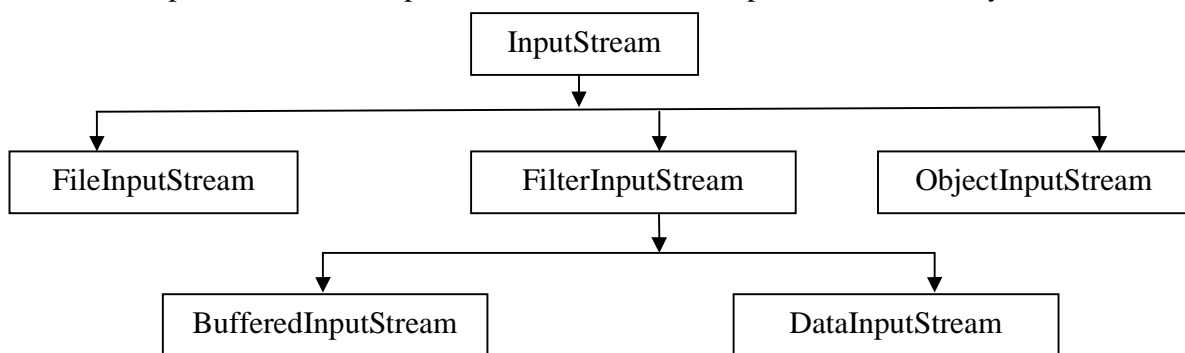
    String str = fmt.format (d);
    System.out.println (str);
}
}
```

Output:A screenshot of a Windows command prompt window. The title bar shows 'C:\WINDOWS\system32\cmd.exe'. The command prompt shows the following sequence of commands and output:
D:\JQR>javac MyDate.java
D:\JQR>java MyDate
09-Oct-2011 16:45
D:\JQR>

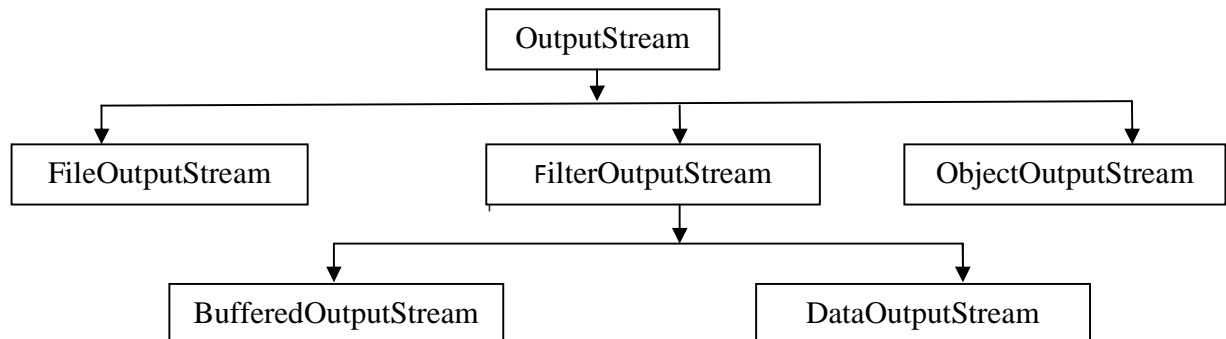
18. Streams and Files

A Stream represents flow of data from one place to another place. Input Streams reads or accepts data. Output Streams sends or writes data to some other place. All streams are represented as classes in java.io package. The main advantage of using stream concept is to achieve hardware independence. This is because we need not change the stream in our program even though we change the hardware. Streams are of two types in Java:

- **Byte Streams:** Handle data in the form of bits and bytes. Byte streams are used to handle any characters (text), images, audio and video files. For example, to store an image file (.gif or .jpg), we should go for a byte stream. To handle data in the form of 'bytes' the abstract classes: InputStream and OutputStream are used. The important classes of byte streams are:



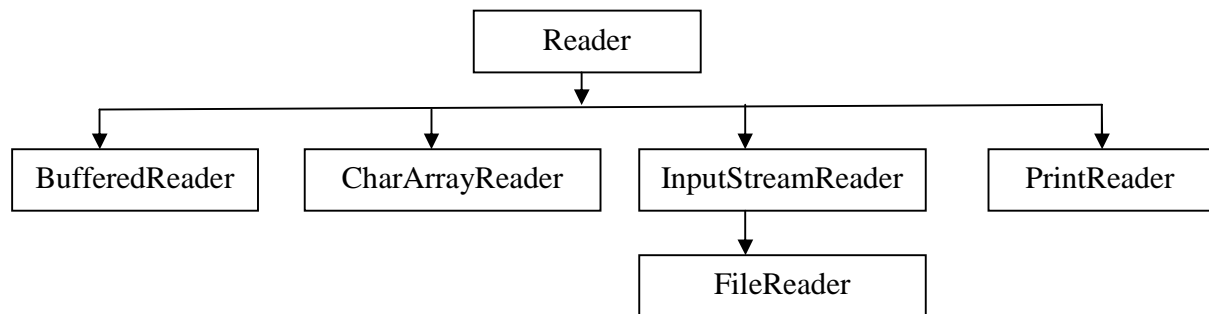
Byte Stream Classes for Reading Data



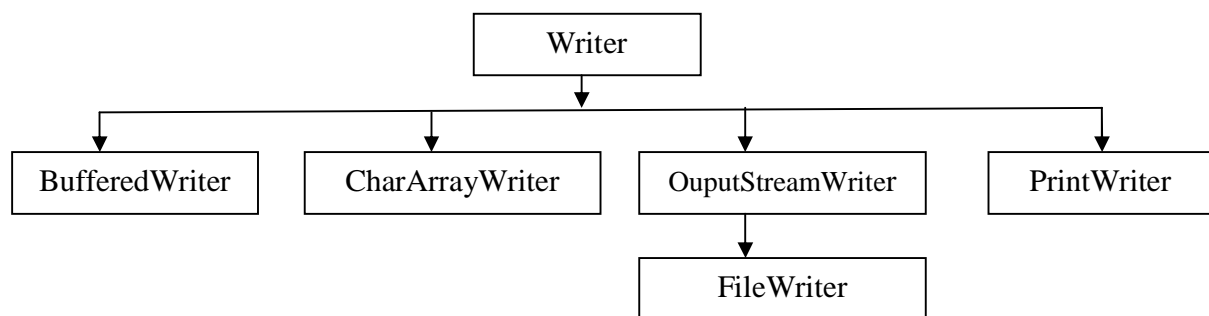
Byte Stream Classes for Writing Data

- FileInputStream/FileOutputStream: They handle data to be read or written to disk files.
- FilterInputStream/FilterOutputStream: They read data from one stream and write it to another stream.
- ObjectInputStream/ObjectOutputStream: They handle storage of objects and primitive data.
- **Character or Text Streams:** Handle data in the form of characters. Character or text streams can always store and retrieve data in the form of characters (or text) only. It means text streams are more suitable for handling text files like the ones we create in Notepad. They are not suitable to handle the images, audio or video files. To handle data in the form of 'text'

the abstract classes: Reader and Writer are used. The important classes of character streams are:



Text Stream Classes for Reading Data



Text Stream Classes for Writing Data

- **BufferedReader/BufferedWriter:** - Handles characters (text) by buffering them. They provide efficiency.
- **CharArrayReader/CharArrayWriter:** - Handles array of characters.
- **InputStreamReader/OutputStreamWriter:** - They are bridge between byte streams and character streams. Reader reads bytes and then decodes them into 16-bit unicode characters. Writer decodes characters into bytes and then writes.
- **PrintReader/PrintWriter:** - Handle printing of characters on the screen.

File: A file represents organized collection of data. Data is stored permanently in the file. Once data is stored in the form of a file we can use it in different programs.

Program 1: Write a program to read data from the keyboard and write it to a text file using byte stream classes.

//Creating a text file using byte stream classes

import java.io.*;

class Create1

```

{
    public static void main(String args[]) throws IOException
    {
        //attach keyboard to DataInputStream
        DataInputStream dis = new DataInputStream (System.in);
        //attach the file to FileOutputStream
        FileOutputStream fout = new FileOutputStream ("myfile");
        //read data from DataInputStream and write into FileOutputStream
        char ch;
    }
}
  
```

```

        System.out.println ("Enter @ at end : " );
        while( (ch = (char) dis.read() ) != '@' )
            fout.write (ch);
        fout.close ();
    }
}

```

Output:

```

C:\WINDOWS\system32\cmd.exe
D:\JQR>javac Create1.java
D:\JQR>java Create1
Enter @ at end :
I am writing first line
I am writing second line
@
D:\JQR>

```

Program 2: Write a program to improve the efficiency of writing data into a file using `BufferedOutputStream`.

//Creating a text file using byte stream classes

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

```
class Create2
```

```

{
    public static void main(String args[]) throws IOException
    {
        //attach keyboard to DataInputStream
        DataInputStream dis = new DataInputStream (System.in);
        //attach file to FileOutputStream, if we use true then it will open in append mode
        FileOutputStream fout = new FileOutputStream ("myfile", true);
        BufferedOutputStream bout = new BufferedOutputStream (fout, 1024);
        //Buffer size is declared as 1024 otherwise default buffer size of 512 bytes is used.
        //read data from DataInputStream and write into FileOutputStream
        char ch;
        System.out.println ("Enter @ at end : " );
        while ( (ch = (char) dis.read() ) != '@' )
            bout.write (ch);
        bout.close ();
        fout.close ();
    }
}

```

Output:

```

C:\WINDOWS\system32\cmd.exe
D:\JQR>javac Create2.java
D:\JQR>java Create2
Enter @ at end :
This is new line in my file
@

D:\JQR>type myfile
I am writing first line
I am writing second line
This is new line in my file
D:\JQR>

```

Program 3: Write a program to read data from *myfile* using `FileInputStream`.

//Reading a text file using byte stream classes

```
import java.io.*;
class Read1
{
    public static void main (String args[]) throws IOException
    {
        //attach the file to FileInputStream
        FileInputStream fin = new FileInputStream ("myfile");
        //read data from FileInputStream and display it on the monitor
        int ch;
        while ( (ch = fin.read() ) != -1 )
            System.out.print ((char) ch);
        fin.close ();
    }
}
```

Output:



```
C:\WINDOWS\system32\cmd.exe
D:\JQR>javac Read1.java
D:\JQR>java Read1
I am writing first line
I am writing second line
This is new line in my file
D:\JQR>
```

Program 4: Write a program to improve the efficiency while reading data from a file using `BufferedInputStream`.

//Reading a text file using byte stream classes

```
import java.io.*;
class Read2
{
    public static void main(String args[]) throws IOException
    {
        //attach the file to FileInputStream
        FileInputStream fin = new FileInputStream ("myfile");
        BufferedInputStream bin = new BufferedInputStream (fin);
        //read data from FileInputStream and display it on the monitor
        int ch;
        while ( (ch = bin.read() ) != -1 )
            System.out.print ( (char) ch);
        fin.close ();
    }
}
```

Output:



```
C:\WINDOWS\system32\cmd.exe
D:\JQR>javac Read2.java
D:\JQR>java Read2
I am writing first line
I am writing second line
This is new line in my file
D:\JQR>
```

Program 5: Write a program to create a text file using character or text stream classes.

//Creating a text file using character (text) stream classes

```
import java.io.*;
class Create3
{
    public static void main(String args[]) throws IOException
    {
        String str = "This is an Institute" + "\n You are a student";    // take a String
        //Connect a file to FileWriter
        FileWriter fw = new FileWriter ("textfile");
        //read chars from str and send to fw
        for (int i = 0; i<str.length () ; i++)
            fw.write (str.charAt (i) );
        fw.close ();
    }
}
```

Output:



```
C:\WINDOWS\system32\cmd.exe
D:\JQR>javac Create3.java
D:\JQR>java Create3
D:\JQR>type textfile
This is an Institute
You are a student
D:\JQR>
```

Program 6: Write a program to read a text file using character or text stream classes.

//Reading data from file using character (text) stream classes

```
import java.io.*;
class Read3
{
    public static void main(String args[]) throws IOException
    {
        //attach file to FileReader
        FileReader fr = new FileReader ("textfile");
        //read data from fr and display
        int ch;
        while ((ch = fr.read()) != -1)
            System.out.print ((char) ch);
        //close the file
        fr.close ();
    }
}
```

Output:



```
C:\WINDOWS\system32\cmd.exe
D:\JQR>javac Read3.java
D:\JQR>java Read3
This is an Institute
You are a student
D:\JQR>
```

Note: Use `BufferedReader` and `BufferedWriter` to improve the efficiency of the above two programs.

Serialization of objects:

- Serialization is the process of storing object contents into a file. The class whose objects are stored in the file should implement 'Serializable' interface of java.io package.
- Serializable interface is an empty interface without any members and methods, such an interface is called 'marking interface' or 'tagging interface'.
- Marking interface is useful to mark the objects of a class for a special purpose. For example, 'Serializable' interface marks the class objects as 'serializable' so that they can be written into a file. If serializable interface is not implemented by the class, then writing that class objects into a file will lead to NotSerializableException.
- static and transient variables cannot be serialized.
- De-serialization is the process of reading back the objects from a file.

Program 7: Write a program to create Employ class whose objects is to be stored into a file.

```
//Employ information
import java.io.*;
import java.util.*;
class Employ implements Serializable
{
    private int id;
    private String name;
    private float sal;
    private Date doj;
    Employ (int i, String n, float s, Date d)
    {
        id = i;
        name = n;
        sal = s;
        doj = d;
    }
    void display ()
    {
        System.out.println (id+ "\t" + name + "\t" + sal + "\t" + doj);
    }
    static Employ getData() throws IOException
    {
        BufferedReader br = new BufferedReader (new InputStreamReader (System.in));
        System.out.print ("Enter employ id : ");
        int id = Integer.parseInt(br.readLine());
        System.out.print ("Enter employ name : ");
        String name = br.readLine ();
        System.out.print ("Enter employ salary : " );
        float sal = Float.parseFloat(br.readLine ());
        Date d = new Date ();
        Employ e = new Employ (id, name, sal, d);
        return e;
    }
}
```

Output:


```

C:\WINDOWS\system32\cmd.exe
D:\JQR>javac Employ.java
D:\JQR>

```

Program 8: Write a program to show serialization of objects.

//ObjectOutputStream is used to store objects to a file

```

import java.io.*;
import java.util.*;
class StoreObj
{
    public static void main (String args[]) throws IOException
    {
        BufferedReader br = new BufferedReader (new InputStreamReader (System.in));
        FileOutputStream fos = new FileOutputStream ("objfile");
        ObjectOutputStream oos = new ObjectOutputStream ( fos );
        System.out.print ("Enter how many objects : ");
        int n = Integer.parseInt(br.readLine () );
        for(int i = 0;i<n;i++)
        {
            Employ e1 = Employ.getData ();
            oos.writeObject (e1);
        }
        oos.close ();
        fos.close ();
    }
}

```

Output:


```

C:\WINDOWS\system32\cmd.exe
D:\JQR>javac StoreObj.java
D:\JQR>java StoreObj
Enter how many objects : 2
Enter employ id : 1
Enter employ name : Ravi
Enter employ salary : 10000
Enter employ id : 2
Enter employ name : Chandra
Enter employ salary : 20000
D:\JQR>

```

Program 9: Write a program showing deserialization of objects.

//ObjectInputStream is used to read objects from a file

```

import java.io.*;
class ObjRead
{
    public static void main(String args[]) throws Exception
    {
        FileInputStream fis = new FileInputStream ("objfile");
        ObjectInputStream ois = new ObjectInputStream (fis);
        try
        {
            Employ e;

```

```

        while ( (e = (Employ) ois.readObject() ) != null)
            e.display ();
    }
    catch(EOFException ee)
    {
        System.out.println ("End of file Reached...");
    }
    finally
    {
        ois.close ();
        fis.close ();
    }
}

```

Output:

```

C:\WINDOWS\system32\cmd.exe
D:\JQR>javac  ObjRead.java
D:\JQR>java  ObjRead
1      Ravi   10000.0  Sun Oct 09 18:51:14 IST 2011
2      Chandra 20000.0  Sun Oct 09 18:51:26 IST 2011
End of file Reached...
D:\JQR>

```

File Class: File class of java.io package provides some methods to know the properties of a file or a directory. We can create the File class object by passing the filename or directory name to it.

- File obj = new File (filename);
- File obj = new File (directoryname);
- File obj = new File ("path", filename);
- File obj = new File ("path", directoryname);

File class Methods:

Methods	Description
boolean isFile ()	Returns true if the File object contains a filename, otherwise false
boolean isDirectory ()	Returns true if the File object contains a directory name
boolean canRead ()	Returns true if the File object contains a file which is readable
boolean canWrite ()	Returns true if the File object contains a file which is writable
boolean canExecute ()	Returns true if the File object contains a file which is executable
Boolean exists ()	Returns true when the File object contains a file or directory which physically exists in the computer.
String getParent ()	Returns the name of the parent directory
String getPath ()	Gives the name of directory path of a file or directory
String getAbsolutePath ()	Gives the fully qualified path
long length ()	Returns a number that represents the size of the file in bytes
boolean delete ()	Deletes the file or directory whose name is in File object
boolean createNewFile ()	Automatically creates a new, empty file indicated by File object, if and only if a file with this name does not yet exist.

Program 10: Write a program that uses File class methods.

//Displaying file properties

```
import java.io.*;
class FileProp
{
    public static void main(String args[])
    {
        String fname = args [0];
        File f = new File (fname);
        System.out.println ("File name: " + f.getName ());
        System.out.println ("Path:" + f.getPath ());
        System.out.println ("Absolute Path:" + f.getAbsolutePath ());
        System.out.println ("Parent:" + f.getParent ());
        System.out.println ("Exists:" + f.exists ());
        if ( f.exists() )
        {
            System.out.println ("Is writable: " + f.canWrite ());
            System.out.println ("Is readable: " + f.canRead ());
            System.out.println ("Is executable: " + f.canExecute ());
            System.out.println ("Is directory: " + f.isDirectory ());
            System.out.println ("File size in bytes: " + f.length ());
        }
    }
}
```

Output:



```
C:\WINDOWS\system32\cmd.exe
D:\JQR>javac FileProp.java
D:\JQR>java FileProp myfile
File name: myfile
Path:myfile
Absolute Path:D:\JQR\myfile
Parent:null
Exists:true
Is writable: true
Is readable: true
Is executable: true
Is directory: false
File size in bytes: 80
D:\JQR>
```

19. Networking in Java

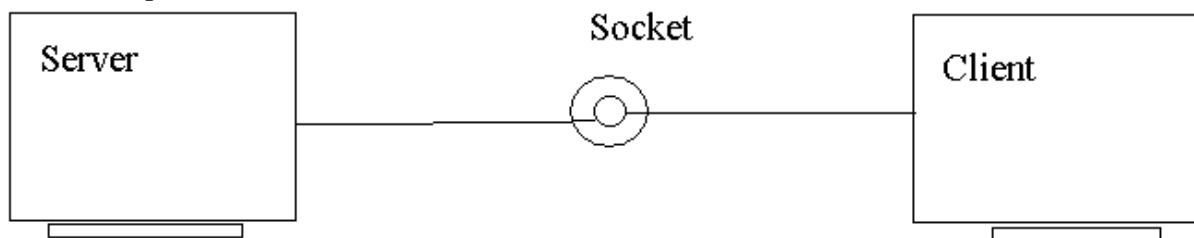
Inter Connection of computers is called a network. In a network, there may be several computers, some of them receiving the services and some of them providing the services to others. The computer which receives service is called a client and the computer which provides the service is called a server. Remember a client sometimes acts as a server and a server acts as a client. There are 3 requirements to establish a network:

- **Hardware:** includes the computers, cables, modems, hubs etc.
- **Software:** includes programs to communicate between servers and clients.
- **Protocol:** set of rules and regulations that represents a way to establish connection and helps in sending and receiving data in a standard format. Popular protocol suit is TCP/IP. TCP is connection oriented and IP connection less protocols.
e.g.: HTTP, FTP, SMTP, POP, UDP etc.

IP address: An IP address is a unique identification number allotted to every computer on a network or internet. IP address contains some bytes which identify the network and the actual computer inside the network.

DNS: Domain Naming Service is a service on internet that maps the IP addresses with corresponding website names.

Socket Programming: is a low level way to establish connection between server and a client with the help of a Socket.



- Data will be sent or received through the Socket.
- Socket at server side is called Server Socket.
- Socket at client side is called Client Socket.
- The id number allotted to a Socket is called port number.
- When a new socket is created a new port number is allotted.
- When the service on the socket is changed we should change the port number.
- A Socket is a point of connection between a server and client.
- Server Socket is created using ServerSocket class.
- Client Socket is created using Socket class.
- ServerSocket and Socket classes are available in java.net package.

Program 1: Write a program to create a server for sending some strings to the client.

//A server that sends message to client

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

```

class Server1
{
    public static void main(String args[]) throws IOException
    {
        //Create Server side socket
        ServerSocket ss = new ServerSocket (777);
        //make this socket accept client connection
        Socket s = ss.accept ();
        System.out.println ("A connection established...");
        //attach OutputStream to socket
        OutputStream obj = s.getOutputStream ();
        //to send data to Socket
        PrintStream ps = new PrintStream (obj);
        //now send the data
        String str = "Hello Client";
        ps.println (str);
        ps.println ("Bye");
        //close connection
        s.close ();
        ss.close ();
        ps.close ();
    }
}

```

Output:

Note: Do not run this program till client is also created.

Program 2: Write a program to create a client which accepts all the strings sent by the server

// a client that accepts data from server

```

import java.util.*;
import java.io.*;
import java.net.*;
class Client1
{
    public static void main(String args[]) throws IOException
    {
        //Create client socket
        Socket s = new Socket ("localhost", 777);
        //attach input stream to Socket
        InputStream obj = s.getInputStream ();
        //to receive data from socket
        BufferedReader br = new BufferedReader (new InputStreamReader (obj));
        //read data coming from server
        String str;
        while ((str = br.readLine() ) != null )
            System.out.println (str);
        //close connection
    }
}

```

```

        s.close ();
        br.close ();
    }
}

```

Output:

```

C:\WINDOWS\system32\cmd.exe
D:\JQR>javac Client1.java
D:\JQR>

```

After compiling Server1.java and Client1.java, run these programs in two separate dos windows:

```

C:\WINDOWS\system32\cmd.exe
D:\JQR>java Server1
A connection established...
D:\JQR>_

C:\WINDOWS\system32\cmd.exe
D:\JQR>java Client1
Hello Client
Bye
D:\JQR>

```

Program 3: Write a program to create a server such that the server receives data from the client using `BufferedReader` and then sends reply to the client using `PrintStream`.

//chat server

```
import java.net.*;
```

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

```
class Server2
```

```

{
    public static void main(String args[]) throws IOException
    {
        //create a server socket
        ServerSocket ss = new ServerSocket (888);
        //accept client connection
        Socket s = ss.accept ();
        System.out.println ("A connection is established...");
        //to send data to client
        PrintStream ps = new PrintStream (s.getOutputStream ());
        //to receive data from client
        BufferedReader br = new BufferedReader (new InputStreamReader
                                                (s.getInputStream ()));
        //to accept data from keyboard to sent to client
        BufferedReader kb = new BufferedReader (new InputStreamReader (System.in));
        while (true) //server runs continuously
        {
            String str, str1;
            while ( (str= br.readLine() ) != null)
            {
                System.out.println (str);
                str1 = kb.readLine ();
                ps.println (str1);
            }
            s.close ();      ss.close ();
            System.exit (0);
        }
    }
}

```

```

    }
}
}

```

Output:

```

C:\WINDOWS\system32\cmd.exe
D:\JQR>javac Server2.java
D:\JQR>

```

Program 4: Write a program to create a client which first connects to a server then starts the communication by sending a string to the server.

//chat client

```
import java.net.*;
```

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

```
class Client2
```

```

{
    public static void main(String args[]) throws IOException
    {
        //create a client socket
        Socket s = new Socket ("localhost", 888);
        //to send data to server
        DataOutputStream dos = new DataOutputStream (s.getOutputStream ());
        //to receive data from server
        BufferedReader br = new BufferedReader (new InputStreamReader
                                                (s.getInputStream ()));

        //to read data from keyboard to send to server
        BufferedReader kb = new BufferedReader (new InputStreamReader (System.in));
        //now communicate with Server
        String str, str1;
        while ( !(str = kb.readLine() ).equals("exit") )
        {
            dos.writeBytes (str+ "\n");
            str1 = br.readLine ();
            System.out.println (str1);
        }
        s.close ();
    }
}

```

Output:

```

C:\WINDOWS\system32\cmd.exe
D:\JQR>javac Client2.java
D:\JQR>

```

```

C:\WINDOWS\system32\cmd.exe
D:\JQR>java Server2
A connection is established...
Hello
Hai
bye
ok
D:\JQR>

```

```

C:\WINDOWS\system32\cmd.exe
D:\JQR>java Client2
Hello
Hai
bye
ok
exit
D:\JQR>

```

20. Threads

Executing the tasks is of two types:

- Single Tasking: Executing only one task at a time is called single tasking. In this single tasking the microprocessor will be sitting idle for most of the time. This means micro processor time is wasted.
- Multi tasking: Executing more than one task at a time is called multi tasking. Multitasking is of two types:
 - Process Based Multitasking: Executing several programs simultaneously is called process based multi tasking.
 - Thread Based Multitasking: Executing different parts of the same program simultaneously with the help of a thread is called thread based multitasking.

Advantage of multitasking is utilizing the processor time in an optimum way.

Java provides built-in support for multithreaded programming. A multithreaded program contains two or more parts that can run concurrently. Each part of such a program is called a thread. Thread is a smallest unit of code. Thread is also defined as a subprocess. A Thread sometimes called an execution context or a light weight process.

Uses of Threads:

- Threads are used in designing serverside programs to handle multiple clients at a time.
- Threads are used in games and animations.

Program 1: Write a program to know the currently running Thread

//Currently running thread

```
class Current
{
    public static void main(String args[])
    {
        System.out.println ("This is first statement");
        Thread t = Thread.currentThread ();
        System.out.println ("Current Thread: " + t);
        System.out.println ("Its name: " + t.getName ());
        System.out.println ("Its priority:" + t.getPriority ());
    }
}
```

Output:



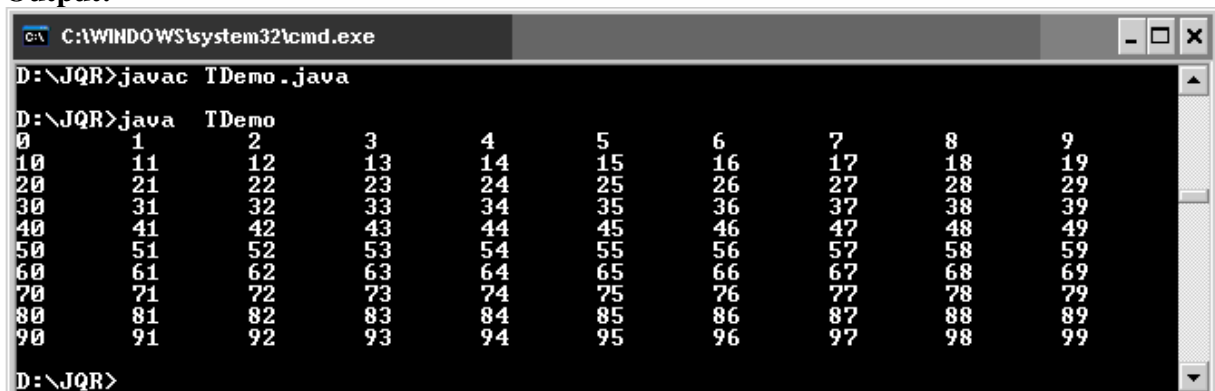
```
C:\WINDOWS\system32\cmd.exe
D:\JQR>javac Current.java
D:\JQR>java Current
This is first statement
Current Thread: Thread[main,5,main]
Its name: main
Its priority:5
D:\JQR>
```

Creating a Thread:

- Write a class that extends Thread class or implements Runnable interface this is available in lang package.
- Write public void run () method in that class. This is the method by default executed by any thread.
- Create an object to that class.
- Create a thread and attach it to the object.
- Start running the threads.

Program 2: Write a program to create and run a Thread.

```
//creating and running a Thread
class MyThread extends Thread
{
    public void run ()
    {
        for (int i = 0;i<100;i++)
        {
            System.out.print (i + "\t");
        }
    }
}
class TDemo
{
    public static void main(String args[])
    {
        MyThread obj = new MyThread ();
        Thread t = new Thread (obj);
        t.start ();
    }
}
```

Output:


```
C:\WINDOWS\system32\cmd.exe
D:\JQR>javac TDemo.java
D:\JQR>java TDemo
0      1      2      3      4      5      6      7      8      9
10     11     12     13     14     15     16     17     18     19
20     21     22     23     24     25     26     27     28     29
30     31     32     33     34     35     36     37     38     39
40     41     42     43     44     45     46     47     48     49
50     51     52     53     54     55     56     57     58     59
60     61     62     63     64     65     66     67     68     69
70     71     72     73     74     75     76     77     78     79
80     81     82     83     84     85     86     87     88     89
90     91     92     93     94     95     96     97     98     99
D:\JQR>
```

Multi Tasking Using Threads: In multi tasking, several tasks are executed at a time. For this purpose, we need more than one thread. For example, to perform 2 tasks we can take 2 threads and attach them to the 2 tasks. Then those tasks are simultaneously executed by the two threads. Using more than one thread is called 'multi threading'.

Program 3: Write a program to create more than one thread.

//using more than one thread is called Multi Threading

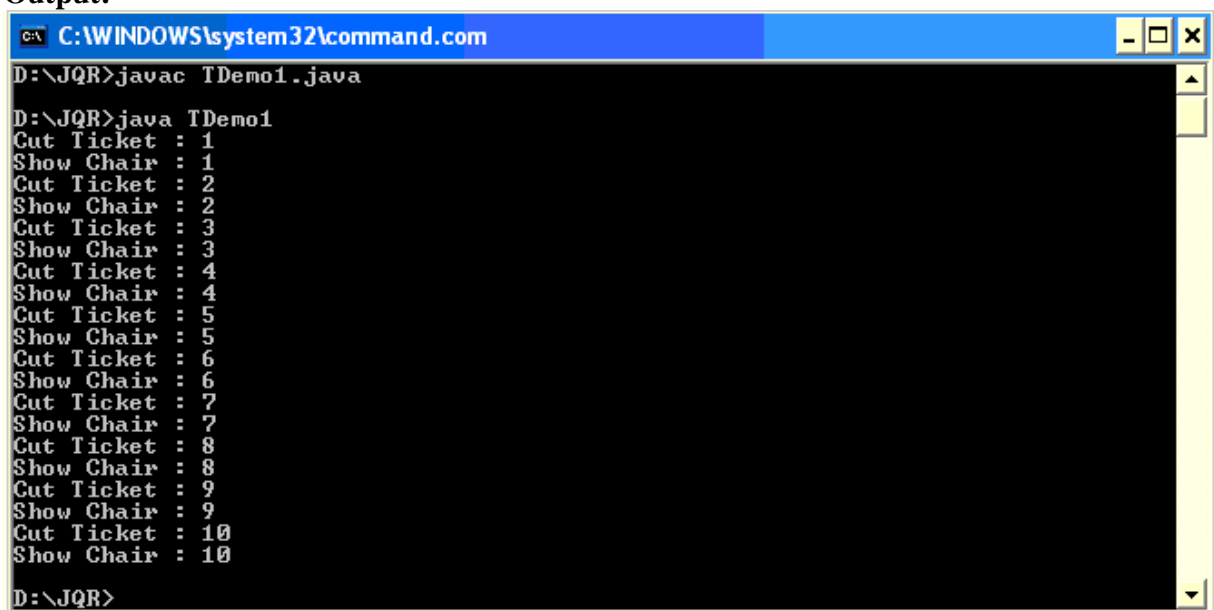
class Theatre extends Thread

```
{    String str;
    Theatre (String str)
    {        this.str = str;
    }
    public void run()
    {        for (int i = 1; i <= 10 ; i++)
            {            System.out.println (str + " : " + i);
                try
                {            Thread.sleep (2000);
                }
                catch (InterruptedException ie) {    ie.printStackTrace ();    }
            }
    }
}
```

class TDemo1

```
{    public static void main(String args[])
    {        Theatre obj1 = new Theatre ("Cut Ticket");
        Theatre obj2 = new Theatre ("Show Chair");
        Thread t1 = new Thread (obj1);
        Thread t2 = new Thread (obj2);
        t1.start ();
        t2.start ();
    }
}
```

Output:



```
C:\WINDOWS\system32\command.com
D:\JQR>javac TDemo1.java
D:\JQR>java TDemo1
Cut Ticket : 1
Show Chair : 1
Cut Ticket : 2
Show Chair : 2
Cut Ticket : 3
Show Chair : 3
Cut Ticket : 4
Show Chair : 4
Cut Ticket : 5
Show Chair : 5
Cut Ticket : 6
Show Chair : 6
Cut Ticket : 7
Show Chair : 7
Cut Ticket : 8
Show Chair : 8
Cut Ticket : 9
Show Chair : 9
Cut Ticket : 10
Show Chair : 10
D:\JQR>
```


In the preceding example, we have used 2 threads on the 2 objects of TDemo1 class. First we have taken a String variable str in Theatre class. Then we passed two strings- cut ticket and show chair into that variable from TDemo1 class. When t1. start () is executed, it starts execution run () method code showing cut ticket. Note that in run () method, we used: Thread. sleep (2000) is a static method in Thread class, which is used to suspend execution of a thread for some specified milliseconds. Since this method can throw InterruptedException, we caught it in catch block. When Thread t1 is suspended immediately t2. start () will make the thread t2 to execute and when it encounters Thread.sleep(2000), it will suspend for specified time meanwhile t1 will get executed respectively. In this manner, both the threads are simultaneously executed.

Multiple Threads Acting on Single Object: When two people (threads) want to perform same task then they need same object (run () method) to be executed each time. Take the case of railway reservation. Every day several people want reservation of a berth for them. The procedure to reserve the berth is same for all the people. So we need some object with same run () method to be executed repeatedly for all the people (threads).

Let us think that only one berth is available in a train and two passengers (threads) are asking for that berth in two different counters. The clerks at different counters sent a request to the server to allot that berth to their passengers. Let us see now to whom that berth is allotted.

Program 4: Write a program to create multiple threads and make the threads to act on single object.

//Multiple Threads acting on single object

class Reserve implements Runnable

```
{
    int available = 1;
    int wanted;
    Reserve (int i)
    {
        wanted = i;
    }
    public void run()
    {
        synchronized (this)
        {
            System.out.println ("Number of berths available: " + available);
            if ( available >= wanted)
            {
                String name = Thread.currentThread ().getName ();
                System.out.println (wanted + " berths allotted to: " + name);
                try
                {
                    Thread.sleep (2000); // wait for printing the ticket
                    available = available - wanted;
                }
                catch (InterruptedException ie)
                {
                    ie.printStackTrace ();
                }
            }
            else
            {
                System.out.println ("Sorry, no berths available");
            }
        }
    }
}
```

```

class Safe
{
    public static void main(String args[])
    {
        Reserve obj = new Reserve (1);
        Thread t1 =new Thread (obj);
        Thread t2 = new Thread (obj);
        t1.setName ("First Person");
        t2.setName ("Second Person");
        t1.start ();
        t2.start ();
    }
}

```

Output:

```

C:\WINDOWS\system32\command.com
D:\JQR>javac Safe.java
D:\JQR>java Safe
Number of berths available: 1
1 berths allotted to: First Person
Number of berths available: 0
Sorry, no berths available
D:\JQR>

```

If we would not use synchronized (this) block in the preceding program then when thread t1 enter into the run () method, it sees available number of berths as 1 and hence it allots it to First Person and displays “1 Berths reserved for First Person”. Then it enters try { } block inside run () method, where it will sleep for 2 seconds. In this time, the ticket will be printed on the printer. When the first thread is sleeping thread t2 also enters the run () method, it also sees that there is 1 berth remaining. The reason is for this is that the available number of berths is not yet updated by the first thread. So the second thread also sees 1 berth as available and it allots the same berth to the Second Person. Then the thread t2 will also go into sleep state. Thread t1 wakes up first and then it updates the available number of berths to zero (0). But at the same time the second thread has already allotted the same berth to the Second Person also. Since both the threads are acting on the same object simultaneously, the result will be unreliable.

Thread Synchronization or Thread Safe: When a thread is acting on an object preventing other threads from acting on the same object is called Thread Synchronization or Thread Safe. The Object on which the Threads are synchronized is called synchronized object or Mutex (Mutually Exclusive Lock). Thread synchronization is done in two ways:

- Using synchronized block we can synchronize a block of statements.
e.g.: synchronized (obj)

```

{
    statements;
}

```
- To synchronize an entire method code we can use synchronized word before method name
e.g.: synchronized void method ()

```

{
}

```

Thread Creation: To create a Thread, we can use the following forms:

```
Thread t1 = new Thread ();
Thread t2 = new Thread (obj);
Thread t3 = new Thread (obj, "thread-name");
```

Thread Class Methods:

- To know the currently running thread: `Thread t = Thread.currentThread ();`
- To start a thread: `t.start ();`
- To stop execution of a thread for a specific time: `Thread.sleep (milliseconds);`
- To get the name of the thread: `String name = t.getName ();`
- To set the new name to the thread: `t.setName ("New Name");`
- To get the priority of the thread: `int priority = t.getPriority();`
- To set the priority of the thread: `t.setPriority (int priority);`

Thread priorities can change from 1 to 10. We can also use the following constants to represent priorities:

`Thread.MAX_PRIORITY` value is 10

`Thread.MIN_PRIORITY` value is 1

`Thread.NORM_PRIORITY` value is 5

- To test if a thread is still alive: `t.isAlive ()` returns true/false
- To wait till a thread dies: `t.join ();`
- To send a notification to a waiting thread: `obj.notify ();`
- To send notification to all waiting threads: `obj.notifyAll ();`
- To wait till the obj is released (till notification is sent): `obj.wait ();`

Deadlock: When a Thread locked an object and waiting for another object to be released by another Thread, and the other thread is also waiting for the first thread to release the first object, both the threads will continue waiting forever. This is called "Thread Deadlock".

Even if we synchronize the threads, there is possibility of other problems like deadlock. Daily, thousands of people book tickets in trains and cancel tickets also. If a programmer is to develop code for this, he may visualize that booking tickets and canceling them are reverse procedures. Hence, he will write these 2 tasks as separate and opposite tasks and assign 2 different threads to do these tasks simultaneously.

To book a ticket, the thread will enter the train object to verify that the ticket is available or not. When there is a ticket, it updates the available number of tickets in the train object. For this, it takes say 150 milli seconds. Then it enters the compartment object. In compartment object, it should allot the ticket for the passenger and update its status to reserved. This means the thread should go through both the train and compartment objects.

Similarly, let us think if a thread has to cancel a ticket, it will first enter compartment object and updates the status of the ticket as available. For this it is taking say 200 milliseconds. Then it enters train object and updates the available number of tickets there. So, this thread also should go through both the compartment and train objects.

When the BookTicket thread is at train object for 150 milliseconds, the CancelTicket thread will be at compartment object for 200 milliseconds. Because we are using multiple (more than one) threads, we should synchronize them. So, the threads will lock those objects. When 150 milliseconds time is over, BookTicket thread tries to come out of train object and wants to lock on compartment object, by entering it. At that time, it will find that the compartment object

is already locked by another thread (CancelTicket) and hence it will wait. BookTicket thread will wait for compartment object for another 50 milli seconds.

After 200 milliseconds time is up, the CancelTicket thread which is in compartment object completes its execution and wants to enter and lock on train object. But it will find that the train object is already under lock by BookTicket thread and hence is not available. Now, CancelTicket will wait for the train object which should be unlocked by BookTicket.

In this way, BookTicket thread keeps on waiting for the CancelTicket thread to unlock the compartment object and the CancelTicket thread keeps on waiting for the BookTicket to unlock the train object. Both the threads will wait forever in this way, this situation is called DealLock.

Program 5: Write a program to get a deadlock situation using threads.

//to cancel the ticket

class CancelTicket extends Thread

```
{
    Object train, comp;
    CancelTicket (Object train, Object comp)
    {
        this.train = train;
        this.comp = comp;
    }
    public void run()
    {
        synchronized (comp)
        {
            System.out.println ("Cancel ticket has locked on compartment");
            try
            {
                Thread.sleep (2000);
            }
            catch (InterruptedException ie) { }
            System.out.println ("Cancel ticket tries to lock train object...");
            synchronized (train)
            {
                System.out.println ("Cancel ticket has locked train...");
            }
        }
    }
}
```

//to book the ticket

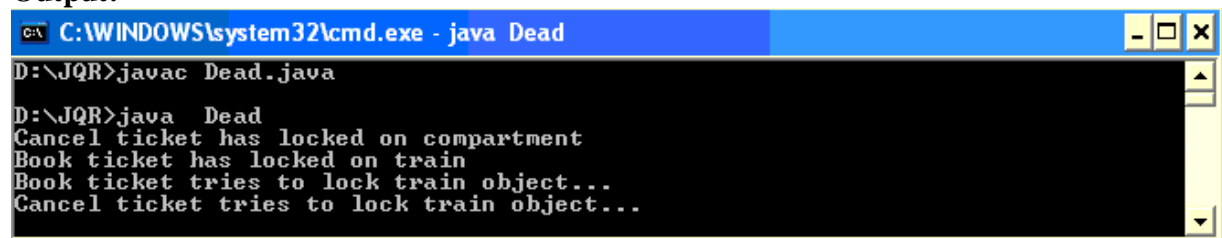
class BookTicket extends Thread

```
{
    Object train, comp;
    BookTicket (Object train, Object comp)
    {
        this.train = train;
        this.comp = comp;
    }
    public void run()
    {
        synchronized (train)
        {
            System.out.println ("Book ticket has locked on train");
        }
    }
}
```

```

        try
        {
            Thread.sleep (2000);
        }
        catch (InterruptedException ie)
        {
        }
        System.out.println ("Book ticket tries to lock train object...");
        synchronized (comp)
        {
            System.out.println ("Book ticket has locked compartment...");
        }
    }
}
}
class Dead
{
    public static void main (String args[])
    {
        Object train = new Object ();
        Object compartment = new Object ();
        CancelTicket obj1 = new CancelTicket (train, compartment);
        BookTicket obj2 = new BookTicket (train, compartment);
        Thread t1 = new Thread (obj1);
        Thread t2 = new Thread (obj2);
        t1.start ();
        t2.start ();
    }
}

```

Output:


```

C:\WINDOWS\system32\cmd.exe - java Dead
D:\JQR>javac Dead.java
D:\JQR>java Dead
Cancel ticket has locked on compartment
Book ticket has locked on train
Book ticket tries to lock train object...
Cancel ticket tries to lock train object...

```

There is no specific solution for preventing deadlock. The programmer should exercise proper caution while planning the logic of the program to avoid deadlocks.

Thread Communication: In some cases two or more threads should communicate with each other. One thread output may be send as input to other thread. For example, a consumer thread is waiting for a Producer to produce the data (or some goods). When the Producer thread completes production of data, then the Consumer thread should take that data and use it.

In producer class we take a StringBuffer object to store data, in this case; we take some numbers from 1 to 5. These numbers are added to StringBuffer object. Until producer completes placing the data into StringBuffer the consumer has to wait. Producer sends a notification immediately after the data production is over.

Program 6: Write a program to demonstrate Thread communication

//inter thread communication

class Producer implements Runnable

```

{
    StringBuffer sb;
    Producer ()
    {
        sb = new StringBuffer();
    }
    public void run ()
    {
        synchronized (sb)
        {
            for (int i=1;i<=5;i++)
            {
                try
                {
                    sb.append (i + " : ");
                    Thread.sleep (500);
                    System.out.println (i + " appended");
                }
                catch (InterruptedException ie){ }
            }
            sb.notify ();
        }
    }
}

```

class Consumer implements Runnable

```

{
    Producer prod;
    Consumer (Producer prod)
    {
        this.prod = prod;
    }
    public void run()
    {
        synchronized (prod.sb)
        {
            try
            {
                prod.sb.wait ();
            }
            catch (Exception e) { }
            System.out.println (prod.sb);
        }
    }
}

```

class Communicate

```

{
    public static void main(String args[])
    {
        Producer obj1 = new Producer ();
        Consumer obj2 = new Consumer (obj1);
        Thread t1 = new Thread (obj1);
    }
}

```

```

        Thread t2 = new Thread (obj2);
        t2.start ();
        t1.start ();
    }
}

```

Output:

```

C:\WINDOWS\system32\cmd.exe
D:\JQR>javac Communicate.java
D:\JQR>java Communicate
1 appended
2 appended
3 appended
4 appended
5 appended
1 : 2 : 3 : 4 : 5 :
D:\JQR>

```

Both sleep () and wait () methods are used to suspend a thread execution for a specified time. When sleep () is executed inside a synchronized block, the object is still under lock. When wait () method is executed, it breaks the synchronized block, so that the object lock is removed and it is available.

Thread Group: A ThreadGroup represents a group of threads. The main advantage of taking several threads as a group is that by using a single method, we will be able to control all the threads in the group.

- Creating a thread group: `ThreadGroup tg = new ThreadGroup ("groupname");`
- To add a thread to this group (tg): `Thread t1 = new Thread (tg, targetobj, "threadname");`
- To add another thread group to this group (tg):
`ThreadGroup tg1 = new ThreadGroup (tg, "groupname");`
- To know the parent of a thread: `tg.getParent ();`
- To know the parent thread group: `t.getThreadGroup ();`
This returns a ThreadGroup object to which the thread t belongs.
- To know the number of threads actively running in a thread group: `t.activeCount ();`
- To change the maximum priority of a thread group tg: `tg.setMaxPriority ();`

Program 7: Write a program to demonstrate the creation of thread group.

```

//Using ThreadGroup
import java.io.*;
class WhyTGroups
{
    public static void main (String args[]) throws IOException
    {
        Reservation res = new Reservation ();
        Cancellation can = new Cancellation ();
        //Create a ThreadGroup
        ThreadGroup tg = new ThreadGroup ("Reservation Group");
        //Create 2 threads and add them to thread group
        Thread t1 = new Thread (tg, res, "First Thread");
        Thread t2 = new Thread (tg, res, "Second Thread");
    }
}

```

```

        //Create another thread group as a child to tg
        ThreadGroup tg1 = new ThreadGroup (tg, "Cancellation Group");
        Thread t3 = new Thread (tg1, can, "Third Thread");
        Thread t4 = new Thread (tg1, can, "Fourth Thread");
        //find parent group of tg1
        System.out.println ("Parent of tg1 = " + tg1.getParent ());
        //set maximum priority
        tg1.setMaxPriority (7);
        System.out.println ("Thread group of t1 = " + t1.getThreadGroup ());
        System.out.println ("Thread group of t3 = " + t3.getThreadGroup ());
        t1.start ();
        t2.start ();
        t3.start ();
        t4.start ();
        System.out.println ("Number of threads in this group : " + tg.activeCount ());
    }
}
class Reservation extends Thread
{
    public void run ()
    {
        System.out.println ("I am Reservation Thread");
    }
}
class Cancellation extends Thread
{
    public void run ()
    {
        System.out.println ("I am Cancellation Thread");
    }
}
}

```

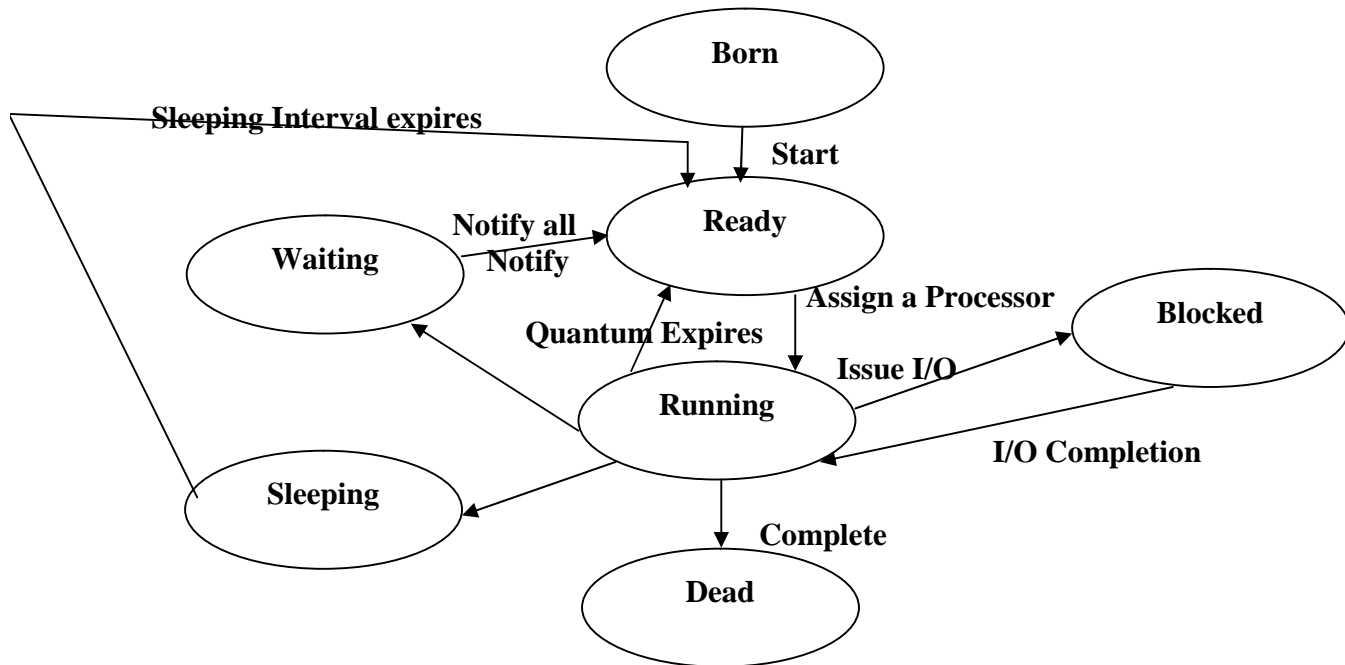
Output:

```

C:\WINDOWS\system32\cmd.exe
D:\JQR>javac WhyTGroups.java
D:\JQR>java WhyTGroups
Parent of tg1 = java.lang.ThreadGroup[name=Reservation Group,maxpri=10]
Thread group of t1 = java.lang.ThreadGroup[name=Reservation Group,maxpri=10]
Thread group of t3 = java.lang.ThreadGroup[name=Cancellation Group,maxpri=7]
I am Reservation Thread
Number of threads in this group : 4
I am Cancellation Thread
I am Cancellation Thread
I am Reservation Thread
D:\JQR>

```


Thread States (Life-Cycle of a Thread): The life cycle of a thread contains several states. At any time the thread falls into any one of the states.



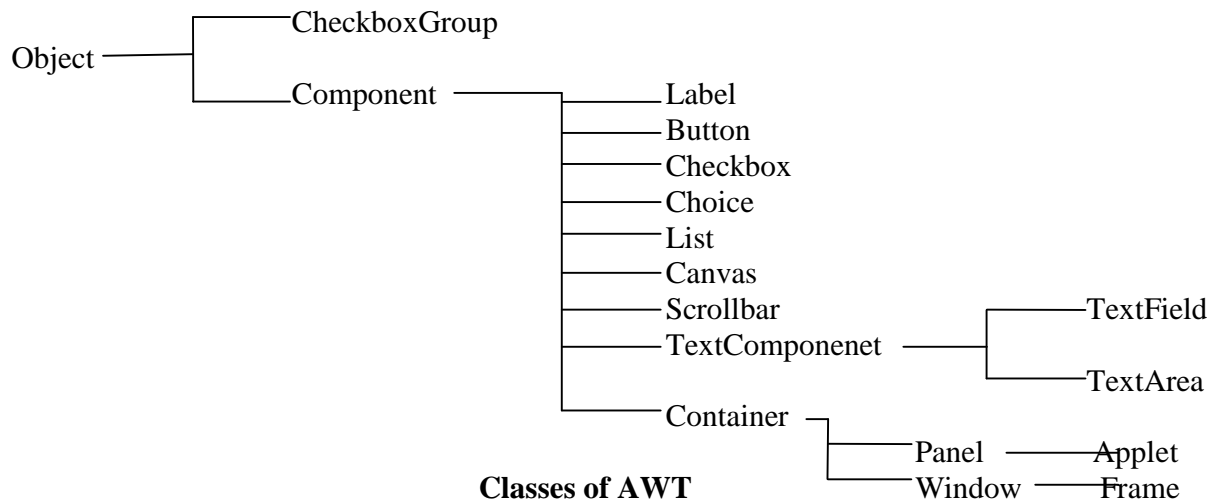
- The thread that was just created is in the born state.
- The thread remains in this state until the thread's start method is called. This causes the thread to enter the ready state.
- The highest priority ready thread enters the running state when the system assigns a processor to the thread i.e., the thread begins executing.
- When a running thread calls wait, the thread enters into a waiting state for the particular object on which wait was called. Every thread in the waiting state for a given object becomes ready on a call to notify all by another thread associated with that object.
- When a sleep method is called in a running thread, that thread enters into the suspended (sleep) state. A sleeping thread becomes ready after the designated sleep time expires. A sleeping thread cannot use a processor even if one is available.
- A thread enters the dead state when its run () method completes (or) terminates for any reason. A dead thread is eventually disposed of by the system.
- One common way for a running thread to enter the blocked state is when the thread issues an input or output request. In this case, a blocked thread becomes ready when the input or output waits for completion. A blocked thread can't use a processor even if one is available.

21. Abstract Window Toolkit

User interaction with the program is of two types:

- CUI (Character User Interface): In CUI user interacts with the application by typing characters or commands. In CUI user should remember the commands. It is not user friendly.
- GUI (Graphical User Interface): - In GUI user interacts with the application through graphics. GUI is user friendly. GUI makes application attractive. It is possible to simulate real object in GUI programs.

In java to write GUI programs we can use awt (Abstract Window Toolkit) package. java.awt is a package that provides a set of classes and interfaces to create GUI programs.



A window represents a box shaped area on the screen. Window does not have border and title.

A Frame is a top level window that is not contained in another window. A Frame contains border and title.

Creating the Frame:

- We can create a Frame by creating Frame class object. `Frame obj = new Frame ();`
(or)

Create a class that extends Frame class then create an object to that class.

```
class MyClass extends Frame
```

```
MyClass obj = new MyClass ();
```

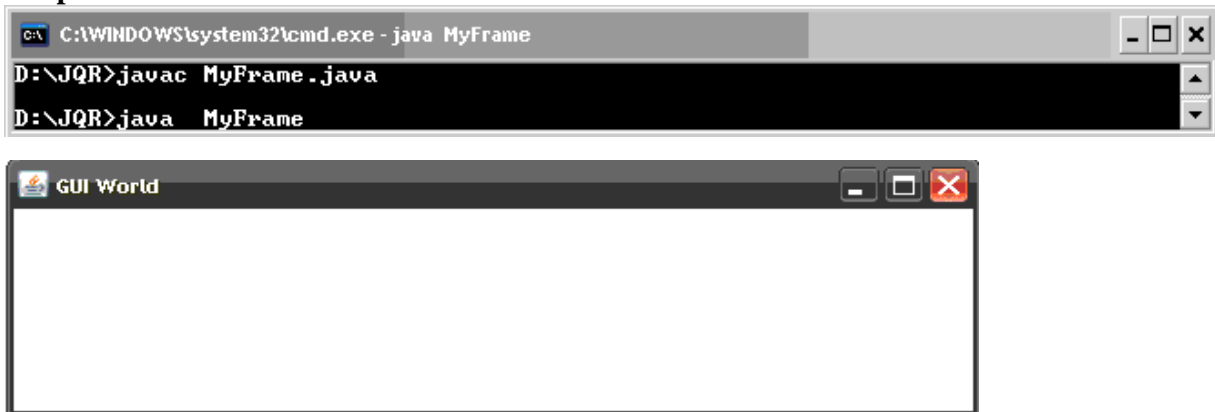
- After creating the Frame we need to set Frame width and height using `setSize ()` method as:
`f.setSize (400, 350);`
- We can display the frame using `setVisible ()` method as:
`f.setVisible (true);`

Program 1: Write a program to create a Frame without extending Frame class.

```
//creating a Frame
import java.awt.*;
class MyFrame
```

```
{  
    public static void main(String args[])  
    {  
        Frame f1 = new Frame ();  
        f1.setSize (500,150);  
        f1.setTitle ("GUI World");  
        f1.setVisible (true);  
    }  
}
```

Output:



Program 2: Write a program to create a Frame by extending Frame class.

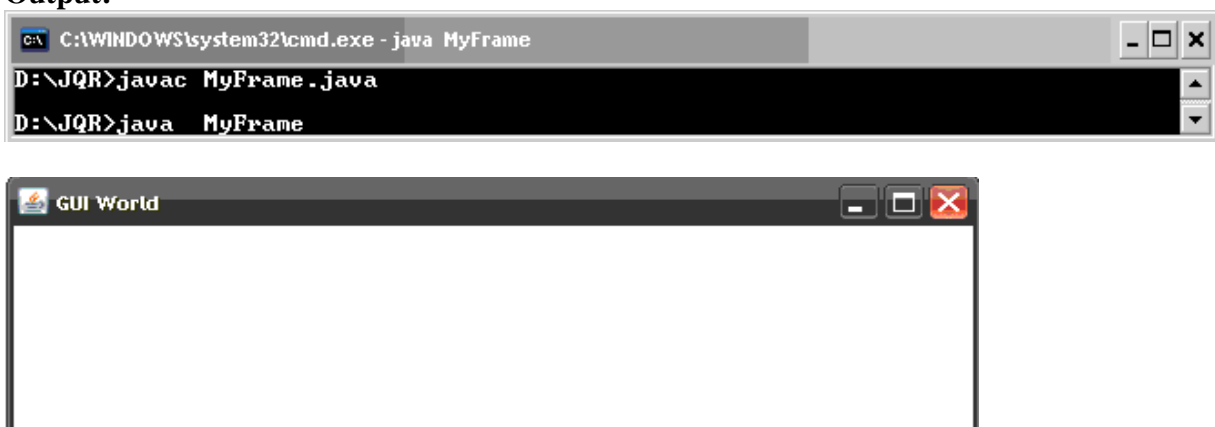
//creating a Frame

import java.awt.*;

class MyFrame extends Frame

```
{  
    public static void main(String args[])  
    {  
        MyFrame f1 = new MyFrame ();  
        f1.setSize (500,200);  
        f1.setTitle ("GUI World");  
        f1.setVisible (true);  
    }  
}
```

Output:



The frame can be minimized, maximized and resized but cannot be closed. Even if we click on close button of the frame, it will not perform any closing action. Closing a frame means attaching action to the component. To attach actions to the components, we need 'Event Delegation Model'.

Event-Delegation-Model: Graphical representation of an object is called a component. User interaction with the component is called event. When an event is generated by the user on the component, component will delegate (hand over) the event to the listener. The listener will delegate the event to one of its method. The method is finally executed and the event is handled. This is called Event-Delegation-Model. Event-Delegation-Model is used in awt to provide actions for components. In Event Delegation Model:

- Attach the Listener to the component.
- Implement all the methods of the Listener.
- When an Event is generated one of these methods performs the required action.

Closing the Frame: We know Frame is also a component. We want to close the frame by clicking on its close button. Let us follow these steps to see how to use event delegation model to do this:

- We should attach a listener to the frame component. Remember, all listeners are available in java.awt.event package. The most suitable listener to the frame is 'WindowListener'. It can be attached using addWindowListener () method as:

```
f.addWindowListener (WindowListener obj);
```

Please note that the addWindowListener () method has a parameter that is expecting object of WindowListener interface. Since it is not possible to create an object to an interface, we should create an object to the implementation class of the interface and pass it to the method.

- Implement all the methods of the WindowListener interface. The following methods are found in WindowListener interface:

```
public void windowActivated (WindowEvent e)
public void windowClosed (WindowEvent e)
public void windowClosing (WindowEvent e)
public void windowDeactivated (WindowEvent e)
public void windowDeiconified (WindowEvent e)
public void windowIconified (WindowEvent e)
public void windowOpened (WindowEvent e)
```

In all the preceding methods, WindowListener interface calls public void windowClosing () method when the frame is being closed. So, implementing this method alone is enough, as:

```
public void windowClosing (WindowEvent e)
{
    //Close the application
    System.exit (0);
}
```

For, the remaining methods, we can provide empty body.

- So, when the frame is closed the body of this method is executed and the application gets closed. In this way, we can handle the frame closing event.

An adapter class is an implementation class of a listener interface which contains all methods implemented with empty body. For example, WindowAdapter is an adapter class of

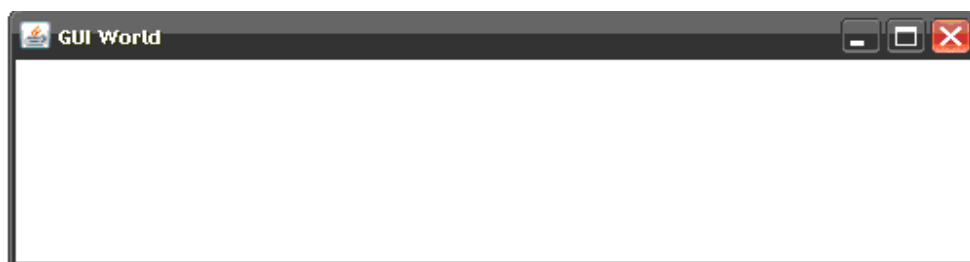
WindowListener interface. WindowAdapter in java.awt.event package contains all the methods of the WindowListener interface with an empty implementation (body). If we can extend MyClass from this WindowAdapter class, then we need not write all the methods with empty implementation. We can write only that method which we are in need.

Program 3: Write a program to close the frame using WindowAdapter class.

//creating a Frame and closing the same

```
import java.awt.*;
class MyFrame extends Frame
{
    public static void main(String args[])
    {
        MyFrame f1 = new MyFrame ();
        f1.setSize (500,200);
        f1.setTitle ("GUI World");
        f1.setVisible (true);
        f1.addWindowListener (new MyClass ());
    }
}
class MyClass extends WindowAdapter
{
    public void windowClosing (WindowEvent e)
    {
        System.exit (0);
    }
}
```

Output:



Note: Click on close button, the Frame closes.

The code of MyClass can be copied directly into addWindowListener () method, as:

```
f1.addWindowListener (new WindowAdapter ()
{
    public void windowClosing(WindowEvent e)
    {
        System.exit (0);
    }
});
```

We cannot find the name of MyClass anywhere in the code. It means the name of MyClass is hidden in MyFrame class and hence MyClass is an inner class in MyFrame class whose name is not mentioned. Such an inner class is called 'anonymous inner class'. An anonymous inner class is an inner class whose name is not written and for which only one object is created. A class for which only one object is created is called singleton class.

Displaying text in the Frame: We need paint () method whenever we want to display some new drawing or text or images in the Frame. The paint () method is automatically called when a frame is created and displayed. The paint () method refreshes the frame contents automatically when a drawing is displayed. The paint () method takes Graphics Class object as parameter. Graphics class is present in java.awt package.

- To display some text or strings in the frame, we can take the help of drawstring () method of Graphics class as:

```
g.drawString ("Hai Readers", x, y);
```

Here, the string "Hai Readers" will be displayed starting from the coordinates (x, y).

- If we want to set some color for the text, we can use setColor () method of Graphics class as:

```
g.setColor (Color.red);
```

There are two ways to set a color in awt.

- The first way is by directly mentioning the needed standard color name from Color class as Color.black, Color.blue, Color.cyan, Color.pink, Color.red, Color.orange, Color.magenta, Color.darkGray, Color.gray, Color.lightGray, Color.green, Color.yellow and Color.white etc.
- The second way to mention any color is by combining the three primary colors: red, green and blue while creating Color class object as:

```
Color c = new Color (r, g, b);
```

Here, r, g, b values can change from 0 to 255. 0 represents no color. 10 represent low intensity whereas 200 represent high intensity of color.

e.g.: Color c = new Color (255, 0, 0); //red color

- To set some font to the text, we can use setFont () method of Graphics class, as:

```
g.setFont (Font object);
```

This method takes Font class object, which can be created as:

```
Font f = new Font ("SansSerif", Font.BOLD, 30);
```

Here, "SansSerif" represents the font name; Font.BOLD represents the font style and 30 represents the font size in pixels.

Program 4: Write a program to display a message in the frame.

//Displaying a message in the frame

```
import java.awt.*;
```

```
import java.awt.event.*;
```

```
class Message extends Frame
```

```
{
    Message ()
    {
        //code to close the Frame
        this.addWindowListener (new WindowAdapter ()
        {
            public void windowClosing (WindowEvent we)
            {
```

```

        System.exit (0);
    }
});
}
public static void main (String args[])
{
    Message m = new Message ();
    m.setTitle ("Simple Message");
    m.setSize (650,150);
    m.setVisible (true);
}
public void paint (Graphics g)
{
    this.setBackground (Color.green);
    g.setColor (Color.red);
    Font f = new Font ("Times New Roman", Font.BOLD+Font.ITALIC, 60);
    g.setFont (f);
    g.drawString ("Hello Readers!", 50, 100);
}
}

```

Output:

Drawing in the Frame: Graphics class of java.awt package has the following methods which help to draw various shapes.

- drawLine (int x1, int y1, int x2, int y2): to draw a line connecting (x1, y1) and (x2, y2).
- drawRect (int x, int y, int w, int h): draws outline of the rectangle, rectangles top left corner starts at (x, y) the width is w and height is h.
- drawRoundRect (int x, int y, int w, int h, int arcw, int arch): draws the outline of the rectangle with rounded corners. Rectangles top left corner starts at (x, y) the width is w and height is h. The rounding is specified by arcw and arch.
- drawOval (int x, int y, int w, int h): This method draws a circle or ellipse.
- drawArc(int x, int y, int w, int h, int sangle, int aangle): draws an arc where sangle stands for starting angle and aangle stands for ending angle.

- `drawPolygon (int x[], int y[], int n)`: This method draws a polygon that connects pairs of coordinates . Here, `x[]` is an array which holds x coordinates of points and `y[]` is an array which holds y coordinates, `n` represents the number of pairs of coordinates.

To fill any shape with a desired color, first of all we should set a color using `setColor ()` method. Then any of the following methods will draw those respective shapes by filling with the color.

- `fillRect (int x, int y, int w, int h)`: Draws a rectangle and fills it with the specified color.
- `drawRoundRect (int x, int y, int w, int h, int arcw, int arch)`: Draws filled rectangle with rounded corners.
- `drawOval (int x, int y, int w, int h)`: Draws filled oval with a specified color.
- `drawArc(int x, int y, int w, int h, int sangle, int aangle)`: Draws an arc and fills it with a specified color.
- `drawPolygon (int x[], int y[], int n)`: Draws and fills a polygon with a specified color.

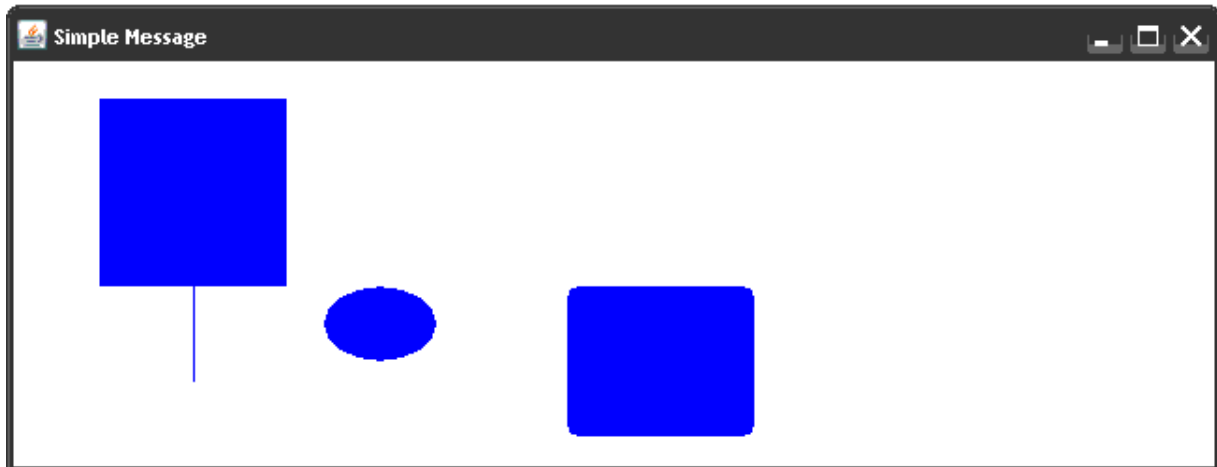
Program 5: Write a program to draw different shapes in the frame.

//Program to draw different shapes

```
import java.awt.*;
import java.awt.event.*;
class Draw1 extends Frame
{
    Draw1 ()
    {
        this.addWindowListener (new WindowAdapter ()
        {
            public void windowClosing(WindowEvent we)
            {
                System.exit (0);
            }
        });
    }
    public static void main (String args[])
    {
        Draw1 d = new Draw1 ();
        d.setTitle ("Simple Message");
        d.setSize (650,350);
        d.setVisible (true);
    }
    public void paint (Graphics g)
    {
        g.setColor (Color.blue);
        g.drawLine (100, 100, 100, 200);
        g.fillRect (50, 50, 100, 100);
        g.fillRoundRect (300, 200, 100, 80, 10, 10);
        g.fillOval (170, 150, 60, 40);
    }
}
```

Output:





Displaying images in the frame:

- Load the image into Image class object using getImage () method of Toolkit class.
`Image img = Toolkit.getDefaultToolkit ().getImage ("diamonds.gif");`
- But, loading the image into img will take some time, for this purpose we need MediaTracker class. Add image to MediaTracker class and allot an identification number to it starting from 0, 1...
`MediaTracker track = new MediaTracker (this);`
`track.addImage (img, 0);`
- Now, MediaTracker keeps JVM waiting till the image is loaded completely. This is done by waitForId () method.
`track.waitForId (0);`
- Once the image is loaded and available in img, then we can display the image using drawImage () method of Graphics class as:
`g.drawImage (img, 50, 50, null);`
 Here null represents ImageObserver class object which is not required. ImageObserver is useful to store history of how the image is loaded into the Object.

Program 6: Write a Program to display image on the frame using awt.

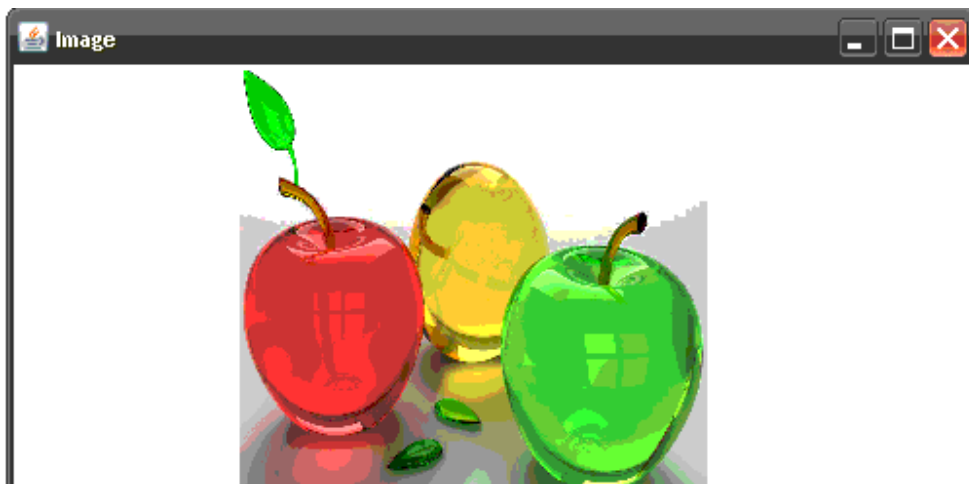
//Program to display an image

```
import java.awt.*;
import java.awt.event.*;
class Images extends Frame
{
    Image img;
    Images ()
    {
        img = Toolkit.getDefaultToolkit().getImage ("Fruit.jpg");
        MediaTracker track = new MediaTracker (this);
        track.addImage (img,0);
        try
        {
            track.waitForID (0);
        }
    }
}
```

```

        catch (InterruptedException ie){}
        addWindowListener (new WindowAdapter ()
        {
            public void windowClosing (WindowEvent we)
            {
                System.exit (0);
            }
        });
    }
    public void paint(Graphics g)
    {
        g.drawImage (img, 120, 20, 240, 240, null);
    }
    public static void main(String args[])
    {
        Images ob = new Images();
        ob.setTitle("Image");
        ob.setSize(500,250);
        ob.setVisible(true);
    }
}

```

Output:

Component Class Methods: A component is a graphical representation of an object on the screen. For example, push buttons, radio buttons, menus etc are components. Even frame is also a component. There is Component class available in java.awt package which contains the following methods which are applicable to any component.

Method	Description
Font getFont ()	This method returns the font of the component.
void setFont (Font f)	This method sets a particular font f for the text of the component.
Color getForeground ()	This method gives the foreground color of the component.
void setForeground (Color c)	This method sets a foreground color c to the component.
Color getBackground ()	Gets the background color of the component.
void setBackground (Color c)	Sets the background color c for the component.
String getName ()	Returns the name of the component.
void setName (String name)	Sets a new name for the component.
int getHeight ()	Returns the height of the component in pixels as an integer.
int getWidth ()	Returns the width of the component in pixels as an integer.
Dimension getSize ()	Returns the size of the component as an object of Dimension class. Dimension.width and Dimension.height will provide the width and height of the component.
int getX ()	Returns the current x coordinate of the components origin.
int getY ()	Returns the current y coordinate of the components origin.
Point getLocation ()	Gets the location of the component in the form of a point specifying the components top-left corner.
void setLocation (int x, int y)	Moves the component to a new location specified by (x, y).
void setSize(int width, int height)	Resizes the component so that it has new width and height as passed to setSize () method.
void setVisible (boolean b)	Shows the component if the value of b is true or hides the component if the value of parameter b is false.
void setEnabled (boolean b)	Enables the component if the value of b is true or disables the component if the value of parameter b is false.
void setBounds (int x, int y, int w, int h)	This method allots a rectangular area starting at (x ,y) coordinates and with width w and height h. The component is resized to this area before its display. This method is useful to specify the location of the component in the frame.

- After creating a component, we should add the component to the frame. For this purpose, add () method is used. `f.add (component);` where f is frame class object.
- Similarly, to remove a component from the frame, we can use remove () method as: `f.remove (component);` where f is frame class object.
- Frame class contains a method called setLayout (). setLayout () is useful to set a layout for the frame. A layout represents a manner of arranging components in the frame. All layouts are represented as implementation classes of LayoutManager interface. For example, the following layouts are available in AWT:
 - FlowLayout: FlowLayout is useful to arrange the components in a line after the other. When a line is filled with components, they are automatically placed in the next line.
 - BorderLayout: BorderLayout is useful to arrange the components in the 4 borders of the frame as well as in the center. The borders are specified as South, North, East, West and Center.
 - CardLayout: A cardLayout treats each component as a card. Only one card is visible at a time and it arranges the components as a stack of cards.

- **GridLayout:** It is useful to divide the display area into a two dimensional grid form that contains several rows and columns. The display area is divided into equal sized rectangles and one component is placed in each rectangle.
- **GridBagLayout:** This layout is more flexible as compared to other layouts since in this layout the components can span more than one row or column and the size of the components can be adjusted to fit the display area.

We will discuss about layout manager in later chapter. The following points are helpful to understand how to work with a layout manager:

- To set a layout for our components, we can pass the layout class object to the `setLayout ()` method as: `setLayout (new FlowLayout ());`
- Suppose, we do not want to set any layout, then we should pass null to the `setLayout ()` method as: `setLayout (null);`
- Suppose, we do not use `setLayout ()` method at all, then the Java compiler assumes a default layout manager. The default layout in case of a frame is `BorderLayout`.

Listeners and Listener Methods: Listeners are available for components. A Listener is an interface that listens to an event from a component. Listeners are available in `java.awt.event` package. The methods in the listener interface are to be implemented, when using that listener.

Component	Listener	Listener methods
Button	ActionListener	public void actionPerformed (ActionEvent e)
Checkbox	ItemListener	public void itemStateChanged (ItemEvent e)
CheckboxGroup	ItemListener	public void itemStateChanged (ItemEvent e)
TextField	ActionListener FocusListener	public void actionPerformed (ActionEvent e) public void focusGained (FocusEvent e) public void focusLost (FocusEvent e)
TextArea	ActionListener FocusListener	public void actionPerformed (ActionEvent e) public void focusGained (FocusEvent e) public void focusLost (FocusEvent e)
Choice	ActionListener ItemListener	public void actionPerformed (ActionEvent e) public void itemStateChanged (ItemEvent e)
List	ActionListener ItemListener	public void actionPerformed (ActionEvent e) public void itemStateChanged (ItemEvent e)
Scrollbar	AdjustmentListener MouseMotionListener	public void adjustmentValueChanged (AdjustmentEvent e) public void mouseDragged (MouseEvent e) public void mouseMoved (MouseEvent e)
Label	No listener is needed	

Creating Push Buttons: Button class is useful to create push buttons. A push button triggers a series of events.

- To create push button: `Button b1 =new Button("label");`
- To get the label of the button: `String l = b1.getLabel();`
- To set the label of the button: `b1.setLabel("label");`
- To get the label of the button clicked: `String str = ae.getActionCommand();`
where ae is object of `ActionEvent`

Program 7: Write a program to create push buttons.

//Program to create push buttons

```
import java.awt.*;
import java.awt.event.*;
class MyButton extends Frame implements ActionListener
{
    Button b1, b2;
    MyButton ()
    {
        setLayout (null);
        b1 = new Button ("Red");
        b2 = new Button ("Yellow");
        b1.setBounds (100, 50, 75, 40);
        b2.setBounds (100, 120, 75, 40);
        b1.addActionListener (this);
        b2.addActionListener (this);
        add (b1);
        add (b2);
        //code to close the frame
        addWindowListener (new WindowAdapter ()
        {
            public void windowClosing (WindowEvent we)
            {
                System.exit (0);
            }
        });
    }
    public void actionPerformed (ActionEvent ae)
    {
        String str = ae.getActionCommand ();
        if (str.equals ("Red") )
            setBackground (Color.red);
        if (str.equals ("Yellow"))
            setBackground (Color.yellow);
    }
    public static void main(String args[])
    {
        MyButton ob = new MyButton ();
        ob.setSize (500,200);
        ob.setTitle ("Buttons...!");
        ob.setVisible (true);
    }
}
```

Output:





Checkbox: A Checkbox is a square shaped box which provides a set of options to the user.

- To create a Checkbox: `Checkbox cb = new Checkbox ("label");`
- To create a checked Checkbox: `Checkbox cb = new Checkbox ("label", null, true);`
- To get the state of a Checkbox: `boolean b = cb.getState ();`
- To set the state of a Checkbox: `cb.setState (true);`
- To get the label of a Checkbox: `String s = cb.getLabel ();`

Program 8: Write a program to create checkboxes.

//Program to create checkboxes

```
import java.awt.*;
import java.awt.event.*;
class MyCheckbox extends Frame implements ItemListener
{
    Checkbox cb1, cb2;
    String msg;
    MyCheckbox ()
    {
        setLayout (new FlowLayout ());
        cb1 = new Checkbox ("Bold", null, true);
        cb2 = new Checkbox ("Italic");
        add (cb1);
        add (cb2);
        cb1.addItemListener (this);
        cb2.addItemListener (this);
        addWindowListener (new WindowAdapter ()
        {
            public void windowClosing (WindowEvent we)
            {
                System.exit (0);
            }
        });
    }
    public void itemStateChanged (ItemEvent ie)
    {
        repaint ();
    }
}
```

```

    public static void main (String args[])
    {
        MyCheckbox ob = new MyCheckbox ();
        ob.setTitle ("Checkbox Demo");
        ob.setSize (500,200);
        ob.setVisible (true);
    }
    public void paint (Graphics g)
    {
        g.drawString ("Checkbox state:", 20, 100);
        msg = "Bold:" + cb1.getState() ;
        g.drawString (msg, 20, 120);
        msg = "Italic:" + cb2.getState();
        g.drawString (msg, 20, 160);
    }
}

```

Output:

Radio Button: A Radio button represents a round shaped button such that only one can be selected from a panel. Radio button can be created using CheckboxGroup class and Checkbox classes.

- To create a radio button: `CheckboxGroup cbg = new CheckboxGroup ();`
`Checkbox cb = new Checkbox ("label", cbg, true);`
- To know the selected checkbox: `Checkbox cb = cbg.getSelectedCheckbox ();`
- To know the selected checkbox label: `String label = cbg.getSelectedCheckbox ().getLabel ();`

Program 9: Write a program to create Radio Button.

//Program to create a Radio Button

```

import java.awt.*;
import java.awt.event.*;
class MyRadio extends Frame implements ItemListener
{
    String msg = "";

```

```

CheckboxGroup cbg;
Checkbox y, n;
MyRadio ()
{
    setLayout (new FlowLayout());
    cbg = new CheckboxGroup ();
    y = new Checkbox ("YES", cbg, true);
    n = new Checkbox ("NO", cbg, false);
    add (y);
    add (n);
    y.addItemListener (this);
    n.addItemListener (this);
    addWindowListener (new WindowAdapter()
    {
        public void windowClosing (WindowEvent we)
        {
            System.exit (0);
        }
    });
}
public void itemStateChanged (ItemEvent ie)
{
    repaint ();
}
public void paint(Graphics g)
{
    g.drawString ("You selected: " + cbg.getSelectedCheckbox ().getLabel () , 100, 50);
}
public static void main(String args[])
{
    MyRadio ob = new MyRadio ();
    ob.setTitle ("Radio Buttons");
    ob.setSize (600,200);
    ob.setVisible (true);
}
}

```

Output:

Choice Menu: Choice menu is a popdown list of items. Only one item can be selected.

- To create a choice menu: `Choice ch = new Choice();`
- To add items to the choice menu: `ch.add ("text");`
- To know the name of the item selected from the choice menu: `String s = ch.getSelectedItem ();`
- To know the index of the currently selected item: `int i = ch.getSelectedIndex();`
This method returns -1, if nothing is selected.

Program 10: Write a program to create a choice box.

```
//choice box
import java.awt.*;
import java.awt.event.*;
class MyChoice extends Frame implements ItemListener
{
    Choice ch;
    MyChoice ()
    {
        setLayout (new FlowLayout ());
        ch = new Choice();
        ch.add ("India");
        ch.add ("Pakistan");
        ch.add ("Afghanistan");
        ch.add ("China");
        ch.add ("Sri Lanka");
        ch.add ("Bangladesh");
        add (ch);
        ch.addItemListener (this);
        addWindowListener (new WindowAdapter()
        {
            public void windowClosing (WindowEvent we)
            {
                System.exit (0);
            }
        });
    }
    public void itemStateChanged (ItemEvent ie)
    {
        repaint ();
    }
    public void paint (Graphics g)
    {
        g.drawString ("U Selected: ", 20, 100);
        g.drawString (ch.getSelectedItem (), 20, 120);
    }
    public static void main(String args[])
    {
        MyChoice ob = new MyChoice ();
        ob.setSize (500,200);
        ob.setTitle ("Choice Demo");
    }
}
```

```

        ob.setVisible (true);
    }
}

```

Output:

List box: A List box is similar to a choice box, it allows the user to select multiple items.

- To create a list box: `List lst = new List();`
(or)
`List lst = new List (3, true);`

This list box initially displays 3 items. The next parameter true represents that the user can select more than one item from the available items. If it is false, then the user can select only one item.

- To add items to the list box: `lst.add("text");`
- To get the selected items: `String x[] = lst.getSelectedItems();`
- To get the selected indexes: `int x[] = lst.getSelectedIndexes ();`

Program 11: Write a program to create a list box.

```

//List box
import java.awt.*;
import java.awt.event.*;
class MyList extends Frame implements ItemListener
{
    List ch;
    int[] msg;
    MyList ()
    {
        setLayout (new FlowLayout());
        ch = new List(3,true);
        ch.add ("India");
        ch.add ("Pakistan");
        ch.add ("Afghanistan");
        ch.add ("China");
        ch.add ("Sri Lanka");
    }
}

```

```

        ch.add ("Bangladesh");
        add (ch);
        ch.addItemListener (this);
        addWindowListener (new WindowAdapter ()
        {
            public void windowClosing (WindowEvent we)
            {
                System.exit (0);
            }
        });
    }
    public void itemStateChanged (ItemEvent ie)
    {
        repaint ();
    }
    public void paint (Graphics g)
    {
        try
        {
            g.drawString ("U Selected:", 20, 150);
            msg = ch.getSelectedIndexes ();
            for (int i =0;i<msg.length; i++)
            {
                String item = ch.getItem (msg[i]);
                g.drawString (item, 100, 150+i*20);
            }
        }catch (NullPointerException ie){}
    }
    public static void main (String args[])
    {
        MyList ob = new MyList ();
        ob.setSize (500,200);
        ob.setTitle ("List Demo");
        ob.setVisible (true);
    }
}

```

Output:

Label: A label is a constant text that is displayed with a text.

- To create a label: `Label l = new Label("text",alignmentConstant);`

Note: - alignmentconstant: `Label.RIGHT`, `Label.LEFT` and `Label.CENTER`

TextField: `TextField` allows a user to enter a single line of text.

- To create a `TextField`: `TextField tf = new TextField(25);`
(or)
`TextField tf = new TextField ("defaulttext", 25);`
- To get the text from a `TextField`: `String s = tf.getText();`
- To set the text into a `TextField`: `tf.setText("text");`
- To hide the text being typed into the `TextField` by a character: `tf.setEchoChar('char');`

TextArea: `TextArea` is similar to a `TextField`, but it accepts more than one line of text.

- To create a `TextArea`: `TextArea ta = new TextArea();`
(or)
`TextArea ta = new TextArea (rows, cols);`

Note: `TextArea` supports `getText ()` and `setText ()`

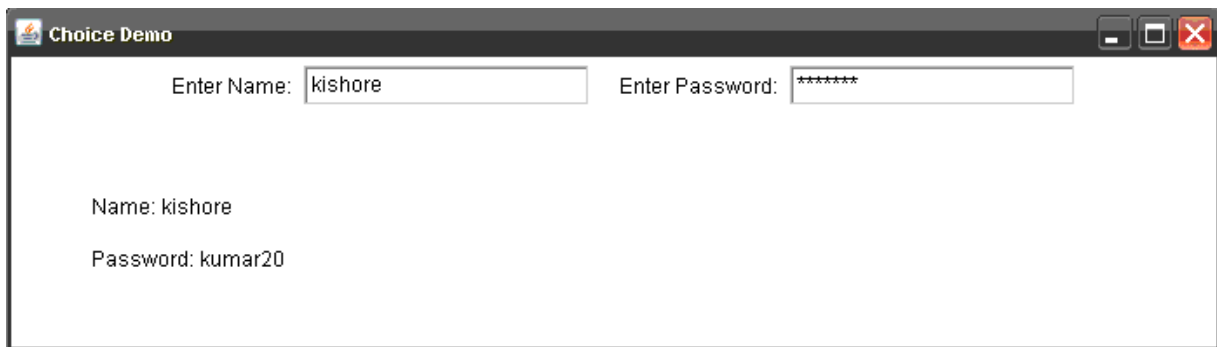
Program 12: Write a program to create label and textboxes.

```
//Labels and TextFields
import java.awt.*;
import java.awt.event.*;
class MyText extends Frame implements ActionListener
{
    Label n, p;
    TextField name, pass;
    MyText ()
    {
        setLayout (new FlowLayout ());
        n = new Label ("Enter Name:", Label.RIGHT);
        p = new Label ("Enter Password:", Label.RIGHT);
        name = new TextField(20);
        pass = new TextField(20);
        pass.setEchoChar ('*');
        add (n);      add (name);
        add (p);      add (pass);
        name.addActionListener (this);
        pass.addActionListener (this);
        addWindowListener (new WindowAdapter ()
        {
            public void windowClosing (WindowEvent we)
            {
                System.exit (0);
            }
        });
    }
    public void actionPerformed (ActionEvent ae)
    {
        repaint ();
    }
}
```

```

    public void paint (Graphics g)
    {
        g.drawString ("Name: " + name.getText (), 50, 120);
        g.drawString ("Password: " + pass.getText (), 50, 150);
    }
    public static void main (String args[])
    {
        MyText ob = new MyText ();
        ob.setSize (500,200);
        ob.setTitle ("Choice Demo");
        ob.setVisible (true);
    }
}

```



Scrollbar Class: Scrollbar class is useful to create scrollbars that can be attached to a frame or text area. Scrollbars can be arranged vertically or horizontally.

- To create a scrollbar : Scrollbar sb = new Scrollbar (alignment, start, step, min, max);
 alignment: Scrollbar.VERTICAL, Scrollbar.HORIZONTAL
 start: starting value (e.g. 0)
 step: step value (e.g. 30) // represents scrollbar length
 min: minimum value (e.g. 0)
 max: maximum value (e.g. 300)
- To know the location of a scrollbar: int n = sb.getValue ();
- To update scrollbar position to a new position: sb.setValue (int position);
- To get the maximum value of the scrollbar: int x = sb.getMaximum ();
- To get the minimum value of the scrollbar: int x = sb.getMinimum ();
- To get the alignment of the scrollbar: int x = getOrientation ();
 This method return 0 if the scrollbar is aligned HORIZONTAL, 1 if aligned VERTICAL.

Program 13: Write a program to create a vertical scrollbar with scroll button length 30 px and with the starting and ending positions ranging from 0 to 400 px.

```

//Creating a vertical scrollbar
import java.awt.*;

```

```
import java.awt.event.*;
class MyScroll extends Frame implements AdjustmentListener
{
    String msg = "";
    Scrollbar sl;
    MyScroll ()
    {
        setLayout (null);
        sl = new Scrollbar (Scrollbar.VERTICAL, 0, 30, 0, 400);
        sl.setBounds (250, 50, 30, 200);
        add (sl);
        sl.addAdjustmentListener (this);
        addWindowListener (new WindowAdapter ()
        {
            public void windowClosing (WindowEvent we)
            {
                System.exit (0);
            }
        });
    }
    public void adjustmentValueChanged (AdjustmentEvent ae)
    {
        repaint ();
    }
    public void paint (Graphics g)
    {
        g.drawString ("SCROLLBAR POSITION: ", 20, 150);
        msg += sl.getValue ();
        g.drawString (msg, 20,180);
        msg = "";
    }
    public static void main (String args[])
    {
        MyScroll ms = new MyScroll ();
        ms.setTitle ("My Scroll Bar");
        ms.setSize (400, 250);
        ms.setVisible (true);
    }
}
```

Output:

The screenshot shows a Windows command prompt window with the title bar "C:\WINDOWS\system32\cmd.exe - java MyScroll". The command prompt shows the following commands and output:

```
D:\JQR>javac MyScroll.java
D:\JQR>java MyScroll
```



Handling Mouse Events: The user may click, release, drag or move a mouse while interacting with the application. If the programmer knows what the user has done, he can write the code according to the mouse event. To trap the mouse events, `MouseListener` and `MouseMotionListener` interfaces of `jav.awt.event` package are used.

`MouseListener` interface contains the following methods:

Method	Description
<code>public void mouseClicked (MouseEvent me)</code>	This method is invoked when the mouse button has been clicked on a component.
<code>public void mousePressed (MouseEvent me)</code>	This method is invoked when the mouse button has been pressed on a component.
<code>public void mouseReleased (MouseEvent me)</code>	This method is invoked when the mouse button has been released on a component.
<code>public void mouseEntered (MouseEvent me)</code>	This method is invoked when the mouse enters a component.
<code>public void mouseExited (MouseEvent me)</code>	This method is invoked when the mouse exits a component.

`MouseMotionListener` interface contains the following methods:

Method	Description
<code>public void mouseDragged (MouseEvent me)</code>	This method is invoked when the mouse button is pressed on a component and then dragged.
<code>public void mouseMoved (MouseEvent me)</code>	This method is invoked when the mouse cursor has been moved onto a component but no buttons have been pushed.

The `MouseEvent` class has the following methods:

Method	Description
<code>int getButton ()</code>	This method returns a value representing a mouse button, when it is clicked it returns 1 if the left button is clicked, 2 if the middle button is clicked and 3 if the right button is clicked.
<code>int getClickCount ()</code>	This method returns the number of mouse clicks associated with this event.
<code>int getX ()</code>	This method returns the horizontal x position of the event relative to the source component.
<code>int getY ()</code>	This method returns the vertical y position of the event relative to the source component.

Program 14: Write a program to demonstrate the mouse events.

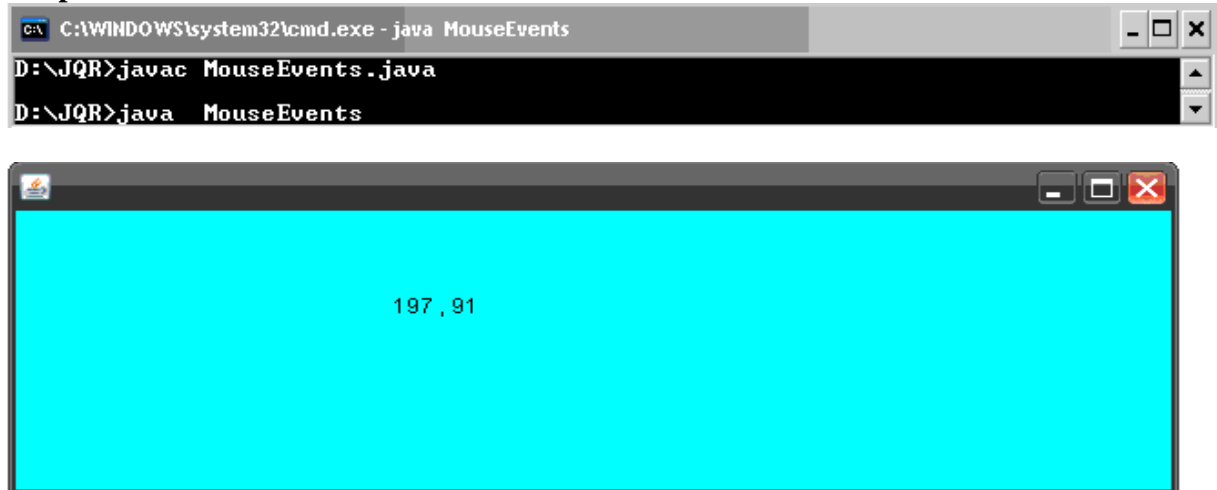
```
//Mouse Events
import java.awt.*;
import java.awt.event.*;
class MouseEvents extends Frame implements MouseListener, MouseMotionListener
{
    int x,y;
    MouseEvents ()
    {
        setLayout (new FlowLayout());
        addMouseListener (this);
        addMouseMotionListener (this);
        addWindowListener (new WindowAdapter ()
        {
            public void windowClosing (WindowEvent we)
            {
                System.exit (0);
            }
        });
    }
    public void paint (Graphics g)
    {
        g.drawString (x + " , " + y, x, y);
    }
    public void mouseClicked (MouseEvent me)
    {
        setBackground (Color.yellow);
    }
    public void mousePressed(MouseEvent me)
    {
        setBackground (Color.red);
    }
    public void mouseReleased(MouseEvent me)
    {
        setBackground (Color.green);
    }
    public void mouseEntered (MouseEvent me)
    {
        setBackground (Color.blue);
    }
    public void mouseExited(MouseEvent me)
    {
        setBackground (Color.white);
    }
    public void mouseDragged(MouseEvent me)
    {
        setBackground (Color.black);
    }
    public void mouseMoved(MouseEvent me)
    {
        x = me.getX ();      y = me.getY ();
        setBackground (Color.cyan);
        repaint ();
    }
    public static void main(String args[])
    {
        MouseEvents ob = new MouseEvents ();
        ob.setSize(600,200);
        ob.setVisible(true);
    }
}
```



```

    }
}

```

Output:

Handling Keyboard Events: A user interacts with the application by pressing keys on the keyboard. A programmer should know which key the user has pressed on the keyboard. These are also called 'events'. Knowing these events will enable the programmer to write his code according to the key pressed. `KeyListener` interface of `java.awt.event` package helps to know which key is pressed or released by the user. It has 3 methods:

Method	Description
<code>public void keyPressed (KeyEvent ke)</code>	This method is called when a key is pressed on the keyboard. This include any key on the keyboard along with special keys like function keys, shift, alter, caps lock, home and end etc.
<code>public void keyTyped (KeyEvent ke)</code>	This method is called when a key is typed on the keyboard. This is ame as <code>keyPressed ()</code> method but this method is called when general keys like A to Z or 1 to 9 etc are typed. It cannot work with special keys.
<code>public void keyReleased (KeyEvent ke)</code>	This method is called when a key is released.

`KeyEvent` class has the following methods to know which key is typed by the user:

Method	Description
<code>char getKeyChar ()</code>	This method returns the key name (or character) related to the key pressed or released.
<code>int getKeyCode ()</code>	This method returns an integer number which is the value of the key pressed by the user.

The following are the key codes for the keys on the keyboard. They are defined as constants in `KeyEvent` class. Remember `VK` represents Virtual Key.

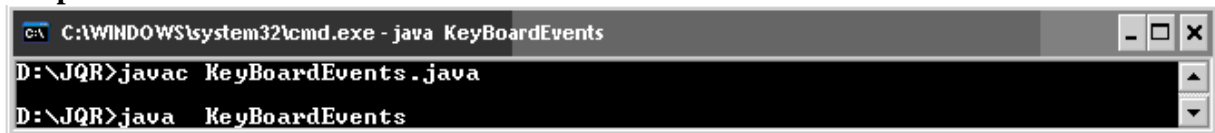
Key Code	Description
<code>VK_A</code> to <code>VK_Z</code>	To represent keys from a to z
<code>VK_0</code> to <code>VK_9</code>	To represent keys from 0 to 9
<code>VK_F1</code> to <code>VK_F12</code>	To represent keys from F1 to F12

VK_HOME, VK_END	To represent home, end keys
VK_PAGE_UP, VK_PAGE_DOWN	To represent pageup, pagedown keys
VK_INSERT, VK_DELETE	To represent insert, Delete keys
VK_CAPS_LOCK, VK_ALT, VK_ESCAPE	To represent capslock, alter, escape keys
VK_CONTROL, VK_SHIFT, VK_TAB	To represent control, shift, tab keys
VK_LEFT, VK_RIGHT, VK_UP, VK_DOWN	To represent arrow keys
static String getKeyText (int keyCode)	This method returns a string describing the keyCode such as HOME, F1 or A

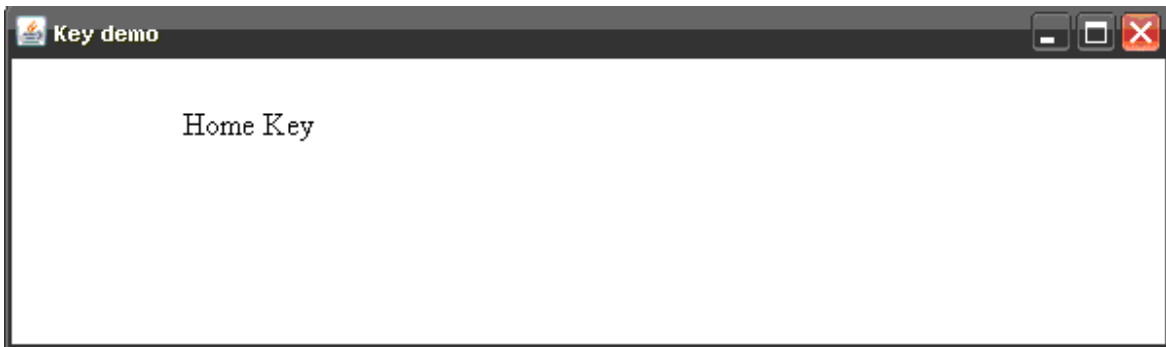
Program 15: Write a program to trap a key which is pressed on the keyboard.

```
//Keyboard events
import java.awt.*;
import java.awt.event.*;
class KeyBoardEvents extends Frame implements KeyListener
{
    String str=" ";
    KeyBoardEvents ()
    {
        addKeyListener (this);
        addWindowListener (new WindowAdapter ()
        {
            public void windowClosing (WindowEvent we)
            {
                System.exit (0);
            }
        });
    }
    public void paint (Graphics g)
    {
        g.drawString (str, 100, 100);
        str=" ";
    }
    public void keyPressed (KeyEvent ke)
    {
        int keyCode = ke.getKeyCode();
        if ( keyCode == KeyEvent.VK_F1)
            str += "F1 key";
        if ( keyCode == KeyEvent.VK_HOME)
            str += "Home key";
        repaint ();
    }
    public void keyReleased (KeyEvent ke)
    {
    }
    public void keyTyped (KeyEvent ke)
    {
    }
    public static void main (String args[])
    {
        KeyBoardEvents ob = new KeyBoardEvents ();
        ob.setTitle ("Key demo");
        ob.setSize (600,450);
    }
}
```

```
        ob.setVisible (true);  
    }  
}
```

Output:

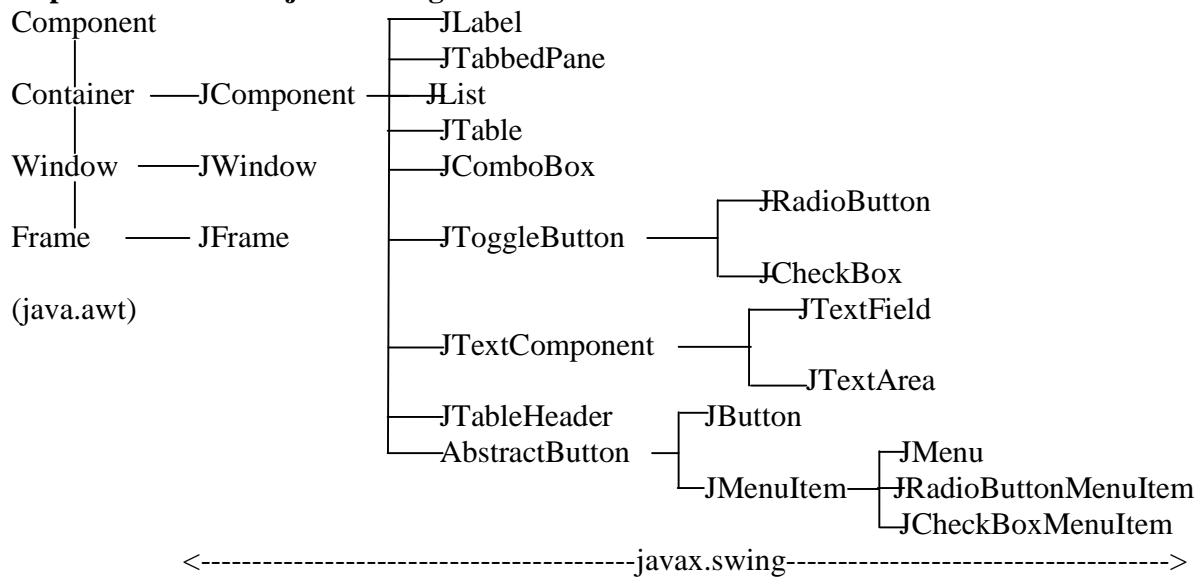
```
C:\WINDOWS\system32\cmd.exe - java KeyBoardEvents  
D:\JQR>javac KeyBoardEvents.java  
  
D:\JQR>java KeyBoardEvents
```



22. Swings

When awt components are created, internally native methods (c language functions) are executed which create peer component (equivalent component). awt components are heavy weight components. The look and feel of awt component change depending upon the operating system. JFC represents a class library that is extended from awt. JFC stands for Java Foundation Classes. JFC is an extension of the original awt, it does not replace awt. JFC components are light weight components, they take less system resources.

Important classes of javax.swing:



Creating a Frame:

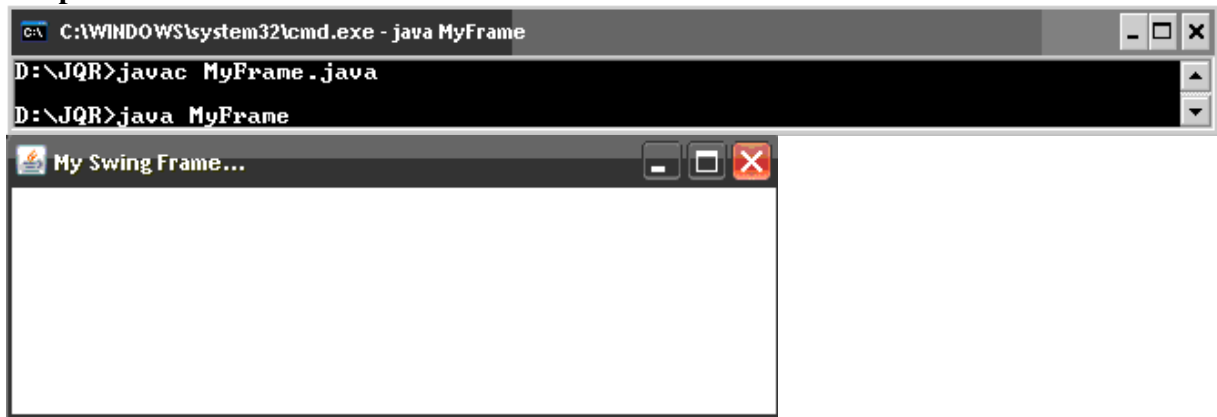
Create an object to JFrame: `JFrame ob = new JFrame ("title");`
(or)

Create a class as subclass to JFrame `class: MyFrame extends JFrame`

Create an object to that class : `MyFrame ob = new MyFrame ();`

Program 1: Write a program to create a frame by creating an object to JFrame class.

```
//A swing Frame
import javax.swing.*;
class MyFrame
{
    public static void main (String args[])
    {
        JFrame jf = new JFrame ("My Swing Frame...");
        jf.setSize (400,200);
        jf.setVisible (true);
        jf.setDefaultCloseOperation (JFrame.EXIT_ON_CLOSE);
    }
}
```

Output:

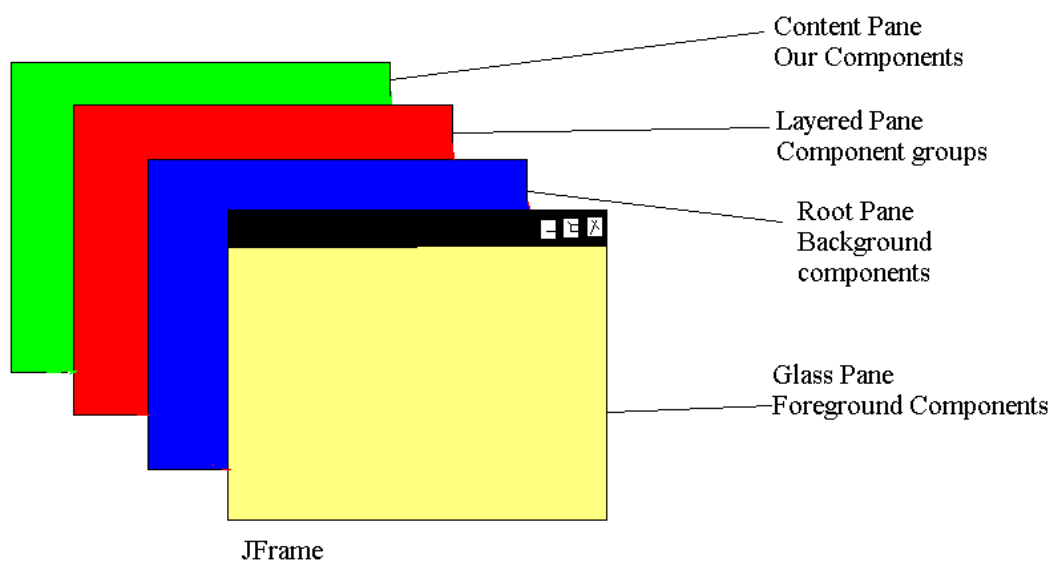
Note: To close the frame, we can take the help of `getDefaultCloseOperation ()` method of `JFrame` class, as shown here:

```
getDefaultCloseOperation (constant);
```

where the constant can be any one of the following:

- `JFrame.EXIT_ON_CLOSE`: This closes the application upon clicking on close button.
- `JFrame.DISPOSE_ON_CLOSE`: This disposes the present frame which is visible on the screen. The JVM may also terminate.
- `JFrame.DO_NOTHING_ON_CLOSE`: This will not perform any operation upon clicking on close button.
- `JFrame.HIDE_ON_CLOSE`: This hides the frame upon clicking on close button.

Window Panes: In swings the components are attached to the window panes only. A window pane represents a free area of a window where some text or components can be displayed. For example, we can create a frame using `JFrame` class in `javax.swing` which contains a free area inside it, this free area is called 'window pane'. Four types of window panes are available in `javax.swing` package.



- **Glass Pane:** This is the first pane and is very close to the monitors screen. Any components to be displayed in the foreground are attached to this glass pane. To reach this glass pane, we use `getGlassPane ()` method of `JFrame` class.
- **Root Pane:** This pane is below the glass pane. Any components to be displayed in the background are displayed in this pane. Root pane and glass pane are used in animations also. For example, suppose we want to display a flying aeroplane in the sky. The aeroplane can be displayed as a .gif or .jpg file in the glass pane whereas the blue sky can be displayed in the root pane in the background. To reach this root pane, we use `getRootPane ()` method of `JFrame` class.
- **Layered Pane:** This pane lies below the root pane. When we want to take several components as a group, we attach them in the layered pane. We can reach this pane by calling `getLayeredPane ()` method of `JFrame` class.
- **Content Pane:** This is the bottom most pane of all. Individual components are attached to this pane. To reach this pane, we can call `getContentPane ()` method of `JFrame` class.

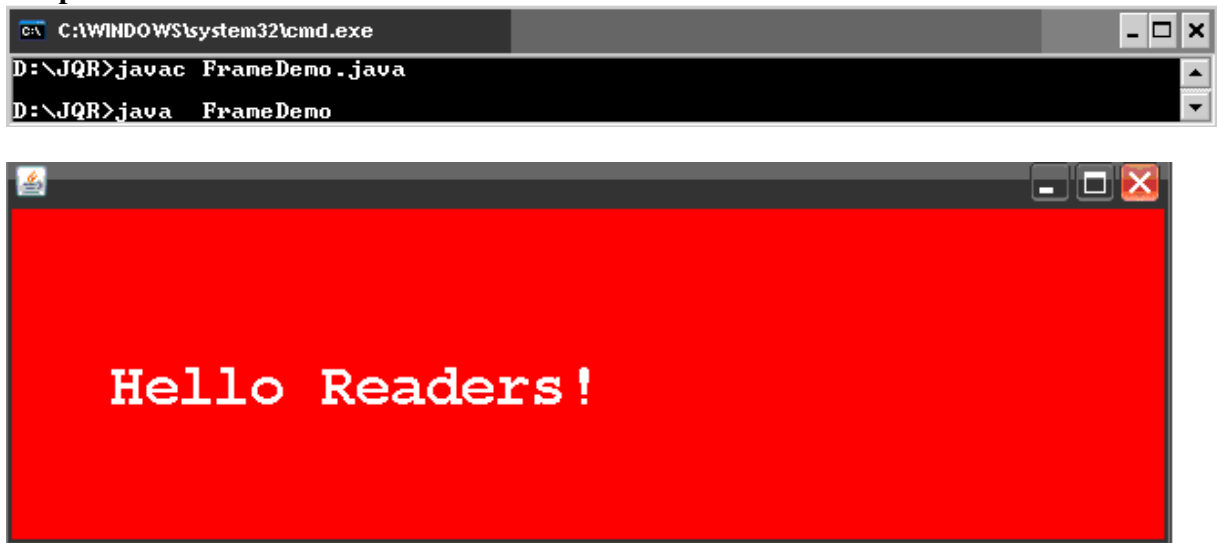
Displaying Text in the Frame: `paintComponent (Graphics g)` method of `JPanel` class is used to paint the portion of a component in swing. We should override this method in our class. In the following example, we are writing our class `MyPanel` as a subclass to `JPanel` and override the `paintComponent ()` method.

Program 2: Write a program to display text in the frame.

```
import javax.swing.*;
import java.awt.*;

class MyPanel extends JPanel
{
    public void paintComponent (Graphics g)
    {
        super.paintComponent (g); //call JPanel's method
        setBackground (Color.red);
        g.setColor (Color.white);
        g.setFont (new Font("Courier New",Font.BOLD,30));
        g.drawString ("Hello Readers!", 50, 100);
    }
}

class FrameDemo extends JFrame
{
    FrameDemo ()
    {
        Container c = getContentPane ();
        MyPanel mp = new MyPanel ();
        c.add (mp);
        setDefaultCloseOperation (JFrame.EXIT_ON_CLOSE);
    }
    public static void main(String args[])
    {
        FrameDemo ob = new FrameDemo ();
        ob.setSize (600, 200);
        ob.setVisible (true);
    }
}
```

Output:

JComponent Class Methods: JComponent class of javax.swing package is the subclass of the Component class of java.awt. So, whatever the methods are available in Component class are also available to JComponent. This is the reason why almost all the methods of AWT are useful in swing also. Additional methods found in JComponent are also applicable for the components created in swing.

When a component is created, to display it in the frame, we should not attach it to the frame directly as we did in AWT. On the other hand, the component should be attached to a window pane.

- To add the component to the content pane, we can write, as:
`c.add (component);` where c represents the content pane which is represented by Container object.
- Similarly, to remove the component we can use remove () method as:
`c.remove (component);`
- To remove all the components from the content pane we can use removeAll () method as:
`c.removeAll ();`
- When components are to be displayed in the frame, we should first set a layout which arranges the components in a particular manner in the frame as:
`c.setLayout (new FlowLayout ());`

The following methods of JComponent class are very useful while handling the components:

- To set some background color to the component, we can use setBackground () method, as:
`component.setBackground (Color.yellow);`
- To set the foreground color to the component, we can use setForeground () method, as:
`component.setForeground (Color.red);`
- To set some font for the text displayed on the component, we can use setFont () method. We should pass Font class object to this method, as:
`component.setFont (Font obj);`

where, Font class object can be created as:

`Font obj = new Font ("fontname", style, size);`

- To set the tool tip text, we can use `setToolTipText ()` method, as:
`component.setToolTipText ("This is for help");`
- To set a mnemonic, we can use `setMnemonic ()` method, as:
`component.setMnemonic ('c');`
- To enable or disable a component, we can use `setEnabled ()` method, as:
`component.setEnabled (true);`
- To make a component to be visible or invisible, we can use `setVisible ()` method, as:
`component.setVisible (true);`
- To know the current height of the component, use `getHeight ()` method, as:
`component.getHeight ();`
- To know the current width of the component, use `getWidth ()` method, as:
`component.getWidth ();`
- To know the current x coordinate of the component
`component.getX ();`
- To know the current y coordinate of the component
`component.getY ();`
- To set the location of the component in the frame, we can use `setBounds ()` method, as:
`component.setBounds (x, y, width, height);`

PushButton:

- To create a JButton with text: `JButton b = new JButton ("OK");`
- To create a JButton with image: `JButton b = new JButton (ImageIcon ii);`
- To create a JButton with text & image: `JButton b = new JButton ("OK", ImageIcon ii);`

It is possible to create components in swing with images on it. The image is specified by `ImageIcon` class object.

Program 3: Write a program to create a JButton component.

```
//JButton Demo
import javax.swing.*;
import java.awt.*;
class JButton1 extends JFrame
{
    JButton b;
    ImageIcon ii;
    JButton1 ()
    {
        Container c = getContentPane ();
        c.setLayout (null);
        ii = new ImageIcon("abc.gif");
        b = new JButton ("Click ME", ii);
        b.setBounds(100,20,150,150);
        b.setBackground (Color.yellow);
        b.setForeground (Color.red);
        c.add (b);
        setDefaultCloseOperation (JFrame.EXIT_ON_CLOSE);
    }
    public static void main(String args[])
    {
        JButton1 ob = new JButton1 ();
```



```

        ob.setTitle ("Swing Frame example");
        ob.setSize (600,250);
        ob.setVisible (true);
    }
}

```

Output:**Label:**

- To create a JLabel component: `JLabel lbl = new JLabel ();`
- To set text to JLabel component: `lbl.setText ("Hello");`
- To create a JLabel with text: `JLabel lbl = new JLabel ("text", JLabel.RIGHT);`
- To create a JLabel with image: `JLabel lbl = new JLabel (ImageIcon ii);`

Program 4: Write a program to create a JLabel component.

```

//JLabelDemo
import javax.swing.*;
import java.awt.*;
class JLabelDemo extends JFrame
{
    JLabel lbl;
    ImageIcon ii;
    JLabelDemo ()
    {
        Container c = getContentPane ();
        c.setLayout (null);
        ii = new ImageIcon("abc.gif");
        lbl = new JLabel(ii);
        lbl.setBounds (100,100,200,200);
        c.add (lbl);
        setDefaultCloseOperation (JFrame.EXIT_ON_CLOSE);
    }
}

```

```

    }
    public static void main(String args[])
    {
        JLabelDemo ob = new JLabelDemo ();
        ob.setTitle ("Swing Frame example");
        ob.setSize (600,250);
        ob.setVisible (true);
    }
}

```

Output:**Creating Components in Swing:**

- To create a checkbox: `JCheckBox cb = new JCheckBox ("Java", true);`
- To create a RadioButton: `JRadioButton rb = new JRadioButton ("Male", true);`
 To tell the JVM that some radio buttons belongs to same group we should use ButtonGroup object. if rb1,rb2,rb3 are RadioButtons then

```

      ButtonGroup bg = new ButtonGroup ();
      bg.add (rb1);
      bg.add (rb2);
      bg.add (rb3);

```
- To create a textfield: `JTextField tf = new JTextField (20);`
- To create a text area: `JTextArea ta = new JTextArea (5, 20);`

Program 5: Write a program to create check boxes, radio buttons and text area components.

```

//Check boxes and radio buttons
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;

```

class CheckRadio extends JFrame implements ActionListener

```
{
    JCheckBox cb1,cb2;
    JRadioButton rb1,rb2;
    ButtonGroup bg;
    JTextArea ta;
    String msg = "";
    CheckRadio ()
    {
        Container c = getContentPane ();
        c.setLayout (new FlowLayout ());
        cb1 = new JCheckBox ("J2SE", true);
        cb2 = new JCheckBox ("J2EE");
        rb1 = new JRadioButton ("Male", true);
        rb2 = new JRadioButton ("Female");
        bg = new ButtonGroup();
        bg.add (rb1);
        bg.add (rb2);
        ta = new JTextArea (5, 20);
        c.add (cb1);
        c.add (cb2);
        c.add (rb1);
        c.add (rb2);
        c.add (ta);
        cb1.addActionListener (this);
        cb2.addActionListener (this);
        rb1.addActionListener (this);
        rb2.addActionListener (this);
        setDefaultCloseOperation (JFrame.EXIT_ON_CLOSE);
    }
    public void actionPerformed (ActionEvent ae)
    {
        if (cb1.getModel ().isSelected ())           //to know the user selection
            msg += "J2SE";
        if (cb2.getModel ().isSelected ())
            msg += "J2EE";
        if (rb1.getModel ().isSelected ())
            msg += "Male";
        if (rb2.getModel ().isSelected () )
            msg += "Female";
        ta.setText (msg);
        msg = "";
    }
    public static void main(String args[])
    {
        CheckRadio ob = new CheckRadio ();
        ob.setTitle ("Swing Frame example");
        ob.setSize (600,250);
        ob.setVisible (true);
    }
}
```

}

Output:

JComboBox class: JComboBox allows us to create a combo box, with a group of items which are displayed as a drop down list. The user can select a single item only.

- To create an empty combo box: `JComboBox box = new JComboBox ();`
- To create a combo box with array of elements:
`JComboBox box = new JComboBox (Object arr[]);`
- To create a combo box with vector of elements:
`JComboBox box = new JComboBox (Vector v);`
- To add the items to the combo box: `box.addItem (" India");`
- To retrieve the selected item from the combo box: `Object obj = box.getSelectedItem ();`
- To retrieve the selected items index: `int i = box.getSelectedIndex ();`
- To get the item of combo box by giving index: `Object obj = box.getItemAt (int index);`
- To get number of items in the combo box: `int n = box.getItemCount ();`
- To remove an item obj from the combo box: `box.removeItem (Object obj);`
- To remove an item by giving index: `box.removeItemAt (int index);`
- To remove all items: `box.removeAllItems ();`

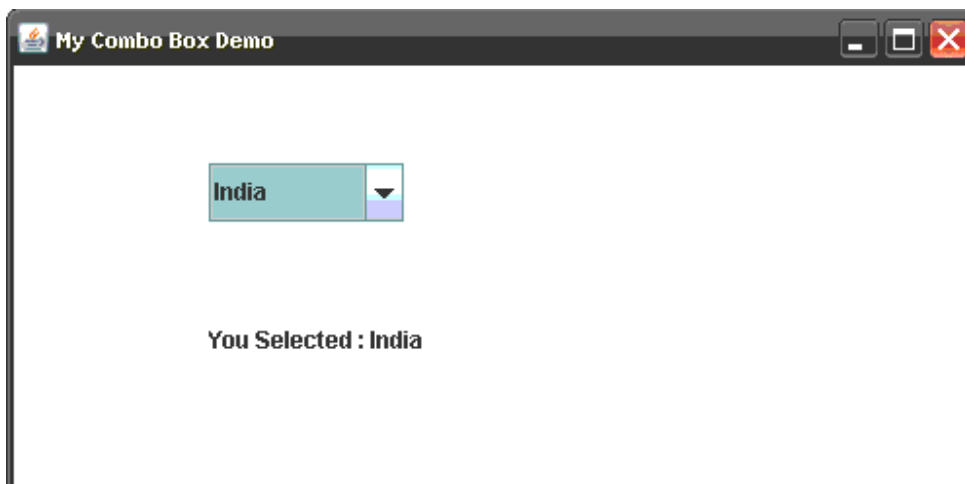
Program 6: Write a program to create a combo box with names of some countries.

```
//JComboBox Demo
import java.awt.*;
import javax.swing.*;
import java.awt.event.*;
class JComboBoxDemo extends JFrame implements ItemListener
{
    JComboBox box;
    JLabel lbl;
    JComboBoxDemo ()
    {
        Container c = getContentPane();
        c.setLayout (null);
        box = new JComboBox ();
        box.addItem ("India");
    }
}
```

```

        box.addItem ("Pakistan");
        box.addItem ("Afghanistan");
        box.addItem ("China");
        box.addItem ("Sri Lanka");
        box.addItem ("Bangladesh");
        box.setBounds (100, 50, 100, 30);
        c.add (box);
        box.addItemListener (this);
        lbl = new JLabel ();
        lbl.setBounds (100, 120,200,40);
        c.add(lbl);
        setDefaultCloseOperation (JFrame.EXIT_ON_CLOSE);
    }
    public void itemStateChanged (ItemEvent ie)
    {
        String str = (String) box.getSelectedItem ();
        lbl.setText ("You Selected : " + str);
    }
    public static void main(String args[])
    {
        JComboBoxDemo ob = new JComboBoxDemo ();
        ob.setSize (500,250);
        ob.setTitle ("My Combo Box Demo");
        ob.setVisible (true);
    }
}

```

Output:

JList Class: JList class is useful to create a list which displays a list of items and allows the user to select one or more items.

- To create an empty list: `JList lst = new JList();`
- To create a list with the array of elements: `JList lst = new JList (Object arr []);`
- To create a list with vector of elements: `JList lst = new JList (Vector v);`
- To know which item is selected in the list: `Object item = lst.getSelectedValue ();`
- To know the selected items index: `int index = lst.getSelectedIndex ();`
- To get all the selected items into an array: `Object arr [] = lst.getSelectedValues ();`
- To get the indexes of all selected items: `int arr[] = lst.getSelectedIndices ();`

Program 7: Write a program to create a list box with names of some countries such that user can select one or more items.

//JList Demo

```
import java.awt.*;
import javax.swing.*;
import javax.swing.event.*;
class JListDemo extends JFrame implements ListSelectionListener
{
    JList lst;
    JLabel lbl;
    Object arr [];
    String msg="";
    JListDemo ()
    {
        Container c = getContentPane();
        c.setLayout (null);
        String items [ ] = { "India","America","Germany","Japan","France" };
        lst = new JList (items);
        lst.setBounds(100,20,100,100);
        c.add (lst);
        lst.addListSelectionListener (this);
        lbl = new JLabel ();
        lbl.setBounds (100, 150,200,40);
        c.add(lbl);
        setDefaultCloseOperation (JFrame.EXIT_ON_CLOSE);
    }
    public void valueChanged (ListSelectionEvent le)
    {
        arr= lst.getSelectedValues ();
        for ( int i=0;i<arr.length;i++)
            msg += (String) arr [i];
        lbl.setText ("You Selected : " + msg);
        msg = "";
    }
    public static void main(String args[])
    {
        JListDemo ob = new JListDemo ();
        ob.setSize (500,250);
        ob.setTitle ("JListDemo");
        ob.setVisible (true);
    }
}
```

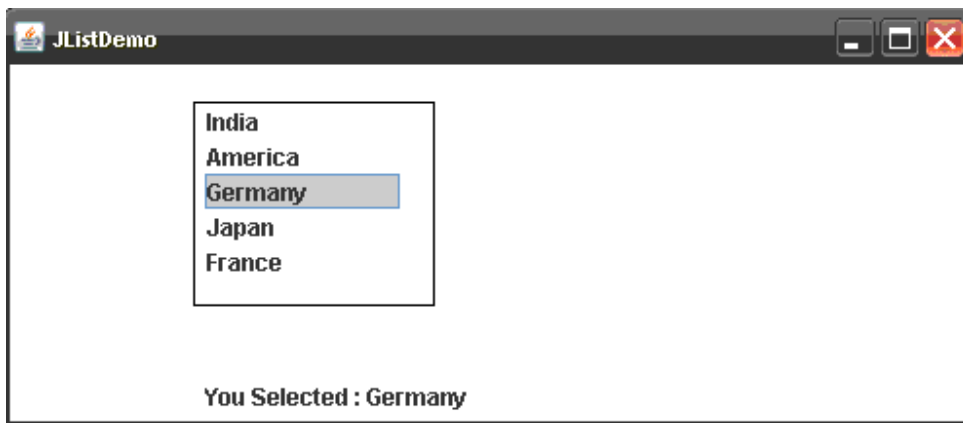
}

Output:

```

C:\WINDOWS\system32\cmd.exe - java JListDemo
D:\JQR>javac JListDemo.java
D:\JQR>java JListDemo

```



JTabbedPane: JTabbedPane is a container to add multiple components on every tab. The user can choose a component from a tab.

- To create a JTabbedPane: `JTabbedPane jtp = new JTabbedPane ();`
- To add tabs: `jtp.addTab ("title", object);`
- To create a Panel containing some components: `class MyPanel extends JPanel`
Now pass 'MyPanel' class object to `addTab ()`
- To remove a tab (and its components) from the tabbedpane: `jtp.removeTabAt (int index);`
- To remove all the tabs and their corresponding components: `jtp.removeAll ();`

Program 8: Write a program to create a tabbed pane with two tab sheets.

```

//JTabbedPane demo
import java.awt.*;
import javax.swing.*;
class JTabbedPaneDemo extends JFrame
{
    JTabbedPaneDemo ()
    {
        JTabbedPane jtp = new JTabbedPane ();
        jtp.add ("Countries", new CountriesPanel ());
        jtp.add ("Capitals", new CapitalsPanel ());
        Container c = getContentPane ();
        c.add (jtp);
    }
    public static void main(String args[])
    {
        JTabbedPaneDemo demo = new JTabbedPaneDemo ();
        demo.setSize (600,450);
        demo.setVisible (true);
        demo.setDefaultCloseOperation (JFrame.EXIT_ON_CLOSE);
    }
}

```

```

    }
}
class CountriesPanel extends JPanel
{
    CountriesPanel ()
    {
        JButton b1, b2, b3;
        b1 = new JButton ("America");
        b2 = new JButton ("India");
        b3 = new JButton ("Japan");
        add (b1);    add (b2);    add (b3);
    }
}
class CapitalsPanel extends JPanel
{
    CapitalsPanel ()
    {
        JCheckBox cb1 = new JCheckBox ("Washington");
        JCheckBox cb2 = new JCheckBox ("New Delhi");
        JCheckBox cb3 = new JCheckBox ("Tokyo");
        add (cb1);    add (cb2);    add (cb3);
    }
}
}

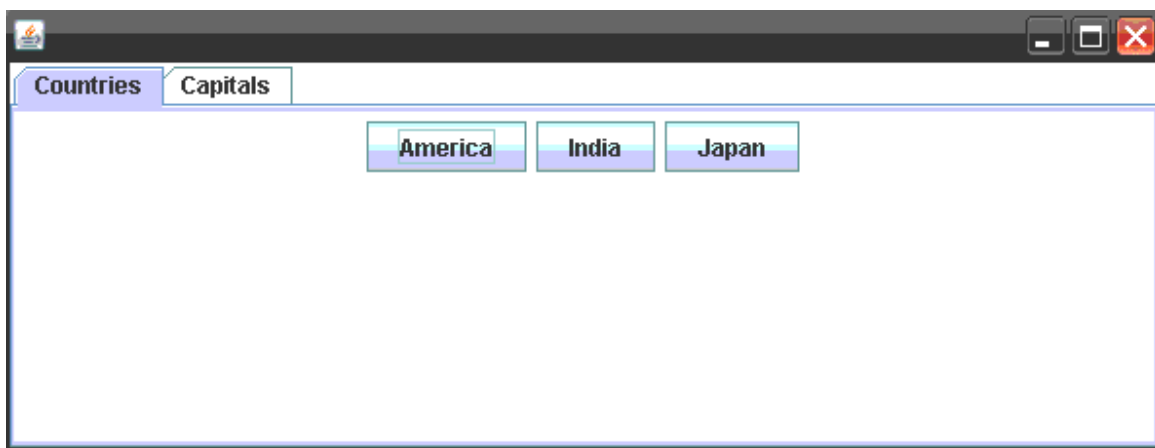
```

Output:

```

C:\WINDOWS\system32\cmd.exe - java JTabbedPaneDemo
D:\JQR>javac JTabbedPaneDemo.java
D:\JQR>java JTabbedPaneDemo

```



JTable: JTable represents data in the form of a table. The table can have rows of data, and column headings.

- To create a JTable: `JTable tab = new JTable (data, column_names);`
Here, data and column_names can be a 2D array or both can be vector of vectors.
- To create a row using a vector: `Vector row = new Vector();`
`row.add (object); //here object represents a column`
`row.add (object);`


```
row.add (object);
```

- To create a table heading, we use `getTableHeader ()` method of `JTable` class.

```
JTableHeader head = tab.getTableHeader ();
```

Note: `JTableHeader` class is defined in `javax.swing.table` package.

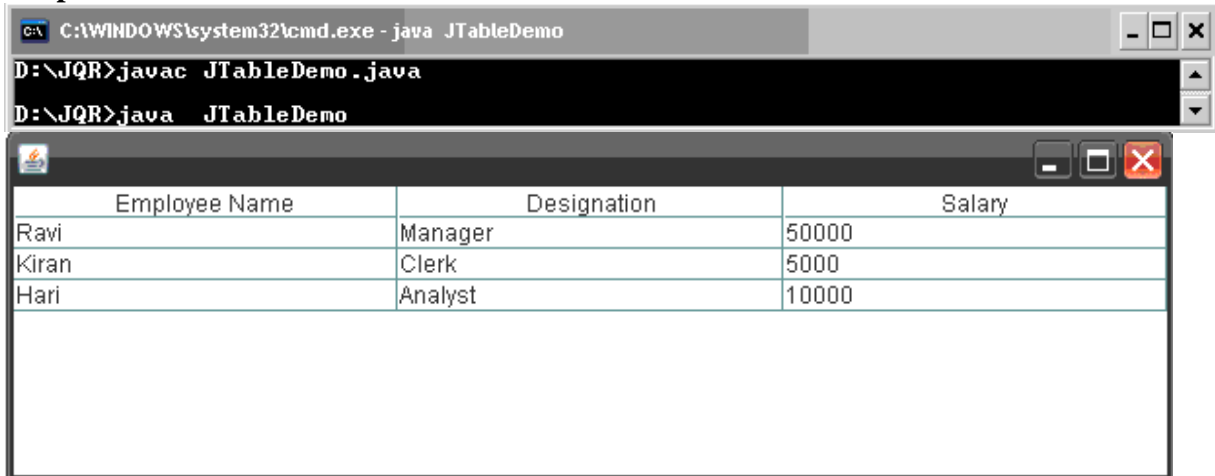
Program 9: Write a program that creates a table with some rows and columns.

```
//creating a table
import java.awt.*;
import javax.swing.*;
import java.util.*;
import javax.swing.table.*;
class JTableDemo extends JFrame
{
    JTableDemo ()
    {
        Vector <Vector> data = new Vector <Vector> ();
        //create first row
        Vector <String> row = new Vector <String> ();
        row.add ("Ravi");
        row.add ("Manager");
        row.add ("50000");
        data.add (row);
        row = new Vector <String>();          //create second row
        row.add ("Kiran");
        row.add ("Clerk");
        row.add ("5000");
        data.add (row);
        row = new Vector<String> ();          //create third row
        row.add ("Hari");
        row.add ("Analyst");
        row.add ("10000");
        data.add (row);
        //create a row with column names
        Vector <String> cols = new Vector<String> ();
        cols.add ("Employee Name");
        cols.add ("Designation");
        cols.add ("Salary");
        //create the table
        JTable tab = new JTable (data, cols);
        //create table header object
        JTableHeader head = tab.getTableHeader ();
        Container c = getContentPane ();
        c.setLayout (new BorderLayout ());
        c.add ("North", head);
        c.add ("Center", tab);
        setDefaultCloseOperation (JFrame.EXIT_ON_CLOSE);
    }
    public static void main(String args[])
```

```

    {
        JTableDemo demo = new JTableDemo ();
        demo.setSize (600, 250);
        demo.setVisible (true);
    }
}

```

Output:

JMenu Class: A menu represents a group of items or options for the user to select. To create a menu, the following steps should be used:

- Create a menu bar using JMenuBar class object: `JMenuBar mb = new JMenuBar ();`
 - Attach this member to the container. `c.add (mb);`
 - Create separate menu to attach to the menubar. `JMenu file = new JMenu ("File");`
Note: Here, "File" is title for the menu which appear in the menubar.
 - Attach this menu to the menubar. `mb.add (file);`
 - Create menuitems using JMenuItem or JCheckBoxMenuItem or JRadioButtonMenuItem classes. `JMenuItem op = new JMenuItem ("Open");`
 - Attach this menuitem to the menu. `file.add (op);`
 - Creating a submenu:
 - Create a menu: `JMenu font = new JMenu ("Font");`
Here, font represents a submenu to be attached to a menu.
 - Now attach it to a menu: `file.add (font);`
 - Attach menuitems to the font submenu. `font.add (obj);`
- Note:** obj can be a JMenuItem or JCheckBoxMenuItem or JRadioButtonMenuItem.

Program 10: Write a program to create a menu with several menu items.

```

//Menu Creation
import java.awt.*;
import javax.swing.*;
import java.awt.event.*;
class MyMenu extends JFrame implements ActionListener
{
    JMenuBar mb;
    JMenu file, edit, font;

```

```

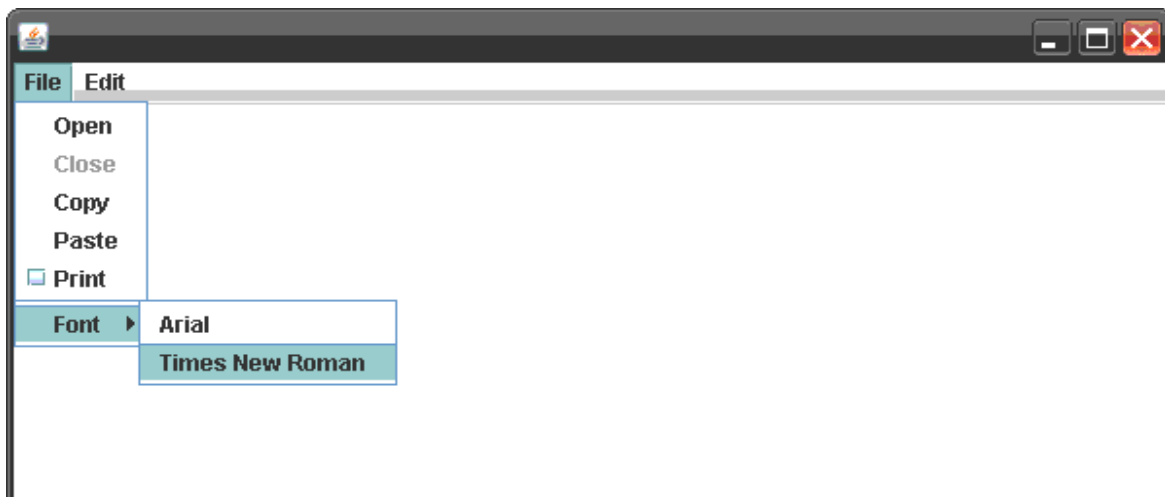
JMenuItem op, cl, cp, pt;
JCheckBoxMenuItem pr;
MyMenu ()
{
    Container c = getContentPane ();
    c.setLayout (new BorderLayout ());
    mb = new JMenuBar ();
    c.add ("North", mb);
    //create file, edit menus
    file = new JMenu ("File");
    edit = new JMenu ("Edit");
    //add menus to mb
    mb.add (file);
    mb.add (edit);
    //create menuitems
    op = new JMenuItem ("Open");
    cl = new JMenuItem ("Close");
    cp = new JMenuItem ("Copy");
    pt = new JMenuItem ("Paste");
    //add menu items to menus
    file.add (op);
    file.add (cl);
    file.add (cp);
    file.add (pt);
    //disable close
    cl.setEnabled (false);
    //create checkbox menu item
    pr = new JCheckBoxMenuItem ("Print");
    //add checkbox menu item to file menu
    file.add (pr);
    //display line (separator)
    file.addSeparator ();
    //create submenu
    font = new JMenu ("Font");
    //add this submenu in file
    file.add (font);
    //add menu items to font
    font.add ("Arial");
    font.add ("Times New Roman");
    //add action listeners to menu items
    op.addActionListener (this);
    cl.addActionListener (this);
    cp.addActionListener (this);
    pt.addActionListener (this);
    pr.addActionListener (this);
    //close the frame
    setDefaultCloseOperation (JFrame.EXIT_ON_CLOSE);

```

```

    }
    public void actionPerformed(ActionEvent ae)
    {
        if (op.isArmed())
            System.out.println ("Open is selected");
        if (cl.isArmed())
            System.out.println ("Close is selected");
        if (cp.isArmed())
            System.out.println ("Copy is selected");
        if (pt.isArmed())
            System.out.println ("Paste is selected");
        if (pr.isArmed())
            System.out.println ("Print is selected");
    }
    public static void main (String args[])
    {
        MyMenu ob = new MyMenu ();
        ob.setSize (600,450);
        ob.setVisible (true);
    }
}

```

Output:

Layout Manager: Layout Manager is an interface that arranges the components on the screen. Layout Manager is implemented in the following classes:

FlowLayout, BorderLayout, CardLayout, GridLayout, GridBagLayout

FlowLayout: FlowLayout is useful to arrange the components in a line after the other. When a line is filled with components, they are automatically placed in the next line.

Program 11: Write a program to create push buttons and arrange them using flow layout.

```
//Flow layout example
import javax.swing.*;
import java.awt.*;
class Example1 extends JFrame
{
    Example1 ()
    {
        Container c = getContentPane ();
        c.setLayout(new FlowLayout(FlowLayout.LEFT,20,40));
        // where 20 and 40 specifies hgap and vgap respectively.
        JButton b1 = new JButton ("First");
        JButton b2 = new JButton ("Second");
        JButton b3 = new JButton ("Third");
        //add(c);
        c.add (b1);    c.add (b2);    c.add (b3);
        setDefaultCloseOperation (JFrame.EXIT_ON_CLOSE);
    }
    public static void main(String args[])
    {
        Example1 ob = new Example1 ();
        ob.setTitle ("Flow Layout...");
        ob.setSize (600,250);
        ob.setVisible (true);
    }
}
```

Output:



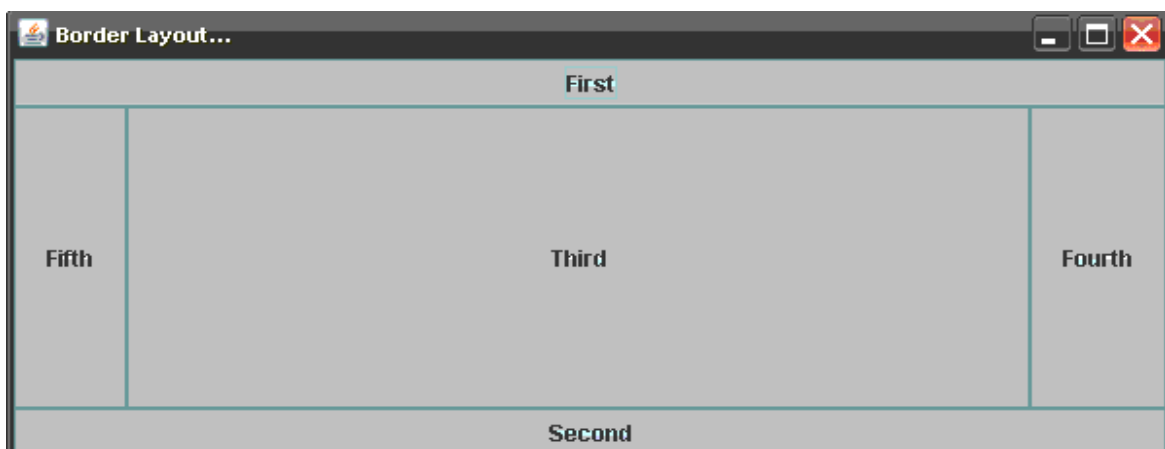
BorderLayout: BorderLayout is useful to arrange the components in the 4 borders of the frame as well as in the center. The borders are specified as South, North, East, West and Center.

Program12: Write a program to create push buttons and arrange them using border layout.

//Border layout example

```
import javax.swing.*;
import java.awt.*;
class Example2 extends JFrame
{
    Example2 ()
    {
        Container c = getContentPane ();
        c.setLayout (new BorderLayout ());
        JButton b1 = new JButton ("First");
        JButton b2 = new JButton ("Second");
        JButton b3 = new JButton ("Third");
        JButton b4 = new JButton ("Fourth");
        JButton b5 = new JButton ("Fifth");
        //add(c);
        c.add (b1,"North");   c.add (b2,"South");   c.add (b3, "Center");
        c.add (b4,"East");    c.add (b5,"West");
        setDefaultCloseOperation (JFrame.EXIT_ON_CLOSE);
    }
    public static void main(String args[])
    {
        Example2 ob = new Example2 ();
        ob.setTitle ("Border Layout...");
        ob.setSize (600,250);
        ob.setVisible (true);
    }
}
```

Output:




CardLayout: A cardLayout is a layout manager which treats each component as a card. Only one card is visible at a time and the container acts as a stack of cards.

Program 13: Write a program to create push buttons and arrange them using border layout.

```
//Card layout example
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
class Example3 extends JFrame implements ActionListener
{
    Container c;
    CardLayout card;
    Example3 ()
    {
        c = getContentPane ();
        card = new CardLayout (50,50);
        c.setLayout (card);
        JButton b1 = new JButton ("First");
        JButton b2 = new JButton ("Second");
        JButton b3 = new JButton ("Third");
        c.add ("button1", b1);
        c.add ("button2", b2);
        c.add ("button3", b3);
        b1.addActionListener (this);
        b2.addActionListener (this);
        b3.addActionListener (this);
        setDefaultCloseOperation (JFrame.EXIT_ON_CLOSE);
    }
    public void actionPerformed(ActionEvent ae)
    {
        card.next(c); // when a button is clicked show the next card.
    }
    public static void main(String args[])
    {
        Example3 ob = new Example3 ();
        ob.setTitle ("Card Layout...");
        ob.setSize (600,250);
        ob.setVisible (true);
    }
}
```

Output:



```
C:\WINDOWS\system32\cmd.exe - java Example3
D:\JQR>javac Example3.java
D:\JQR>java Example3
```



GridLayout: It is useful to divide the container into a two dimensional grid form that contains several rows and columns. The container is divided into equal sized rectangles and one component is placed in each rectangle. `GridLayout ()`, `GridLayout (int rows, int cols)`, `GridLayout (int rows, int cols, int hgap, int vgap)` are constructors.

Program 14: Write a program to create push buttons and arrange them using grid layout.

```
//Grid layout example
import javax.swing.*;
import java.awt.*;
class Example4 extends JFrame
{
    Container c;
    GridLayout grid;
    Example4 ()
    {
        c = getContentPane ();
        grid = new GridLayout (2,2,50,10);
        c.setLayout (grid);
        JButton b1 = new JButton ("First");
        JButton b2 = new JButton ("Second");
        JButton b3 = new JButton ("Third");
        c.add ("button1",b1);
        c.add ("button2",b2);
        c.add ("button3",b3);
        setDefaultCloseOperation (JFrame.EXIT_ON_CLOSE);
    }
    public static void main (String args[])
    {
        Example4 ob = new Example4 ();
        ob.setTitle ("Grid Layout...");
        ob.setSize (600,250);
        ob.setVisible (true);
    }
}
```


Output:

GridBagLayout: This layout is more flexible as compared to other layouts since in this layout the components can span more than one row or column and the size of the components can be adjusted to fit the display area.

- To create grid bag layout: `GridBagLayout obj = new GridBagLayout ();`
- To apply some constraints on the components, we should first create an object to `GridBagConstraints` class, as: `GridBagConstraints cons = new GridBagConstraints ();`
- `GridBagConstraints.gridx`, `GridBagConstraints.gridy` represents the row and column positions of the component at upper left corner.
- `GridBagConstraints.gridwidth`, `GridBagConstraints.gridheight` specifies number of columns and rows in the components display area.
- `GridBagConstraints.ipadx`, `GridBagConstraints.ipady` are useful to leave space horizontally and vertically with in the component.
- `GridBagConstraints.HORIZONTAL` makes the component wide enough to fill its display area horizontally, but do not change its height.
- `GridBagConstraints.VERTICAL` makes the component tall enough to fill its display area vertically, but do not change its width.
- `GridBagConstraints.Both` makes the component fill its display area entirely.
- `GridBagConstraints.insets` is useful to leave some space around the component at the four edges of component. This space is left around the component and the boundary of its display area. `insets` is the object of `Insets` class, so it is created as:

`Insets insets = new Insets (5, 10, 15, 20);`

Program 15: Write a program to create push buttons and arrange them using grid bag layout.

```
//GridBag layout example
import javax.swing.*;
```

```

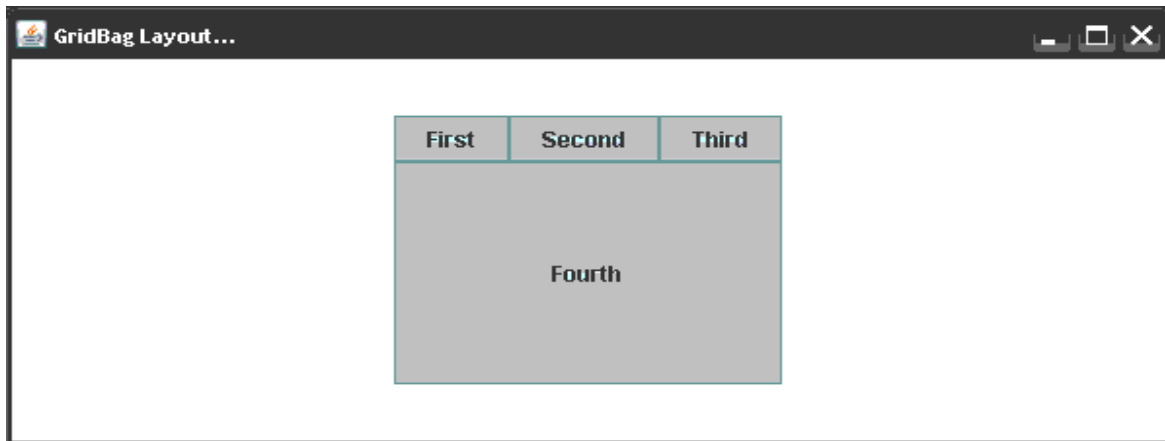
import java.awt.*;
class Example5 extends JFrame
{
    Container c;
    GridBagLayout gbag;
    GridBagConstraints cons;
    Example5 ()
    {
        c = getContentPane ();
        gbag = new GridBagLayout ();
        c.setLayout (gbag);
        cons = new GridBagConstraints();
        JButton b1 = new JButton ("First");
        JButton b2 = new JButton ("Second");
        JButton b3 = new JButton ("Third");
        JButton b4 = new JButton ("Fourth");
        //for all buttons use horizontal filling
        cons.fill = GridBagConstraints.HORIZONTAL;
        cons.gridx=0;
        cons.gridy = 0;
        gbag.setConstraints (b1, cons);
        c.add (b1);
        cons.gridx=1;
        cons.gridy =0;
        gbag.setConstraints (b2, cons);
        c.add (b2);

        cons.gridx=2;
        cons.gridy = 0;
        gbag.setConstraints (b3, cons);
        c.add (b3);
        cons.gridx=0;
        cons.gridy = 1;
        //add 100px height wise
        cons.ipady = 100;
        //let button3 occupy 3 columns width-wise
        cons.gridwidth = 3;
        gbag.setConstraints (b4, cons);
        c.add (b4);
        setDefaultCloseOperation (JFrame.EXIT_ON_CLOSE);
    }
    public static void main(String args[])
    {
        Example5 ob = new Example5 ();
        ob.setTitle ("GridBag Layout...");
        ob.setSize (600, 250);
        ob.setVisible (true);
    }
}

```

Output:

```
C:\WINDOWS\system32\cmd.exe - java Example5
D:\JQR>javac Example5.java
D:\JQR>java Example5
```



23. Applets

An applet is a program that comes from server into a client and gets executed at client side and displays the result. An applet represents byte code embedded in a html page. (applet = bytecode + html) and run with the help of Java enabled browsers such as Internet Explorer. An applet is a Java program that runs in a browser. Unlike Java applications applets do not have a main () method. To create applet we can use java.applet.Applet or javax.swing.JApplet class. All applets inherit the super class 'Applet'. An Applet class contains several methods that helps to control the execution of an applet.

Advantages:

- Applets provide dynamic nature for a webpage.
- Applets are used in developing games and animations.

Creating an applet:

- Let the Applet class extends Applet or JApplet class.
- Override the following methods of Applet class.
 - **public void init ()**: This method is used for initializing variables, parameters to create components. This method is executed only once at the time of applet loaded into memory.
 - **public void start ()**: After init() method is executed, the start method is executed automatically. Start method is executed as long as applet gains focus. In this method code related to opening files and connecting to database and retrieving the data and processing the data is written.
 - **public void stop ()**: This method is executed when the applet loses focus. Code related to closing the files and database, stopping threads and performing clean up operations are written in this stop method.
 - **public void destroy ()**: This method is executed only once when the applet is terminated from the memory.

Executing above methods in that sequence is called applet life cycle. We can also use public void paint (Graphics g) in applets.

After writing an applet, an applet is compiled in the same way as Java application but running of an applet is different. There are two ways to run an applet.

- Executing an applet within a Java compatible web browser.
- Executing an applet using 'appletviewer'. This executes the applet in a window.

To execute an applet using web browser, we must write a small HTML file which contains the appropriate 'APPLET' tag. <APPLET> tag is useful to embed an applet into an HTML page. It has the following form:

```
<APPLET CODE="name of the applet class file" CODEBASE="path of the applet class file"
        HEIGHT = maximum height of applet in pixels WIDTH = maximum width of applet
        in pixels ALIGN = alignment (LEFT, RIGHT, MIDDLE, TOP, BOTTOM)
        ALT = alternate text to be displayed>
  <PARAM NAME = parameter name VALUE = its value>
</APPLET>
```

The <PARAM> tag useful to define a variable (parameter) and its value inside the HTML page which can be passed to the applet. The applet can access the parameter value using `getParameter()` method, as:

```
String value = getParameter ("pname");
```

Where pname is the parameter name and its value is retrieved.

The HTML file must be saved with .html extension. After creating this file, open the Java compatible browser (Internet Explorer) and then load this file by specifying the complete path, then Applet program will get executed.

In order to execute applet program with an applet viewer, simply include a comment at the head of Java Source code file that contains the 'APPLET' tag. Thus, our code is documented with a prototype of the necessary HTML statements and we can test out compiled applet by starting the appletviewer with the Java file as: *appletviewer programname.java*

Program 1: Write an applet program with a message and display the message in `paint()` method.

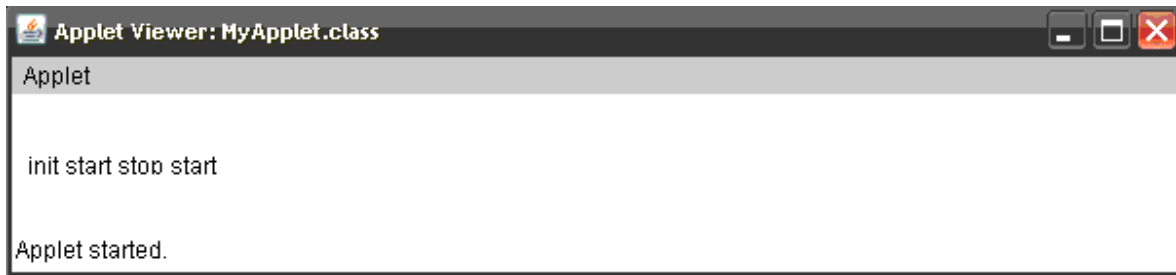
```
/*      <applet code="MyApplet.class" width = 600 height= 450>
      </applet>      */
```

```
import java.applet.Applet;
import java.awt.*;
public class MyApplet extends Applet
{
    String msg="";
    public void init()
    {
        msg += "init";
    }
    public void start()
    {
        msg += " start";
    }
    public void paint(Graphics g)
    {
        g.drawString(msg,10,100);
    }
    public void stop()
    {
        msg += " stop";
    }
    public void destroy()
    {
        msg+= " destroy";
    }
}
```

Output:



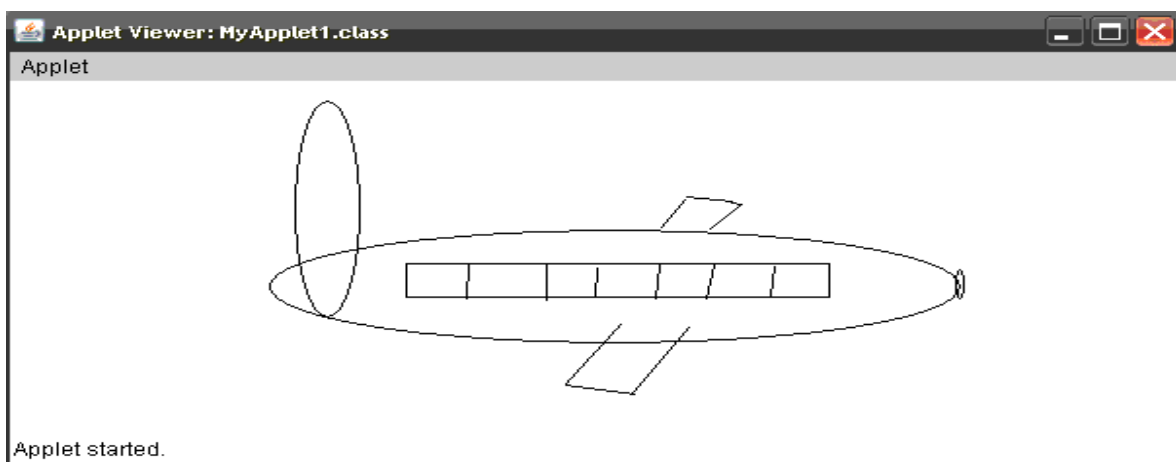
```
C:\WINDOWS\system32\cmd.exe - appletviewer MyApplet.java
D:\JQR>javac MyApplet.java
D:\JQR>appletviewer MyApplet.java
```



Program 2: Write a program to move an image from left to right in an Applet. To load an image use Image class of java.awt.

```
/*      <applet code="MyApplet1.class" width = 600 height= 450>
      </applet>      */
import java.applet.*;
import java.awt.*;
public class MyApplet1 extends Applet
{
    public void paint (Graphics g)
    {
        Image i = getImage (getDocumentBase (), "plane.gif");
        for (int x= 0 ; x<=800 ; x++)
        {
            g.drawImage (i, x, 0, null);
            try
            {
                Thread.sleep (20);
            }
            catch (InterruptedException ie) {
            }
        }
    }
}
```

Output:



Program 3: Write a program to pass employ name and id number to an applet.

```
/*    <applet code="MyApplet2.class" width = 600 height= 450>
      <param name = "t1" value="Hari Prasad">
      <param name = "t2" value ="101">
    </applet>    */
import java.applet.*;
import java.awt.*;
public class MyApplet2 extends Applet
{
    String n;
    String id;
    public void init()
    {
        n = getParameter("t1");
        id = getParameter("t2");
    }
    public void paint(Graphics g)
    {
        g.drawString("Name is : " + n, 100,100);
        g.drawString("Id is : "+ id, 100,150);
    }
}
```

Output:



```
C:\WINDOWS\system32\cmd.exe - appletviewer MyApplet2.java
D:\JQR>javac MyApplet2.java
D:\JQR>appletviewer MyApplet2.java
```

