LAB MANUAL OF

JAVA PROGRAMMING

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

ETCS-357

Maharaja Agrasen Institute of Technology, PSP area, Sector – 22, Rohini, New Delhi – 110085
(Affiliated to Guru Gobind Singh Indraprastha University, New Delhi)

INDEX OF THE CONTENTS

1. INTRODUCTION TO THE LAB

In java programming section, the applications of Java are taken into account. Applications of Java which are taken into details according to the syllabus prescribed by G.G.S.I.P.U for this lab are:

1. Console Based Programming
2. Applets
3. HTML
4. JDBC

2. LAB REQUIREMENTS

For Java Programming

J2SDK 1.7
Java Compatible Web Browser

This Compiler has no special hardware requirements as such. Any System with a minimum 256 MB RAM and any normal processor can use for this lab.

JAVA PROGRAMMING LAB

Paper Code: ETCS-357
Paper: Java Programming Lab

List of Experiments:
(As prescribed by G.G.S.I.P.U)

1. Create a java program to implement stack and queue concept.
2. Write a java package to show dynamic polymorphism and interfaces.
3. Write a java program to show multithreaded producer and consumer application.
4. Create a customized exception and also make use of all the 5 exception keywords.
5. Convert the content of a given file into the uppercase content of the same file.
6. Develop an analog clock using applet.
7. Develop a scientific calculator using swings.
8. Create an editor like MS-word using swings.
9. Create a servlet that uses Cookies to store the number of times a user has visited your servlet.
10. Create a simple java bean having bound and constrained properties.

NOTE:- At least 8 Experiments out of the list must be done in the semester.

3. LIST OF EXPERIMENTS (As prescribed by G.G.S.I.P.U)

Paper Code: ETCS-357	**P**	**C**
Paper: Java Programming Lab	**2**	**1**

Java Programming (List of Experiments)

Week 1

1. Write a program to print "Hello World" on the screen.
2. Write a program that calculates how long it takes to drive from New York to Los Angeles at 75 mile
   per hour (Use 3000 miles as the approximate distance between two cities).

3. Write a program that creates and initializes a four-element int array. Calculate and display the average
   of its values.

4. Write a program that creates a 2-d array with int values the first element should be an array containing 32. The second array should be an array containing 500 and 300 .The third element should be an array containing 39.45 and 600.Declare, allocate and initialize the array display its length and elements.

5 Write a program to swap two values using object reference. Your program should have a swap function

Week 2

6. Write an application that accepts two doubles as its command line arguments, multiple these together and display the product.
7. Write an application that accepts one command line argument; display the line of reporting if number is even or odd.
8. Write an application that accepts radius of a circle as its command line argument display the area.
9. WAP that describes a class person. It should have instance variables to record name, age and salary.
   Create a person object. Set and display its instance variables.
10. Write a program that uses length property for displaying any number of command line arguments.

Week 3

11. WAP that creates a class circle with instance variables for the centre and the radius. Initialize and display its variables.

12. Modify experiment 1 to have a constructor in class circle to initialize its variables.
13. Modify experiment 2 to show constructor overloading.
14. WAP to display the use of this keyword.
15. Write a program that can count the number of instances created for the class.

Week 4

16. WAP that implements method overloading.
17. WAP that shows passing object as parameter.
18. WAP that illustrates method overriding
19. Write a program to show that the value of non static variable is not visible to all the instances, and therefore cannot be used to count the number of instances.
20. WAP to illustrate simple inheritance

Week 5

21. WAP illustrating a super class variable a referencing as sub class object.
22. WAP illustrating all uses of super keywords.
23. Create an abstract class shape. Let rectangle and triangle inherit this shape class. Add necessary functions.
24. Write an application that creates a package p1. Add some classes in it.
25. Write an application that uses the package p1 created in the program 21.

Week 6

26. Write an application that creates an 'interface' and implements it.
27. Write an application to illustrate Interface Inheritance.
28. Write an application that shows the usage of try, catch, throws and finally.
29. Write an application that shows how to create a user-defined exception.

Week 7

30. Write an application that executes two threads. One thread displays "An" every 1000 milliseconds and other displays "B" every 3000 milliseconds. Create the threads by extending the Thread class.
31. Write an application that shows thread synchronization.
32. Write an application that displays deadlock between threads.
33. Write an application that shows thread priorities.

Week 8

34. Write an Applet that displays "Hello World" (Background color-black, text color-blue and your name in the status window.)
35. Write a program that displays the life cycle of an Applet.
36. Write an Applet displaying line, rectangle, rounded rectangle, filled rectangle, filled rounded rectangle, circle, ellipse, arc, filled arc and polygon, all in different colors
37. Write an Applet that displays a counter in the middle of applet.

Week 9

38. Write an Applet that displays a counter in the middle of applet. The counter starts from zero and keeps on incrementing after every second.
39. Write an Applet that draws a dot at a random location in its display area every 200ms. Any existing dots are not erased. Therefore dots accumulate as the applet executes.
40. Write an Applet that illustrates how to process mouse click, enter, exit, press and release events. The background color changes when the mouse is entered, clicked, pressed, released or exited.

41. Write an Applet that displays your name whenever the mouse is clicked.
Week 10
42. Use adapter classes to write an Applet those changes to cyan while the mouse is being dragged. At all other times the applet should be white.
43. Use inner classes to write an Applet those changes to cyan while the mouse is being dragged. At all other times the applet should be white.
44. Use anonymous classes to write an Applet those changes to cyan while the mouse is being dragged. At all other times the applet should be white.
Week 11
Q45 Basic File handling program in java with reader/writer .
Q46. Write a program that read from a file and write to file.
Week 12
Q47. Write RMI based client-server programs.
Week 13
Q48. Write programs of database connectivity using JDBC-ODBC drivers.

4. LIST OF ADVANCE EXPERIMENTS (Beyond the syllabus prescribed by G.G.S.I.P.U)

List of Advance experiments given to the students is summarized as below:

- Calendar
- Search & Replace in a file
- Scientific Calculator
- Address Book
- Paint Brush
- Note Pad

5. PROJECTS TO BE ALLOTED
(Beyond the syllabus prescribed by G.G.S.I.P.U)

Students will be divided into a group of four/five and projects are allotted to those groups. This project is to be submitted at the end of the semester along with a project report by the individual student.

List of projects given to the students is summarized as below:

- Dx Ball Game
- Moving ball with Java Script
- Checker Board game with Java Script
- Digital Image Processing
- Library Management System
- Global Defender Game
- Brick Game
- Rapid Roll Game
- Tic Tack Toe

Students can select project work of their own choice subject to the permission of concern faculty.

NOTE: The project is to be made in Java Language preferably.

6. FORMAT OF THE LAB RECORD TO BE PREPARED BY THE STUDENTS

1. The front page of the lab record prepared by the students should have a cover page as displayed below.

NAME OF THE LAB

Font should be(Size 20", italics bold, Times New Roman)

Faculty name

Font should be (12", Times Roman)

Student name

Roll No.:

Semester:

Group:

Font should be (12", Times Roman)

Maharaja Agrasen Institute of Technology, PSP Area,

Sector – 22, Rohini, New Delhi – 110085

Font should be (18", Times Roman)

2. The second page in the record should be the index as displayed below.

Java Programming

PRACTICAL RECORD

PAPER CODE : ETCS-357

Name of the student :

University Roll No. :

Branch :

Section/ Group :

PRACTICAL DETAILS

Experiments according to ITC lab syllabus prescribed by GGSIPU

Exp. no	Experiment Name	Date of performance	Date of checking	Remarks

PROJECT DETAILS

1. TITLE :
2. MEMBERS IN THE PROJECT GROUP :
3. PROJECT REPORT ATTACHED :
   a) YES b) NO
4. SOFT COPY SUBMITTED :
   a) YES b) NO

Signature of the lecturer Signature of the student

( ) ( )

3. Each practical which student is performing in the lab should have the following details :
   a) Topic Detail
   b) AIM
   c) Algorithm
   d) Source Code
   e) Output
   f) Viva questions

4. Project report should be added at last page.

7. MARKING SCHEME FOR THE PRACTICAL EXAMS

There will be two practical exams in each semester.

- Internal Practical Exam
- External Practical Exam

INTERNAL PRACTICAL EXAM

It is taken by the concerned lecturer of the batch.

MARKING SCHEME FOR THIS EXAM IS:

Total Marks: 40

Division of 40 marks is as follows

1. Regularity: 25
   - Performing program in each turn of the lab
   - Attendance of the lab
   - File
2. Viva Voice: 10
3. Project: 5

NOTE: For the regularity, marks are awarded to the student out of 10 for each experiment performed in the lab and at the end the average marks are giving out of 25.

<u>EXTERNAL PRACTICAL EXAM</u>

It is taken by the concerned lecturer of the batch and by an external examiner. In this exam student needs to perform the experiment allotted at the time of the examination, a sheet will be given to the student in which some details asked by the examiner needs to be written and at the last viva will be taken by the external examiner.

MARKING SCHEME FOR THIS EXAM IS:

Total Marks: 60

Division of 60 marks is as follows

1. Sheet filled by the student: 20
2. Viva Voice: 15
3. Experiment performance: 15
4. File submitted: 10

NOTE:

- Internal marks + External marks = Total marks given to the students
(40 marks) (60 marks) (100 marks)
- Experiments given to perform can be from any section of the lab.

8. DETAILS OF THE EACH SECTION ALONG WITH EXAMPLES, EXERCISES & EXPECTED VIVA QUESTIONS

Java Programming

THIS SECTION COVERS:

1. **Console Based Programming**
2. **OOP's Based Programming**
3. **AWT and Event Handling**
4. **Java DataBase Connectivity (JDBC)**
5. **Socket Programming and RMI**
6. **Java Native Interface (JNI)**
7. **Collection Interface**

JAVA PROGRAMMING ENVIRONMENT (AN INTRODUCTION)

Anyone who is learning to program has to choose a programming environment that makes it possible to create and to run programs. Programming environments can be divided into two very different types: integrated development environments and command-line environments. All programming environments for Java require some text editing capability, a Java compiler, and a way to run applets and stand-alone applications. An integrated development environment, or IDE, is a graphical user interface program that integrates all these aspects of programming and probably others (such as a debugger, a visual interface builder, and project management). A command-line environment is just a collection of commands that can be typed in to edit files, compile source code, and run programs.

Command line environment is preferable for beginning programmers. IDEs can simplify the management of large numbers of files in a complex project, but they are themselves complex programs that add another level of complications to the already difficult task of learning the fundamentals of programming.

Java was developed at Sun Microsystems, Inc., and the primary source for information about Java is Sun's Java Web site, http://java.sun.com/. At this site, one can read documentation on-line and you can download documentation and software. The documentation includes the Java API reference and the Java tutorial.

The current version of Java on the Sun site is version 1.4. It is available for the Windows, Linux, and Solaris operating systems. One can download the "J2SE 1.4 SDK." This is the "Java 2 Platform Standard Edition Version 1.4 Software Development Kit." This package includes a Java compiler, a Java virtual machine that can be used to run Java programs, and all the standard Java packages. The JRE is the "Java Runtime Environment." It only includes the parts of the system that are needed to run Java programs. It does not have a compiler.

Integrated Development Environments

There are sophisticated IDEs for Java programming that are available.

- Eclipse IDE -- An increasingly popular professional development environment that supports Java development, among other things. Eclipse is itself written in Java. It is available from http://www.eclipse.org/.
- NetBeans IDE -- A pure Java IDE that should run on any system with Java 1.7 or later. NetBeans is a free, "open source" program. It is essentially the open source version of the next IDE. It can be downloaded from www.netbeans.org.
- Sun ONE Studio 4 for Java, Community Edition, for Linux, Solaris, Windows 2000 to all versions till 2010, Windows NT, and Windows 98SE. This was formerly known as "Forte for Java", and it might be referred under that name. Again, it requires a lot of resources, with a 256 MB memory recommendation. Main site currently at http://www.sun.com/software/sundev/jde/index.html. It is available from there and on the J2SE download page, http://java.sun.com/j2se/1.7/download.html. The Community Edition is the free version.
- Borland JBuilder Personal Edition, for Linux, Solaris, MacOS X, Windows 2000, Windows XP, and Windows NT. Requires a lot of disk space & memory (256 MB memory recommended). Company Web page at http://www.borland.com. Jbuilder site at http://www.borland.com/jbuilder/index.html. The Personal Edition, which is free, has more than enough features for most programmers.
- BlueJ is a Java IDE written in Java that is meant particularly for educational use. It is available from http://www.bluej.org/.
- JCreator, for Windows. It looks like a nice lighter-weight IDE that works on top of Sun's SDK. There is a free version, as well as a shareware version. It is available at http://www.jcreator.com.

There are other products similar to JCreator, for Windows and for other operating systems.

Text Editors

To use a command-line environment for programming good text editor is needed. A programmer's text editor is a very different thing from a word processor. Most important, it saves work in plain text files and it doesn't insert extra carriage returns beyond the ones you actually type. A good programmer's text editor will do a lot more than this. Here are some features to look for:

- Syntax coloring. Shows comments, strings, keywords, etc., in different colors to make the program easier to read and to help you find certain kinds of errors.
- Function menu. A pop-up menu that lists the functions in your source code. Selecting a function from this will take you directly to that function in the code.
- Auto-indentation. When you indent one line, the editor will indent following lines to match, since that's what you want more often than not when you are typing a program.
- Parenthesis matching. After typing a closing parenthesis the cursor jumps back to the matching parenthesis momentarily so one can see where it is. Alternatively, there might be a command that will highlight all the text between matching parentheses. The same thing works for brackets and braces.
- Indent Block and Unindent Block commands. These commands apply to a highlighted block of text. They will insert or remove spaces at the beginning of each line to increase or decrease the indentation level of that block of text. When you make changes in your program, these commands can help you keep the indentation in line with the structure of the program.
- Control of tabs. Don’t use tab characters for indentation. A good editor can be configured to insert multiple space characters when tab key is pressed.

There are many free text editors that have some or all of these features. **Jedit**,a programmer's text editor written entirely in Java. It requires Java 1.3 or better. It has many features listed above, and there are plug-ins available to add additional features. Since it is written in pure Java, it can be used on any operating system that supports Java 1.3. In addition to being a nice text editor, it shows what can be done with the Swing GUI. Jedit is free and can be downloaded from http://www.jedit.org.

On Linux, use **nedit**. It has all the above features, except a function menu. Under Linux, it is likely that *nedit* is included in distribution, although it may not have been installed by default. It can be downloaded from http://www.nedit.org/ and is available for many UNIX platforms in addition to Linux. Features such as syntax coloring and auto-indentation are not turned on by default. One can configure them in the Options menu. Use the "Save Options" command to make the configuration permanent. Of course, as alternatives to nedit, the Gnome and KDE desktops for Linux have their own text editors.

Using the Java SDK

After installing Sun's Software Development Kit for Java, one can use the commands "javac", "java", and "appletviewer" for compiling and running Java programs and applets. These commands must be on the "path" where the operating system searches for commands.

Make a directory to hold Java programs. Create program with a text editor, or copy the program to be compiled into program directory

If program contains more than a few errors, most of them will scroll out of the window. In Linux and UNIX, a command window usually has a scroll bar that can be used to review the errors. In Windows 2000 to 2010/NT/XP (but **not** Windows 95/98), one can save the errors in a file which can be viewed later in a text editor.

The command in Windows is

javac SourceFile.java >& errors.txt

The ">& errors.txt" redirects the output from the compiler to the file, instead of to the DOS window. It is possible to compile all the Java files in a directory at one time. Use the command "javac *.java".

After compiled class files are made, run application or applet. For running a stand-alone application -- one that has a main () routine -- use the "java" command from the SDK to run the application. If the class file that contains the main () routine is named Main. class, then run the program with the command:

java Main

SAMPLE CONSOLE BASED PROGRAM

The following program, For Demo, uses the general form of the for statement to print the numbers 1 through 10 to standard output:

Steps to write JAVA Program

1. Create JAVA file by text editor (eg vi editor).

2. Write the program as per JAVA syntax.

3. Save the file with .java extension.

4. Compile the file with JAVA compiler (javac filename.java) and create class file.

5. Run the class file with JAVA interpreter (java classname.class) and check the output.

PRACTICAL – 1

Aim 1(a). Factorial of a number
(b). Determine If Year Is Leap Year
(c). Fibonacci Series
(d). Palindrome Number

Software Required: JDK 1.7
Theory:
Selection Statements

<u>The If Statement</u>

The if statement executes a block of code only if the specified expression is true. If the value is false, then the if block is skipped and execution continues with the rest of the program. You can either have a single statement or a block of code within an if statement. Note that the conditional expression must be a Boolean expression.

The simple if statement has the following syntax:

<u>if (<conditional expression>)</u>
<u>< statement action></u>

Below is an example that demonstrates conditional execution based on if statement condition.

```
public class IfStatementDemo {

	public static void main(String[] args) {
		int a = 10, b = 20;
		if (a > b)
			System.out.println("a > b");
		if (a < b)
			System.out.println("b > a");
	}
}
```

Output

b > a

<u>The If-else Statement</u>

The if/else statement is an extension of the if statement. If the statements in the if statement fails, the statements in the else block are executed. You can either have a single statement or a block of code within if-else blocks. Note that the conditional expression must be a Boolean expression.

The if-else statement has the following syntax:

if (<conditional expression>)
< statement action>
else
< statement action>

Below is an example that demonstrates conditional execution based on if else statement condition.

```
public class IfElseStatementDemo {

	public static void main(String[] args) {
		int a = 10, b = 20;
		if (a > b) {
			System.out.println("a > b");
		} else {
			System.out.println("b > a");
		}
	}
}
```

Output

b > a

Iteration Statements

While Statement

The while statement is a looping construct control statement that executes a block of code while a condition is true. You can either have a single statement or a block of code within the while loop. The loop will never be executed if the testing expression evaluates to false. The loop condition must be a boolean expression.

The syntax of the while loop is

while (<loop condition>)
< statements>

Below is an example that demonstrates the looping construct namely while loop used to print numbers from 1 to 10.

```
public class WhileLoopDemo {

	public static void main(String[] args) {
		int count = 1;
		System.out.println("Printing Numbers from 1 to 10");
```

```
		while (count <= 10) {
			System.out.println(count++);
		}
	}
}
```

Output

Printing Numbers from 1 to 10
1
2
3
4
5
6
7
8
9
10

Do-while Loop Statement

The do-while loop is similar to the while loop, except that the test is performed at the end of the loop instead of at the beginning. This ensures that the loop will be executed at least once. A do-while loop begins with the keyword do, followed by the statements that make up the body of the loop. Finally, the keyword while and the test expression completes the do-while loop. When the loop condition becomes false, the loop is terminated and execution continues with the statement immediately following the loop. You can either have a single statement or a block of code within the do-while loop.

The syntax of the do-while loop is

do
< loop body>
while (<loop condition>);

Below is an example that demonstrates the looping construct namely do-while loop used to print numbers from 1 to 10.

```
public class DoWhileLoopDemo {

	public static void main(String[] args) {
		int count = 1;
		System.out.println("Printing Numbers from 1 to 10");
		do {
```

```
                        System.out.println(count++);
                } while (count <= 10);
        }
}
```

Output

Printing Numbers from 1 to 10
1
2
3
4
5
6
7
8
9
10

Below is an example that creates A Fibonacci sequence controlled by a do-while loop

```
public class Fibonacci {

        public static void main(String args[]) {
                System.out.println("Printing Limited set of Fibonacci Sequence");
                double fib1 = 0;
                double fib2 = 1;
                double temp = 0;
                System.out.println(fib1);
                System.out.println(fib2);
                do {
                        temp = fib1 + fib2;
                        System.out.println(temp);
                        fib1 = fib2; //Replace 2nd with first number
                        fib2 = temp; //Replace temp number with 2nd number
                } while (fib2 < 5000);
        }
}
```

Output

Printing Limited set of Fibonacci Sequence

0.0
1.0

1.0
2.0
3.0
5.0
8.0
13.0
21.0
34.0
55.0
89.0
144.0
233.0
377.0
610.0
987.0
1597.0
2584.0
4181.0
6765.0

<u>For Loops</u>

The for loop is a looping construct which can execute a set of instructions a specified number of times. It’s a counter controlled loop.

The syntax of the loop is as follows:

for (<initialization>; <loop condition>; <increment expression>)
< loop body>

The first part of a for statement is a starting initialization, which executes once before the loop begins. The <initialization> section can also be a comma-separated list of expression statements. The second part of a for statement is a test expression. As long as the expression is true, the loop will continue. If this expression is evaluated as false the first time, the loop will never be executed. The third part of the for statement is the body of the loop. These are the instructions that are repeated each time the program executes the loop. The final part of the for statement is an increment expression that automatically executes after each repetition of the loop body. Typically, this statement changes the value of the counter, which is then tested to see if the loop should continue.
All the sections in the for-header are optional. Any one of them can be left empty, but the two semicolons are mandatory. In particular, leaving out the <loop condition> signifies that the loop condition is true. The (;;) form of for loop is commonly used to construct an infinite loop.

Below is an example that demonstrates the looping construct namely for loop used to print numbers from 1 to 10.

```
public class ForLoopDemo {

	public static void main(String[] args) {
		System.out.println("Printing Numbers from 1 to 10");
		for (int count = 1; count <= 10; count++) {
			System.out.println(count);
		}
	}
}
```

Output

Printing Numbers from 1 to 10
1
2
3
4
5
6
7
8
9
10

Continue Statement

A continue statement stops the iteration of a loop (while, do or for) and causes execution to resume at the top of the nearest enclosing loop. You use a continue statement when you do not want to execute the remaining statements in the loop, but you do not want to exit the loop itself.

The syntax of the continue statement is

continue; // the unlabeled form
continue <label>; // the labeled form

You can also provide a loop with a label and then use the label in your continue statement. The label name is optional, and is usually only used when you wish to return to the outermost loop in a series of nested loops.

Below is a program to demonstrate the use of continue statement to print Odd Numbers between 1 to 10.

```
public class ContinueExample {

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

```
                System.out.println("Odd Numbers");
                for (int i = 1; i <= 10; ++i) {
                        if (i % 2 == 0)
                                continue;
                        // Rest of loop body skipped when i is even
                        System.out.println(i + "\t");
                }
        }
}
```

Output

Odd Numbers
1
3
5
7
9

Break Statement

The break statement transfers control out of the enclosing loop (for, while, do or switch statement). You use a break statement when you want to jump immediately to the statement following the enclosing control structure. You can also provide a loop with a label, and then use the label in your break statement. The label name is optional, and is usually only used when you wish to terminate the outermost loop in a series of nested loops.

The Syntax for break statement is as shown below;

break; // the unlabeled form
break <label>; // the labeled form

Below is a program to demonstrate the use of break statement to print numbers Numbers 1 to 10.

```
public class BreakExample {

        public static void main(String[] args) {
                System.out.println("Numbers 1 - 10");
                for (int i = 1;; ++i) {
                        if (i == 11)
                                break;
                        // Rest of loop body skipped when i is even
                        System.out.println(i + "\t");
                }
        }
```

```
}
```

Output

Numbers 1 – 10
1
2
3
4
5
6
7
8
9
10

```
/*
This program shows how to calculate
Factorial of a number.
*/
public class NumberFactorial {
public static void main(String[] args) {
int number = 5;
int factorial = number;
for(int i =(number - 1); i > 1; i--)

{
factorial = factorial * i;
}
System.out.println("Factorial of a number is " + factorial);
}
}
/*
Output of the Factorial program would be
Factorial of a number is 120
*/
```

```
*
This program shows how to check for in the given list of numbers
whether each number is palindrome or not
*/
```

```
public class JavaPalindromeNumberExample {
public static void main(String[] args) {
//array of numbers to be checked
int numbers[] = new int[]{121,13,34,11,22,54};
//iterate through the numbers
for(int i=0; i < numbers.length; i++){
int number = numbers[i];
int reversedNumber = 0;
int temp=0;
/*
* If the number is equal to it's reversed number, then
* the given number is a palindrome number.
*
* For ex,121 is a palindrome number while 12 is not.
*/
//reverse the number
while(number > 0){
temp = number % 10;
number = number / 10;
reversedNumber = reversedNumber * 10 + temp;
8
}
if(numbers[i] == reversedNumber)
System.out.println(numbers[i] + " is a palindrome");
else
System.out.println(numbers[i] + " not a palindrome ");
}
}
}


/*
Output of Java Palindrome Number Example would be
121 is a palindrome number
13 is not a palindrome number
34 is not a palindrome number
11 is a palindrome number
22 is a palindrome number
54 is not a palindrome number
```

PRACTICAL – 2

Aim:(a). Write a program to display a greet message according to marks obtained by student.
(b). Calculate Circle Area using radius
(c). Take personal data from user
(d). Sum and average of n numbers from user

Software Required: JDK1.7

Theory:

Switch Case Statement The switch case statement, also called a case statement is a multi-way branch with several choices. A switch is easier to implement than a series of if/else statements. The switch statement begins with a keyword, followed by an expression that equates to a no long integral value. Following the controlling expression is a code block that contains zero or more labeled cases. Each label must equate to an integer constant and each must be unique. When the switch statement executes, it compares the value of the controlling expression to the values of each case label. The program will select the value of the case label that equals the value of the controlling expression and branch down that path to the end of the code block. If none of the case label values match, then none of the codes within the switch statement code block will be executed. Java includes a default label to use in cases where there are no matches. We can have a nested switch within a case block of an outer switch.

Its general form is as follows:

switch (<non-long integral expression>) {
case label$_1$: <statement$_1$>
case label$_2$: <statement$_2$>
…
case label$_n$: <statement$_n$>
default: <statement>
} // end switch

When executing a switch statement, the program falls through to the next case. Therefore, if you want to exit in the middle of the switch statement code block, you must insert a break statement, which causes the program to continue executing after the current code block.

Below is a java example that demonstrates conditional execution based on nested if else statement condition to find the greatest of 3 numbers.

```
public class SwitchCaseStatementDemo {

	public static void main(String[] args) {
		int a = 10, b = 20, c = 30;
		int status = -1;
		if (a > b && a > c) {
			status = 1;
		} else if (b > c) {
```

```
                    status = 2;
            } else {
                    status = 3;
            }
            switch (status) {
            case 1:
                    System.out.println("a is the greatest");
                    break;
            case 2:
                    System.out.println("b is the greatest");
                    break;
            case 3:
                    System.out.println("c is the greatest");
                    break;
            default:
                    System.out.println("Cannot be determined");
            }
    }
}
```

Output

c is the greatestclass SwitchDemo{

(b). Calculate Circle Area using radius

```
class Sum_Product_ofDigit{
public static void main(String args[]){
int num = Integer.parseInt(args[0]);
//taking value as command line argument.
int temp = num,result=0;
//Logic for sum of digit
while(temp>0){
result = result + temp;
temp--;
}
System.out.println("Sum of Digit for "+num+" is : "+result);
//Logic for product of digit
temp = num;
result = 1;
while(temp > 0){
result = result * temp;
temp--;
}
System.out.println("Product of Digit for "+num+" is : "+result);
}
```

}

(b). Calculate Circle Area using radius

```
/*
This program shows how to calculate
area of circle using it's radius.
*/
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
public class CalculateCircleAreaExample {
public static void main(String[] args) {
int radius = 0;
System.out.println("Please enter radius of a circle");
try
{
//get the radius from console
BufferedReader br = new BufferedReader(new
InputStreamReader(System.in));
radius = Integer.parseInt(br.readLine());
}
//if invalid value was entered
catch(NumberFormatException ne)
{
System.out.println("Invalid radius value" + ne);
System.exit(0);
}
catch(IOException ioe)
{
System.out.println("IO Error :" + ioe);
System.exit(0);
}
/*
* Area of a circle is
* pi * r * r
* where r is a radius of a circle.
*/
//NOTE : use Math.PI constant to get value of pi
double area = Math.PI * radius * radius;
System.out.println("Area of a circle is " + area);
}
}
/*
Output of Calculate Circle Area using Java Example would be

Please enter radius of a circle
```

19
Area of a circle is 1134.1149479459152

(c) – Take personal data from user

```
/*
This program shows how take user input using scanner class
.
*/

* import the Scanner class */
import java.util.Scanner;

/** demonstrate how to read from System.in */
class ReadConsole {
public static void main(String[] args) {
Scanner scanner = new Scanner(System.in);

System.out.print("Enter your full name: ");
String name = scanner.nextLine();

System.out.print("Enter your Zodiac sign: ");
String zodiac = scanner.next();

System.out.print("Enter your weight (kg): ");
double weight = scanner.nextDouble();

System.out.print("Enter your lucky number: ");
int luckyNum = scanner.nextInt();

System.out.println("Hello, " + name + ".");
System.out.println("Your lucky number is " + luckyNum + ".");
System.out.println("You weigh " + weight + " kg.");
System.out.println("Your Zodiac sign is " + zodiac + ".");
}
}
```

Output:

```
Enter your full name: destin
Enter your Zodiac sign: Cancer
Enter your weight (kg): 70.45
Enter your lucky number: 7

Hello, destin.
Your lucky number is 7.
You weigh 70.45 kg.
Your Zodiac sign is Cancer.
```

PRACTICAL – 3

Aim(a). Write an application that creates stack class and extends the class to provide functionality.Use super and this keyword

(b). Write a program showing Multilevel inheritance

Software Required: JDK1.7

Theory:

Inheritance:

As the name suggests, inheritance means to take something that is already made. It is one of the most important feature of Object Oriented Programming. It is the concept that is used for reusability purpose. Inheritance is the mechanism through which we can derived classes from other classes. The derived class is called as child class or the subclass or we can say the extended class and the class from which we are deriving the subclass is called the base class or the parent class. To derive a class in java the keyword extends is used. To clearly understand the concept of inheritance you must go through the following example.

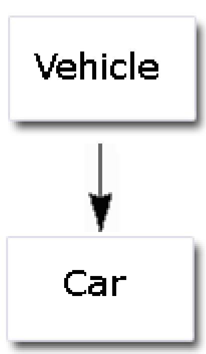
The concept of inheritance is used to make the things from general to more specific e.g. When we hear the word vehicle then we got an image in our mind that it moves from one place to another place it is used for traveling or carrying goods but the word vehicle does not specify whether it is two or three or four wheeler because it is a general word. But the word car makes a more specific image in mind than vehicle, that the car has four wheels . It concludes from the example that car is a specific word and vehicle is the general word. If we think technically to this example then vehicle is the super class (or base class or parent class) and car is the subclass or child class because every car has the features of it's parent (in this case vehicle) class.

The following kinds of inheritance are there in java.

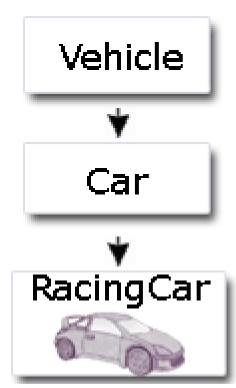
Simple Inheritance

Multilevel Inheritance

Pictorial Representation of Simple and Multilevel Inheritance



Simple Inheritance



Multilevel Inheritance

Simple Inheritance

When a subclass is derived simply from it's parent class then this mechanism is known as simple inheritance. In case of simple inheritance there is only a sub class and it's parent class. It is also called single inheritance or one level inheritance.

eg.

```
class A {
int x;
```

```
int y;
int get(int p, int q){
x=p; y=q; return(0);
}
void Show(){
System.out.println(x);
}
}

class B extends A{
public static void main(String args[]){
A a = new A();
a.get(5,6);
a.Show();
}
void display(){
System.out.println("B");
}
}
```

Multilevel Inheritance

It is the enhancement of the concept of inheritance. When a subclass is derived from a derived class then this mechanism is known as the multilevel inheritance. The derived class is called the subclass or child class for it's parent class and this parent class works as the child class for it's just above (parent) class. Multilevel inheritance can go up to any number of level.

e.g.

```
class A {
int x;
int y;
int get(int p, int q){
x=p; y=q; return(0);
}
void Show(){
System.out.println(x);
}
}
class B extends A{
void Showb(){
System.out.println("B");
}
}

class C extends B{
void display(){
System.out.println("C");
}
public static void main(String args[]){
```

```
A a = new A();
a.get(5,6);
a.Show();
}
}
```

Java does not support multiple Inheritance

Multiple Inheritance

The mechanism of inheriting the features of more than one base class into a single class is known as multiple inheritance. Java does not support multiple inheritance but the multiple inheritance can be achieved by using the interface.

In Java Multiple Inheritance can be achieved through use of Interfaces by implementing more than one interfaces in a class.

super keyword

The super is java keyword. As the name suggest super is used to access the members of the super class.It is used for two purposes in java.

The first use of keyword super is to access the hidden data variables of the super class hidden by the sub class.

e.g. Suppose class A is the super class that has two instance variables as int a and float b. class B is the subclass that also contains its own data members named a and b. then we can access the super class (class A) variables a and b inside the subclass class B just by calling the following command.

super.member;

Here member can either be an instance variable or a method. This form of super most useful to handle situations where the local members of a subclass hides the members of a super class having the same name. The following example clarify all the confusions.

```
class A{
int a;
float b;
void Show(){
System.out.println("b in super class: " + b);
}

}

class B extends A{
int a;
float b;
B( int p, float q){
a = p;
super.b = q;
}
void Show(){
super.Show();
System.out.println("b in super class: " + super.b);
System.out.println("a in sub class: " + a);
}
```

```
public static void main(String[] args){
B subobj = new B(1, 5);
subobj.Show();
}
}
```

Output:

```
C:\>java B
b in super class: 5.0
b in super class: 5.0
a in sub class: 1
```

Use of super to call super class constructor: The second use of the keyword super in java is to call super class constructor in the subclass. This functionality can be achieved just by using the following command.

super(param-list);

Here parameter list is the list of the parameter requires by the constructor in the super class. super must be the first statement executed inside a super class constructor. If we want to call the default constructor then we pass the empty parameter list. The following program illustrates the use of the super keyword to call a super class constructor.

```
class A{
int a;
int b;
int c;
A(int p, int q, int r){
a=p;
b=q;
c=r;
}
}

class B extends A{
int d;
B(int l, int m, int n, int o){
super(l,m,n);
d=o;
}
void Show(){
System.out.println("a = " + a);
System.out.println("b = " + b);
System.out.println("c = " + c);
System.out.println("d = " + d);
}

public static void main(String args[]){
B b = new B(4,3,8,7);
b.Show();
```

```
}
}
```

Output:

PRACTICAL – 4

Aim (a). Write a program using abstract methods and classes showing Runtime polymorphism.
(b). Write a program on Interfaces

Software Required: JDK1.7

Theory:

There are situations in which you will want to define a superclass that declares the structure of a given abstraction without providing a complete implementation of every method. That is, sometimes you will want to create a superclass that only defines a generalized form that will be shared by all of its subclasses, leaving it to each subclass to fill in the details. Such a class determines the nature of the methods that the subclasses must implement. One way this situation can occur is when a superclass is unable to create a meaningful implementation for a method. This is the case with the class Figure used in the preceding example. The definition of area( ) is simply a placeholder. It will not compute and display the area of any type of object.

As you will see as you create your own class libraries, it is not uncommon for a method to have no meaningful definition in the context of its superclass. You can handle this situation two ways. One way, as shown in the previous example, is to simply have it report a warning message. While this approach can be useful in certain situations—such as debugging—it is not usually appropriate. You may have methods which must be overridden by the subclass in order for the subclass to have any meaning. Consider the class Triangle. It has no meaning if area( ) is not defined. In this case, you want some way to ensure that a subclass does, indeed, override all necessary methods. Java's
solution to this problem is the abstract method.

You can require that certain methods be overridden by subclasses by specifying the abstract type modifier. These methods are sometimes referred to as subclasser responsibility because they have no implementation specified in the superclass. Thus, a subclass must override them—it cannot simply use the version defined in the superclass. To declare an abstract method, use this general form:

abstract type name(parameter-list);

As you can see, no method body is present. Any class that contains one or more abstract methods must also be declared abstract. To declare a class abstract, you simply use the abstract keyword in front of the class keyword at the beginning of the class declaration. There can be no objects of an abstract class. That is, an abstract class cannot be directly instantiated with the new operator. Such objects would be useless, because an abstract class is not fully defined. Also, you cannot declare abstract constructors, or abstract static methods. Any subclass of an abstract class must either implement all of the abstract methods in the superclass, or be itself declared abstract.

Here is a simple example of a class with an abstract method, followed by a class which implements that method:

```
// A Simple demonstration of abstract.
abstract class A {
abstract void callme();
// concrete methods are still allowed in abstract classes
void callmetoo() {
System.out.println("This is a concrete method.");
}
}
```

```
class B extends A {
void callme() {
System.out.println("B's implementation of callme.");
}
}
class AbstractDemo {
public static void main(String args[]) {
B b = new B();
b.callme();
b.callmetoo();
}
}
```

Notice that no objects of class A are declared in the program. As mentioned, it is not possible to instantiate an abstract class. One other point: class A implements a concrete method called callmetoo( ). This is perfectly acceptable. Abstract classes can include as much implementation as they see fit.

Although abstract classes cannot be used to instantiate objects, they can be used to create object references, because Java's approach to run-time polymorphism is implemented through the use of superclass references. Thus, it must be possible to create a reference to an abstract class so that it can be used to point to a subclass object. You will see this feature put to use in the next example.

Using an abstract class, you can improve the Figure class shown earlier. Since there is no meaningful concept of area for an undefined two-dimensional figure, the following version of the program declares area( ) as abstract inside Figure. This, of course, means that all classes derived from Figure must override area( ).

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

```
class Triangle extends Figure {
Triangle(double a, double b) {
super(a, b);
}
// override area for right triangle
double area() {
System.out.println("Inside Area for 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);
Figure figref; // this is OK, no object is created
figref = r;
System.out.println("Area is " + figref.area());
figref = t;
System.out.println("Area is " + figref.area());
}
}
```

As the comment inside main( ) indicates, it is no longer possible to declare objects of type Figure, since it is now abstract. And, all subclasses of Figure must override area( ). To prove this to yourself, try creating a subclass that does not override area( ). You will receive a compile-time error.

Although it is not possible to create an object of type Figure, you can create a reference variable of type Figure. The variable figref is declared as a reference to Figure, which means that it can be used to refer to an object of any class derived from Figure. As explained, it is through superclass reference variables that overridden methods are resolved at run time.

Interfaces:

b. What is interface in Java?

Interface in java is core part of Java programming language and one of the way to achieve abstraction in Java along with abstract class. Even though interface is fundamental object oriented concept ; Many Java programmers thinks Interface in Java as advanced concept and refrain using interface from early in programming career. At very basic level interface in java is a keyword but same time it is an object oriented term to define contracts and abstraction , This contract is followed by any implementation of Interface in Java. Since multiple inheritance is not allowed in Java, interfaceis only way to implement multiple inheritance at Type level. In this Java tutorial we will see What is an interface in Java, How to use interface in Java and where to use interface in Java and some important points related to Java interface. What is an interface in Java is also a common core Java question which people asked on various programming exams and interviews.

Key Points about Interface in Java

In last section we saw What is an interface in Java and learned that interface provides abstraction in Java and its only way to achieve multiple inheritance at type level in Java. In this section we will see some important properties of interface in Java.

1. Interface in java is declared using keyword interface and it represent a Type like any Class in Java. a reference variable of type interface can point to any implementation of that interface in Java. Its also a good Object oriented design principle to "program for interfaces than implementation" because when you use interface to declare reference variable, method return type or method argument you are flexible enough to accept any future implementation of that interface which could be much better and high performance alternative of current implementation. similarly calling any method on interface doesn't tie you with any particular implementation and you can leverage benefit of better or improved implementation over time. This maintenance aspect of interface is also sought in various software design interview questions in Java.

2) All variables declared inside interface is implicitly public final variable or constants. which brings a useful case of using Interface for declaring Constants. We have used both Class and interface for storing application wide constants and advantage of using Interface was that you can implement interface and can directly access constants without referring them with class name which was the case earlier when Class is used for storing Constants. Though after introduction of static imports in Java 5 this approach doesn't offer any benefit over Class approach.

3) All methods declared inside Java Interfaces are implicitly public and abstract, even if you don't use public or abstract keyword. you can not define any concrete method in interface. That's why interface is used to define contracts in terms of variables and methods and you can rely on its implementation for performing job.

4) In Java its legal for an interface to extend multiple interface. for example following code will run without any compilation error:

interface Session extends Serializable, Clonnable{ }

Here Session interface in Java is also a Serializable and Clonnable. This is not true for Class in Java and one Class can only extend at most another Class. In Java one Class can implement multiple interfaces. They are required to provide implementation of all methods declared inside interface or they can declare themselves as abstract class.

.

When to use interface in Java

Interface is best choice for Type declaration or defining contract between multiple parties. If multiple programmer are working in different module of project they still use each others API by defining interface and not waiting for actual implementation to be ready. This brings us lot of flexibility and speed in terms of coding and development. Use of Interface also ensures best practices like "programming for interfaces than implementation" and results in more flexible and maintainable code. Though interface in Java is not the only one who provides higher level abstraction, you can also use abstract class but choosing between Interface in Java and abstract class is a skill.

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

          shapeA circleshape=new circle();

             circleshape.Draw();
             circleshape.Draw();
    }
}

  interface shapeA
 {
    public   String baseclass="shape";
    public void Draw();
 }
  interface shapeB extends shapeA
 {
    public   String baseclass="shape2";
    public void Draw2();
 }
   class circle implements shapeB
 {
   public   String baseclass="shape3";
   public void Draw() {
       System.out.println("Drawing Circle here:"+baseclass);
   }
   @Override
   public void Draw2() {
        System.out.println("Drawing Circle here:"+baseclass);
   }
 }
```

The output is :
java code
Drawing Circle here:shape3
Drawing Circle here:shape3

PRACTICAL – 5

Aim . Exception handling
(a). Write an program using try, catch, throw and finally.
(b). Create your own exception class
Software Required: JDK1.7

Theory:
An exception is a problem that arises during the execution of a program. An exception can occur for many different reasons, including the following:

- A user has entered invalid data.
- A file that needs to be opened cannot be found.
- A network connection has been lost in the middle of communications, or the JVM has run out of memory.

Some of these exceptions are caused by user error, others by programmer error, and others by physical resources that have failed in some manner.

To understand how exception handling works in Java, you need to understand the three categories of exceptions:

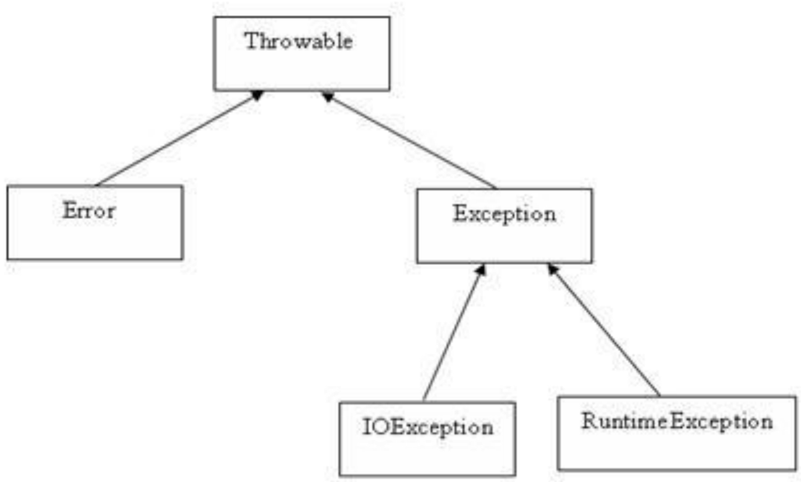
- **Checked exceptions:** A checked exception is an exception that is typically a user error or a problem that cannot be foreseen by the programmer. For example, if a file is to be opened, but the file cannot be found, an exception occurs. These exceptions cannot simply be ignored at the time of compilation.
- **Runtime exceptions:** A runtime exception is an exception that occurs that probably could have been avoided by the programmer. As opposed to checked exceptions, runtime exceptions are ignored at the time of compliation.
- **Errors:** These are not exceptions at all, but problems that arise beyond the control of the user or the programmer. Errors are typically ignored in your code because you can rarely do anything about an error. For example, if a stack overflow occurs, an error will arise. They are also ignored at the time of compilation.

Exception Hierarchy:

All exception classes are subtypes of the java.lang.Exception class. The exception class is a subclass of the Throwable class. Other than the exception class there is another subclass called Error which is derived from the Throwable class.

Errors are not normally trapped form the Java programs. These conditions normally happen in case of severe failures, which are not handled by the java programs. Errors are generated to indicate errors generated by the runtime environment. Example : JVM is out of Memory. Normally programs cannot recover from errors.

The Exception class has two main subclasses : IOException class and RuntimeException Class.



Here is a list of most common checked and unchecked Java's Built-in Exceptions.

Exceptions Methods:

Following is the list of important medthods available in the Throwable class.

SN	Methods with Description
1	**public String getMessage()** Returns a detailed message about the exception that has occurred. This message is initialized in the Throwable constructor.
2	**public Throwable getCause()** Returns the cause of the exception as represented by a Throwable object.
3	**public String toString()** Returns the name of the class concatenated with the result of getMessage()
4	**public void printStackTrace()** Prints the result of toString() along with the stack trace to System.err, the error output stream.

5	**public StackTraceElement [] getStackTrace()** Returns an array containing each element on the stack trace. The element at index 0 represents the top of the call stack, and the last element in the array represents the method at the bottom of the call stack.
6	**public Throwable fillInStackTrace()** Fills the stack trace of this Throwable object with the current stack trace, adding to any previous information in the stack trace.

Catching Exceptions:

A method catches an exception using a combination of the **try** and **catch** keywords. A try/catch block is placed around the code that might generate an exception. Code within a try/catch block is referred to as protected code, and the syntax for using try/catch looks like the following:

```
try
{
  //Protected code
}catch(ExceptionName e1)
{
  //Catch block
}
```

A catch statement involves declaring the type of exception you are trying to catch. If an exception occurs in protected code, the catch block (or blocks) that follows the try is checked. If the type of exception that occurred is listed in a catch block, the exception is passed to the catch block much as an argument is passed into a method parameter.

Example:

The following is an array is declared with 2 elements. Then the code tries to access the 3rd element of the array which throws an exception.

```
// File Name : ExcepTest.java
import java.io.*;
public class ExcepTest{

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

```
      try{
        int a[] = new int[2];
        System.out.println("Access element three :" + a[3]);
      }catch(ArrayIndexOutOfBoundsException e){
        System.out.println("Exception thrown  :" + e);
      }
      System.out.println("Out of the block");
   }
}
```

This would produce following result:

```
Exception thrown  :java.lang.ArrayIndexOutOfBoundsException: 3
Out of the block
```

Multiple catch Blocks:

A try block can be followed by multiple catch blocks. The syntax for multiple catch blocks looks like the following:

```
try
{
  //Protected code
}catch(ExceptionType1 e1)
{
  //Catch block
}catch(ExceptionType2 e2)
{
  //Catch block
}catch(ExceptionType3 e3)
{
  //Catch block
}
```

The previous statements demonstrate three catch blocks, but you can have any number of them after a single try. If an exception occurs in the protected code, the exception is thrown to the first catch block in the list. If the data type of the exception thrown matches ExceptionType1, it gets caught there. If not, the exception passes down to the second catch statement. This continues until the exception either is caught or falls through all catches, in which case the current method stops execution and the exception is thrown down to the previous method on the call stack.

Example:

Here is code segment showing how to use multiple try/catch statements.

```
try
{
  file = new FileInputStream(fileName);
  x = (byte) file.read();
}catch(IOException i)
{
  i.printStackTrace();
  return -1;
}catch(FileNotFoundException f) //Not valid!
{
  f.printStackTrace();
  return -1;
}
```

The throws/throw Keywords:

If a method does not handle a checked exception, the method must declare it using the **throws** keyword. The throws keyword appears at the end of a method's signature.

You can throw an exception, either a newly instantiated one or an exception that you just caught, by using the **throw** keyword. Try to understand the different in throws and throw keywords.

The following method declares that it throws a RemoteException:

```
import java.io.*;
public class className
{
```

```
  public void deposit(double amount) throws RemoteException
  {
    // Method implementation
    throw new RemoteException();
  }
  //Remainder of class definition
}
```

A method can declare that it throws more than one exception, in which case the exceptions are declared in a list separated by commas. For example, the following method declares that it throws a RemoteException and an InsufficientFundsException:

```
import java.io.*;
public class className
{
  public void withdraw(double amount) throws RemoteException,
                              InsufficientFundsException
  {
     // Method implementation
  }
  //Remainder of class definition
}
```

The finally Keyword

The finally keyword is used to create a block of code that follows a try block. A finally block of code always executes, whether or not an exception has occurred.

Using a finally block allows you to run any cleanup-type statements that you want to execute, no matter what happens in the protected code.

A finally block appears at the end of the catch blocks and has the following syntax:

```
try
{
  //Protected code
```

```
}catch(ExceptionType1 e1)
{
   //Catch block
}catch(ExceptionType2 e2)
{
   //Catch block
}catch(ExceptionType3 e3)
{
   //Catch block
}finally
{
   //The finally block always executes.
}
```

Example:

```
public class ExcepTest{

   public static void main(String args[]){
      int a[] = new int[2];
      try{
         System.out.println("Access element three :" + a[3]);
      }catch(ArrayIndexOutOfBoundsException e){
         System.out.println("Exception thrown  :" + e);
      }
      finally{
         a[0] = 6;
         System.out.println("First element value: " +a[0]);
         System.out.println("The finally statement is executed");
      }
   }
}
```

This would produce following result:

```
Exception thrown  :java.lang.ArrayIndexOutOfBoundsException: 3
First element value: 6
The finally statement is executed
```

Note the following:

- A catch clause cannot exist without a try statement.
- It is not compulsory to have finally clauses when ever a try/catch block is present.
- The try block cannot be present without either catch clause or finally clause.
- Any code cannot be present in between the try, catch, finally blocks.

Declaring you own Exception:

You can create your own exceptions in Java. Keep the following points in mind when writing your own exception classes:

- All exceptions must be a child of Throwable.
- If you want to write a checked exception that is automatically enforced by the Handle or Declare Rule, you need to extend the Exception class.
- If you want to write a runtime exception, you need to extend the RuntimeException class.

We can define our own Exception class as below:

```
class MyException extends Exception{
}
```

You just need to extend the Exception class to create your own Exception class. These are considered to be checked exceptions. The following InsufficientFundsException class is a user-defined exception that extends the Exception class, making it a checked exception. An exception class is like any other class, containing useful fields and methods.

Example:

```
// File Name InsufficientFundsException.java
import java.io.*;
```

```
public class InsufficientFundsException extends Exception
{
   private double amount;
   public InsufficientFundsException(double amount)
   {
      this.amount = amount;
   }
   public double getAmount()
   {
      return amount;
   }
}
```

To demonstrate using our user-defined exception, the following CheckingAccount class contains a withdraw() method that throws an InsufficientFundsException.

```
// File Name CheckingAccount.java
import java.io.*;

public class CheckingAccount
{
   private double balance;
   private int number;
   public CheckingAccount(int number)
   {
      this.number = number;
   }
   public void deposit(double amount)
   {
      balance += amount;
   }
```

```
   public void withdraw(double amount) throws
                              InsufficientFundsException
   {
      if(amount <= balance)
      {
         balance -= amount;
      }
      else
      {
         double needs = amount - balance;
         throw new InsufficientFundsException(needs);
      }
   }
   public double getBalance()
   {
      return balance;
   }
   public int getNumber()
   {
      return number;
   }
}
```

The following BankDemo program demonstrates invoking the deposit() and withdraw() methods of CheckingAccount.

```
// File Name BankDemo.java
public class BankDemo
{
   public static void main(String [] args)
   {
```

```
      CheckingAccount c = new CheckingAccount(101);
      System.out.println("Depositing $500...");
      c.deposit(500.00);
      try
      {
         System.out.println("\nWithdrawing $100...");
         c.withdraw(100.00);
         System.out.println("\nWithdrawing $600...");
         c.withdraw(600.00);
      }catch(InsufficientFundsException e)
      {
         System.out.println("Sorry, but you are short $"
                                  + e.getAmount());
         e.printStackTrace();
      }
    }
}
```

Compile all the above three files and run BankDemo, this would produce following result:

```
Depositing $500...


Withdrawing $100...


Withdrawing $600...
Sorry, but you are short $200.0
InsufficientFundsException
        at CheckingAccount.withdraw(CheckingAccount.java:25)
        at BankDemo.main(BankDemo.java:13)
```

Common Exceptions:

In java it is possible to define two catergories of Exceptions and Errors.

- **JVM Exceptions:** - These are exceptions/errors that are exclusively or logically thrown by the JVM. Examples : NullPointerException, ArrayIndexOutOfBoundsException, ClassCastException,

- **Programmatic exceptions** . These exceptions are thrown explicitly by the application or the API programmers Examples: IllegalArgumentException, IllegalStateException

PRACTICAL – 6

Aim . . Multithreading

(a). Create thread using Thread class and Runnable interface

(b). Synchronized keyword

Software Required: JDK1.7

Theory:

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, and each thread defines a separate path of execution.

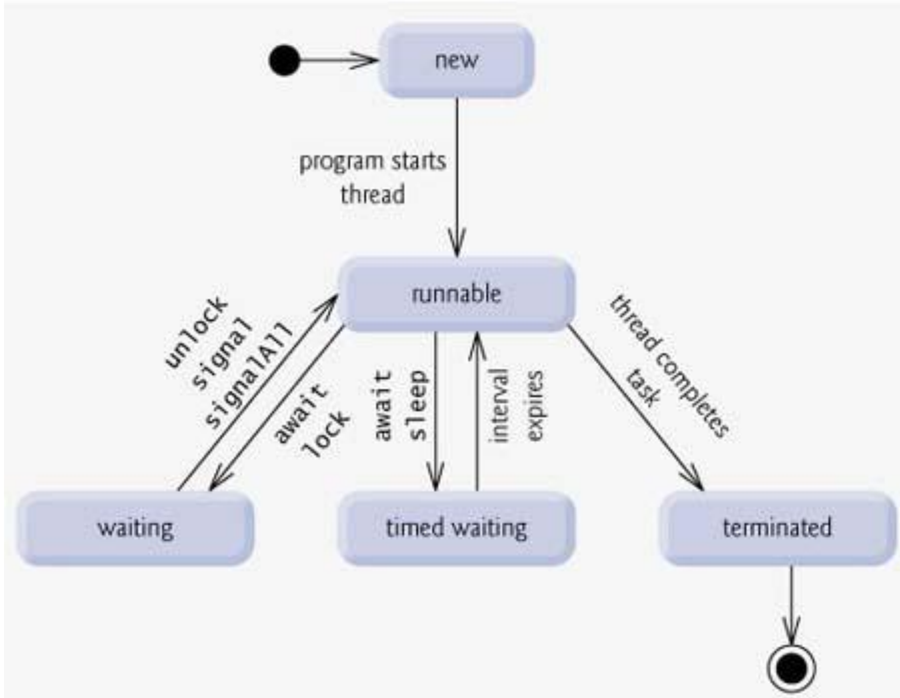
A multithreading is a specialized form of multitasking. Multitasking threads require less overhead than multitasking processes.

I need to define another term related to threads: **process:** A process consists of the memory space allocated by the operating system that can contain one or more threads. A thread cannot exist on its own; it must be a part of a process. A process remains running until all of the non-daemon threads are done executing.

Multithreading enables you to write very efficient programs that make maximum use of the CPU, because idle time can be kept to a minimum.

Life Cycle of a Thread:

A thread goes through various stages in its life cycle. For example, a thread is born, started, runs, and then dies. Following diagram shows complete life cycle of a thread.



Above mentioned stages are explained here:

- **New:** A new thread begins its life cycle in the new state. It remains in this state until the program starts the thread. It is also referred to as a born thread.

- **Runnable:** After a newly born thread is started, the thread becomes runnable. A thread in this state is considered to be executing its task.

- **Waiting:** Sometimes a thread transitions to the waiting state while the thread waits for another thread to perform a task.A thread transitions back to the runnable state only when another thread signals the waiting thread to continue executing.

- **Timed waiting:** A runnable thread can enter the timed waiting state for a specified interval of time. A thread in this state transitions back to the runnable state when that time interval expires or when the event it is waiting for occurs.

- **Terminated:** A runnable thread enters the terminated state when it completes its task or otherwise terminates.

Thread Priorities:

Every Java thread has a priority that helps the operating system determine the order in which threads are scheduled.

Java priorities are in the range between MIN_PRIORITY (a constant of 1) and MAX_PRIORITY (a constant of 10). By default, every thread is given priority NORM_PRIORITY (a constant of 5).

Threads with higher priority are more important to a program and should be allocated processor time before lower-priority threads. However, thread priorities cannot guarantee the order in which threads execute and very much platform dependentant.

Creating a Thread:

Java defines two ways in which this can be accomplished:

- You can implement the Runnable interface.
- You can extend the Thread class, itself.

Create Thread by Implementing Runnable:

The easiest way to create a thread is to create a class that implements the **Runnable** interface.

To implement Runnable, a class need only implement a single method called **run( )**, which is declared like this:

```
public void run( )
```

You will define the code that constitutes the new thread inside run() method. It is important to understand that run() can call other methods, use other classes, and declare variables, just like the main thread can.

After you create a class that implements Runnable, you will instantiate an object of type Thread from within that class. Thread defines several constructors. The one that we will use is shown here:

```
Thread(Runnable threadOb, String threadName);
```

Here *threadOb* is an instance of a class that implements the Runnable interface and the name of the new thread is specified by *threadName*.

After the new thread is created, it will not start running until you call its **start( )** method, which is declared within Thread. The start( ) method is shown here:

```
void start( );
```

Example:

Here is an example that creates a new thread and starts it running:

```
// Create a new thread.
```

```
class NewThread implements Runnable {
   Thread t;
   NewThread() {
      // Create a new, second thread
      t = new Thread(this, "Demo Thread");
      System.out.println("Child thread: " + t);
      t.start(); // Start the thread
   }

   // This is the entry point for the second thread.
   public void run() {
      try {
         for(int i = 5; i > 0; i--) {
            System.out.println("Child Thread: " + i);
            // Let the thread sleep for a while.
            Thread.sleep(500);
         }
     } catch (InterruptedException e) {
         System.out.println("Child interrupted.");
     }
     System.out.println("Exiting child thread.");
   }
}

public class ThreadDemo {
   public static void main(String args[]) {
      new NewThread(); // create a new thread
      try {
         for(int i = 5; i > 0; i--) {
           System.out.println("Main Thread: " + i);
```

```
          Thread.sleep(1000);
        }
      } catch (InterruptedException e) {
        System.out.println("Main thread interrupted.");
      }
      System.out.println("Main thread exiting.");
   }
}
```

This would produce following result:

```
Child thread: Thread[Demo Thread,5,main]
Main Thread: 5
Child Thread: 5
Child Thread: 4
Main Thread: 4
Child Thread: 3
Child Thread: 2
Main Thread: 3
Child Thread: 1
Exiting child thread.
Main Thread: 2
Main Thread: 1
Main thread exiting.
```

Create Thread by Extending Thread:

The second way to create a thread is to create a new class that extends **Thread**, and then to create an instance of that class.

The extending class must override the **run( )** method, which is the entry point for the new thread. It must also call **start( )** to begin execution of the new thread.

Example:

Here is the preceding program rewritten to extend Thread:

```
// Create a second thread by extending Thread
class NewThread extends Thread {
  NewThread() {
    // Create a new, second thread
    super("Demo Thread");
    System.out.println("Child thread: " + this);
    start(); // Start the thread
  }

  // This is the entry point for the second thread.
  public void run() {
    try {
      for(int i = 5; i > 0; i--) {
        System.out.println("Child Thread: " + i);
                                    // Let the thread sleep for a while.
        Thread.sleep(500);
      }
    } catch (InterruptedException e) {
      System.out.println("Child interrupted.");
    }
    System.out.println("Exiting child thread.");
  }
}

public class ExtendThread {
  public static void main(String args[]) {
    new NewThread(); // create a new thread
```

```
      try {
         for(int i = 5; i > 0; i--) {
            System.out.println("Main Thread: " + i);
            Thread.sleep(1000);
         }
      } catch (InterruptedException e) {
         System.out.println("Main thread interrupted.");
      }
      System.out.println("Main thread exiting.");
   }
}
```

This would produce following result:

```
Child thread: Thread[Demo Thread,5,main]
Main Thread: 5
Child Thread: 5
Child Thread: 4
Main Thread: 4
Child Thread: 3
Child Thread: 2
Main Thread: 3
Child Thread: 1
Exiting child thread.
Main Thread: 2
Main Thread: 1
Main thread exiting.
```

Thread Methods:

Following is the list of important medthods available in the Thread class.

SN	Methods with Description
1	**public void start()** Starts the thread in a separate path of execution, then invokes the run() method on this Thread object.
2	**public void run()** If this Thread object was instantiated using a separate Runnable target, the run() method is invoked on that Runnable object.
3	**public final void setName(String name)** Changes the name of the Thread object. There is also a getName() method for retrieving the name.
4	**public final void setPriority(int priority)** Sets the priority of this Thread object. The possible values are between 1 and 10.
5	**public final void setDaemon(boolean on)** A parameter of true denotes this Thread as a daemon thread.
6	**public final void join(long millisec)** The current thread invokes this method on a second thread, causing the current thread to block until the second thread terminates or the specified number of milliseconds passes.
7	**public void interrupt()** Interrupts this thread, causing it to continue execution if it was blocked for any reason.
8	**public final boolean isAlive()** Returns true if the thread is alive, which is any time after the thread has been started but before it runs to completion.

The previous methods are invoked on a particular Thread object. The following methods in the Thread class are static. Invoking one of the static methods performs the operation on the currently running thread

SN	Methods with Description
1	**public static void yield()** Causes the currently running thread to yield to any other threads of the same priority that are

	waiting to be scheduled
2	**public static void sleep(long millisec)** Causes the currently running thread to block for at least the specified number of milliseconds
3	**public static boolean holdsLock(Object x)** Returns true if the current thread holds the lock on the given Object.
4	**public static Thread currentThread()** Returns a reference to the currently running thread, which is the thread that invokes this method.
5	**public static void dumpStack()** Prints the stack trace for the currently running thread, which is useful when debugging a multithreaded application.

Example-1
This example is making your own Exception class by using constructor.

```
// this is a user define exception class etend by Exception class
class Myexception extends Exception
{
public Myexception(int i)
{
System.out.println("you " +i +" entered It exceeding the limit");
}

}

public class ExceptionTest
{
public void show(int i) throws Myexception
{
if(i>100)
throw new Myexception(i);
else
System.out.println(+i+" is less then 100 it is ok");
}

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

```
int i=Integer.parseInt(args[0]);
int j=Integer.parseInt(args[1]);
ExceptionTest t=new ExceptionTest();
try{
t.show(i);
t.show(j);
}
catch(Throwable e)
{
System.out.println("catched exception is"+e);
}
}
}
```

<u>JAVA APPLETS</u>

Applet Basics

All applets are subclasses of **Applet**. Thus, all applets must import **java.applet**. Applets must also import **java.awt**. Recall that AWT stands for the Abstract Window Toolkit. Since all applets run in a window, it is necessary to include support for that window. Applets are not executed by the console-based Java run-time interpreter. Rather, they are executed by either a Web browser or an applet viewer. The figures shown in this chapter were created with the standard applet viewer, called **applet viewer**, provided by the SDK. But you can use any applet viewer or browser you like. Execution of an applet does not begin at **main( )**. Actually, few applets even have **main( )** methods. Instead, execution of an applet is started and controlled with an entirely different mechanism, which will be explained shortly. Output to your applet's window is not performed by **System.out.println( )**. Rather, it is handled with various AWT methods, such as **drawString( )**, which outputs a string to a specified X,Ylocation. Input is also handled differently than in an application. Once an applet has been compiled, it is included in an HTML file using the APPLET tag. The applet will be executed by a Java-enabled web browser when it encounters the APPLET tag within the HTML file. To view and test an applet more conveniently, simply include a comment at the head of your Java source code file that contains the APPLET tag. This way, your code is documented with the necessary HTML statements needed by your applet, and you can test the compiled applet by starting the applet viewer with your Java source code file specified as the target.

Here is an example of such a comment:

```
/*

<applet code="MyApplet" width=200 height=60>

</applet>

*/
```

This comment contains an APPLET tag that will run an applet called **MyApplet** in a window that is 200 pixels wide and 60 pixels high.

Running Applets

If your program is an applet, then you need an HTML file to run it.

<applet code="MyApplet.class" width=300 height=200>
</applet>

The "appletviewer" command from the SDK can then be used to view the applet. If the file name is test.html, use the command

appletviewer test.html

This will only show the applet. It will ignore any text or images in the HTML file. In fact, need in the HTML file is a single applet tag. The applet will be run in a resizable window. Note also that applet can use standard output, System.out, to write messages to the command window. This can be useful for debugging applet.

Use the appletviewer command on any file, or even on a web page address. It will find all the applet tags in the file, and will open a window for each applet. If using a Web browser that does not support Java 2, one can use appletviewer to see the applets. For example, to see the applets, use the command

appletviewer http://../yourcode.html

Of course, it's also possible to view applets in a Web browser. Just open the html file that contains the applet tag for your applet. One problem with this is that if you make changes to the applet, quit the browser and restart it in order to get the changes to take effect. The browser's Reload command might not cause the modified applet to be loaded.

PRACTICAL – 7

Aim : Draw Applets

(a). Using drawrect,fillrect like functions

(b). How to make an applet respond to setup parameters specified in the document

Software Required: JDK1.7

Theory:

Applet An applet is a Java program that runs in a Web browser. An applet can be a fully functional Java application because it has the entire Java API at its disposal.

There are some important differences between an applet and a standalone Java application, including the following:

- An applet is a Java class that extends the java.applet.Applet class.
- A main() method is not invoked on an applet, and an applet class will not define main().
- Applets are designed to be embedded within an HTML page.
- When a user views an HTML page that contains an applet, the code for the applet is downloaded to the user's machine.
- A JVM is required to view an applet. The JVM can be either a plug-in of the Web browser or a separate runtime environment.
- The JVM on the user's machine creates an instance of the applet class and invokes various methods during the applet's lifetime.
- Applets have strict security rules that are enforced by the Web browser. The security of an applet is often referred to as sandbox security, comparing the applet to a child playing in a sandbox with various rules that must be followed.
- Other classes that the applet needs can be downloaded in a single Java Archive (JAR) file.

Life Cycle of an Applet:

Four methods in the Applet class give you the framework on which you build any serious applet:

- **init:** This method is intended for whatever initialization is needed for your applet. It is called after the param tags inside the applet tag have been processed.
- **start:** This method is automatically called after the browser calls the init method. It is also called whenever the user returns to the page containing the applet after having gone off to other pages.

- **stop:** This method is automatically called when the user moves off the page on which the applet sits. It can, therefore, be called repeatedly in the same applet.

- **destroy:** This method is only called when the browser shuts down normally. Because applets are meant to live on an HTML page, you should not normally leave resources behind after a user leaves the page that contains the applet.

- **paint:** Invoked immediately after the start() method, and also any time the applet needs to repaint itself in the browser. The paint() method is actually inherited from the java.awt.

A "Hello, World" Applet:

The following is a simple applet named HelloWorldApplet.java:

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


public class HelloWorldApplet extends Applet
{
  public void paint (Graphics g)
  {
    g.drawString ("Hello World", 25, 50);
  }
}
```

These import statements bring the classes into the scope of our applet class:

- java.applet.Applet.
- java.awt.Graphics.

Without those import statements, the Java compiler would not recognize the classes Applet and Graphics, which the applet class refers to.

The Applet CLASS:

Every applet is an extension of the *java.applet.Applet class*. The base Applet class provides methods that a derived Applet class may call to obtain information and services from the browser context.

These include methods that do the following:

- Get applet parameters
- Get the network location of the HTML file that contains the applet
- Get the network location of the applet class directory
- Print a status message in the browser
- Fetch an image
- Fetch an audio clip
- Play an audio clip
- Resize the applet

Additionally, the Applet class provides an interface by which the viewer or browser obtains information about the applet and controls the applet's execution. The viewer may:

- request information about the author, version and copyright of the applet
- request a description of the parameters the applet recognizes
- initialize the applet
- destroy the applet
- start the applet's execution
- stop the applet's execution

The Applet class provides default implementations of each of these methods. Those implementations may be overridden as necessary.

The "Hello, World" applet is complete as it stands. The only method overridden is the paint method.

Invoking an Applet:

An applet may be invoked by embedding directives in an HTML file and viewing the file through an applet viewer or Java-enabled browser.

The <applet> tag is the basis for embedding an applet in an HTML file. Below is an example that invokes the "Hello, World" applet:

```
<html>

<title>The Hello, World Applet</title>
```

```
<hr>
<applet code="HelloWorldApplet.class" width="320" height="120">
If your browser was Java-enabled, a "Hello, World"
message would appear here.
</applet>
<hr>
</html>
```

Based on the above examples, here is the live applet example: Applet Example.

Note: You can refer to HTML Applet Tag to understand more about calling applet from HTML.

The code attribute of the <applet> tag is required. It specifies the Applet class to run. Width and height are also required to specify the initial size of the panel in which an applet runs. The applet directive must be closed with a </applet> tag.

If an applet takes parameters, values may be passed for the parameters by adding <param> tags between <applet> and </applet>. The browser ignores text and other tags between the applet tags.

Non-Java-enabled browsers do not process <applet> and </applet>. Therefore, anything that appears between the tags, not related to the applet, is visible in non-Java-enabled browsers.

The viewer or browser looks for the compiled Java code at the location of the document. To specify otherwise, use the codebase attribute of the <applet> tag as shown:

```
<applet codebase="http://amrood.com/applets"
code="HelloWorldApplet.class" width="320" height="120">
```

If an applet resides in a package other than the default, the holding package must be specified in the code attribute using the period character (.) to separate package/class components. For example:

```
<applet code="mypackage.subpackage.TestApplet.class"
        width="320" height="120">
```

Getting Applet Parameters:

The following example demonstrates how to make an applet respond to setup parameters specified in the document. This applet displays a checkerboard pattern of black and a second color.

The second color and the size of each square may be specified as parameters to the applet within the document.

CheckerApplet gets its parameters in the init() method. It may also get its parameters in the paint() method. However, getting the values and saving the settings once at the start of the applet, instead of at every refresh, is convenient and efficient.

The applet viewer or browser calls the init() method of each applet it runs. The viewer calls init() once, immediately after loading the applet. (Applet.init() is implemented to do nothing.) Override the default implementation to insert custom initialization code.

The Applet.getParameter() method fetches a parameter given the parameter's name (the value of a parameter is always a string). If the value is numeric or other non-character data, the string must be parsed.

The following is a skeleton of CheckerApplet.java:

```
import java.applet.*;
import java.awt.*;
public class CheckerApplet extends Applet
{
  int squareSize = 50;// initialized to default size
  public void init () {}
  private void parseSquareSize (String param) {}
  private Color parseColor (String param) {}
  public void paint (Graphics g) {}
}
```

Here are CheckerApplet's init() and private parseSquareSize() methods:

```
public void init ()
{
  String squareSizeParam = getParameter ("squareSize");
```

```
   parseSquareSize (squareSizeParam);
   String colorParam = getParameter ("color");
   Color fg = parseColor (colorParam);
   setBackground (Color.black);
   setForeground (fg);
}
private void parseSquareSize (String param)
{
   if (param == null) return;
   try {
      squareSize = Integer.parseInt (param);
   }
   catch (Exception e) {
     // Let default value remain
   }
}
```

The applet calls parseSquareSize() to parse the squareSize parameter. parseSquareSize() calls the library method Integer.parseInt(), which parses a string and returns an integer. Integer.parseInt() throws an exception whenever its argument is invalid.

Therefore, parseSquareSize() catches exceptions, rather than allowing the applet to fail on bad input.

The applet calls parseColor() to parse the color parameter into a Color value. parseColor() does a series of string comparisons to match the parameter value to the name of a predefined color. You need to implement these methods to make this applet works.

Specifying Applet Parameters:

The following is an example of an HTML file with a CheckerApplet embedded in it. The HTML file specifies both parameters to the applet by means of the <param> tag.

```
<html>
<title>Checkerboard Applet</title>
<hr>
```

```
<applet code="CheckerApplet.class" width="480" height="320">
<param name="color" value="blue">
<param name="squaresize" value="30">
</applet>
<hr>
</html>
```

Note: Parameter names are not case sensitive.

Application Conversion to Applets:

It is easy to convert a graphical Java application (that is, an application that uses the AWT and that you can start with the java program launcher) into an applet that you can embed in a web page.

Here are the specific steps for converting an application to an applet.

- Make an HTML page with the appropriate tag to load the applet code.
- Supply a subclass of the JApplet class. Make this class public. Otherwise, the applet cannot be loaded.
- Eliminate the main method in the application. Do not construct a frame window for the application. Your application will be displayed inside the browser.
- Move any initialization code from the frame window constructor to the init method of the applet. You don't need to explicitly construct the applet object.the browser instantiates it for you and calls the init method.
- Remove the call to setSize; for applets, sizing is done with the width and height parameters in the HTML file.
- Remove the call to setDefaultCloseOperation. An applet cannot be closed; it terminates when the browser exits.
- If the application calls setTitle, eliminate the call to the method. Applets cannot have title bars. (You can, of course, title the web page itself, using the HTML title tag.)
- Don't call setVisible(true). The applet is displayed automatically.

PRACTICAL – 8

Aim : . Draw Applets with Event handling
(a). Using MouseListener
(b). On Button using Action Listener
(c). On Key using key Listenser
Software Required: JDK1.7

Theory:
Event Handling:

Applets inherit a group of event-handling methods from the Container class. The Container class defines several methods, such as processKeyEvent and processMouseEvent, for handling particular types of events, and then one catch-all method called processEvent.

Inorder to react an event, an applet must override the appropriate event-specific method.

```
import java.awt.event.MouseListener;
import java.awt.event.MouseEvent;
import java.applet.Applet;
import java.awt.Graphics;

public class ExampleEventHandling extends Applet
                                    implements MouseListener {

   StringBuffer strBuffer;

   public void init() {
            addMouseListener(this);
            strBuffer = new StringBuffer();
      addItem("initializing the apple ");
   }

   public void start() {
      addItem("starting the applet ");
```

```
}

public void stop() {
    addItem("stopping the applet ");
}

public void destroy() {
    addItem("unloading the applet");
}

void addItem(String word) {
    System.out.println(word);
    strBuffer.append(word);
    repaint();
}

public void paint(Graphics g) {
        //Draw a Rectangle around the applet's display area.
    g.drawRect(0, 0,
                    getWidth() - 1,
                    getHeight() - 1);

        //display the string inside the rectangle.
    g.drawString(strBuffer.toString(), 10, 20);
}



public void mouseEntered(MouseEvent event) {
}
public void mouseExited(MouseEvent event) {
```

```
    }
    public void mousePressed(MouseEvent event) {
    }
    public void mouseReleased(MouseEvent event) {
    }

    public void mouseClicked(MouseEvent event) {
            addItem("mouse clicked! ");
    }
}
```

Now let us call this applet as follows:

```
<html>
<title>Event Handling</title>
<hr>
<applet code="ExampleEventHandling.class"
width="300" height="300">
</applet>
<hr>
</html>
```

Initially the applet will display "initializing the applet. Starting the applet." Then once you click inside the rectangle "mouse clicked" will be displayed as well.

Based on the above examples, here is the live applet example: Applet Example.

Displaying Images:

An applet can display images of the format GIF, JPEG, BMP, and others. To display an image within the applet, you use the drawImage() method found in the java.awt.Graphics class.

Following is the example showing all the steps to show images:

```
import java.applet.*;
```

```
import java.awt.*;
import java.net.*;
public class ImageDemo extends Applet
{
  private Image image;
  private AppletContext context;
  public void init()
  {
      context = this.getAppletContext();
      String imageURL = this.getParameter("image");
      if(imageURL == null)
      {
         imageURL = "java.jpg";
      }
      try
      {
         URL url = new URL(this.getDocumentBase(), imageURL);
         image = context.getImage(url);
      }catch(MalformedURLException e)
      {
         e.printStackTrace();
         // Display in browser status bar
         context.showStatus("Could not load image!");
      }
   }
   public void paint(Graphics g)
   {
      context.showStatus("Displaying image");
      g.drawImage(image, 0, 0, 200, 84, null);
      g.drawString("www.javalicense.com", 35, 100);
```

```
   }
}
```

Now let us call this applet as follows:

```
<html>
<title>The ImageDemo applet</title>
<hr>
<applet code="ImageDemo.class" width="300" height="200">
<param name="image" value="java.jpg">
</applet>
<hr>
</html>
```

Based on the above examples, here is the live applet example: Applet Example.

Playing Audio:

An applet can play an audio file represented by the AudioClip interface in the java.applet package. The AudioClip interface has three methods, including:

- **public void play():** Plays the audio clip one time, from the beginning.
- **public void loop():** Causes the audio clip to replay continually.
- **public void stop():** Stops playing the audio clip.

To obtain an AudioClip object, you must invoke the getAudioClip() method of the Applet class. The getAudioClip() method returns immediately, whether or not the URL resolves to an actual audio file. The audio file is not downloaded until an attempt is made to play the audio clip.

Following is the example showing all the steps to play an audio:

```
import java.applet.*;
import java.awt.*;
import java.net.*;
public class AudioDemo extends Applet
{
```

```
private AudioClip clip;
private AppletContext context;
public void init()
{
  context = this.getAppletContext();
  String audioURL = this.getParameter("audio");
  if(audioURL == null)
  {
    audioURL = "default.au";
  }
  try
  {
    URL url = new URL(this.getDocumentBase(), audioURL);
    clip = context.getAudioClip(url);
  }catch(MalformedURLException e)
  {
    e.printStackTrace();
    context.showStatus("Could not load audio file!");
  }
}
public void start()
{
  if(clip != null)
  {
    clip.loop();
  }
}
public void stop()
{
  if(clip != null)
```

```
    {
      clip.stop();
    }
  }
}
```

Now let us call this applet as follows:

```
<html>
<title>The ImageDemo applet</title>
<hr>
<applet code="ImageDemo.class" width="0" height="0">
<param name="audio" value="test.wav">
</applet>
<hr>
</html>
```

8.Event handling in java

The event listener is the feature of java that handles the several events for the several objects, Such as: MouseEvent, KeyEvent, TextEvent, InputEvent etc. Classes for helping in implementing event listeners are present in the *java.awt.event.*;* package. So, to use the events handling in your application import the *java.awt.event.*;* package. This example illustrates that how to handle several events fired on the several objects.

In this example you will see that how to use the event listener and to perform appropriate tasks. In this example the EventListeners.java is our applet class which implements the ActionListener interface. Here four buttons and integer types variables have been used with specific values to perform the Addition, Subtraction, Multiplication and Division operations. All these operations are controlled by the events generated by these buttons. The Text Area named txtArea holds the result of the operation. There are two methods like init()and actionPerformed() have been used in this program for performing the whole operation.

To handle the events generated by these buttons you add action listeners

e.g. object_name.addActionListener(this);.

When the action event occurs, that object's actionPerformed method is invoked.

actionPerformed(**ActionEvent**e)

Here is the java code of the program:

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

public class EventListeners extends Applet implements ActionListener{
TextArea txtArea;
String Add, Subtract,Multiply,Divide;
int i = 10, j = 20, sum =0,Sub=0,Mul = 0,Div = 0;

public void init(){
txtArea = new TextArea(10,20);
txtArea.setEditable(false);
add(txtArea,"center");
Button b = new Button("Add");
Button c = new Button("Subtract");
Button d = new Button("Multiply");
Button e = new Button("Divide");
b.addActionListener(this);
c.addActionListener(this);
d.addActionListener(this);
e.addActionListener(this);

add(b);
add(c);
add(d);
add(e);
}


public void actionPerformed(ActionEvent e){
sum = i + j;
txtArea.setText("");
txtArea.append("i = "+ i + "\t" + "j = " + j + "\n");
Button source = (Button)e.getSource();
if(source.getLabel() == "Add"){
```

```
txtArea.append("Sum : " + sum + "\n");
}

if(i >j){
Sub = i - j;
}
else{
Sub = j - i;
}
if(source.getLabel() == "Subtract"){
txtArea.append("Sub : " + Sub + "\n");
}

Mul = i*j;
if(source.getLabel() == "Multiply"){
txtArea.append("Mul = " + Mul + "\n");
}

if(i > j){
Div = i / j;
}
else{
Div = j / i;
}

if(source.getLabel() == "Divide"){
txtArea.append("Divide = " + Div);
}
}
}
```

Here is the HTML code of the program :

```
<HTML>
<BODY>
<APPLET CODE ="EventListeners" WIDTH="800"HEIGHT="500"></APPLET>
</BODY>
</HTML>
```

To handle the events generated by these buttons you add action listeners

e.g. object_name.addActionListener(this);.

When the action event occurs, that object's actionPerformed method is invoked.

actionPerformed(ActionEvent e)

Here is the java code of the program:

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

public class EventListeners extends Applet implements ActionListener{
  TextArea txtArea;
  String Add, Subtract,Multiply,Divide;
  int i = 10, j = 20, sum =0,Sub=0,Mul = 0,Div = 0;

  public void init(){
  txtArea = new TextArea(10,20);
  txtArea.setEditable(false);
  add(txtArea,"center");
  Button b = new Button("Add");
  Button c = new Button("Subtract");
  Button d = new Button("Multiply");
  Button e = new Button("Divide");
  b.addActionListener(this);
  c.addActionListener(this);
  d.addActionListener(this);
  e.addActionListener(this);

  add(b);
  add(c);
  add(d);
  add(e);
  }


  public void actionPerformed(ActionEvent e){
  sum = i + j;
  txtArea.setText("");
  txtArea.append("i = "+ i + "\t" + "j = " + j + "\n");
  Button source = (Button)e.getSource();
  if(source.getLabel() == "Add"){
  txtArea.append("Sum : " + sum + "\n");
  }

  if(i >j){
```

```
  Sub = i - j;
  }
  else{
  Sub = j - i;
  }
  if(source.getLabel() == "Subtract"){
  txtArea.append("Sub : " + Sub + "\n");
  }

  Mul = i*j;
  if(source.getLabel() == "Multiply"){
  txtArea.append("Mul = " + Mul + "\n");
  }

  if(i > j){
  Div = i / j;
  }
  else{
  Div = j / i;
  }

  if(source.getLabel() == "Divide"){
  txtArea.append("Divide = " + Div);
  }
  }
}
```

Here is the HTML code of the program :

```
<HTML>
 <BODY>
 <APPLET CODE ="EventListeners" WIDTH="800"HEIGHT="500"></APPLET>
 </BODY>
 </HTML>
```

PRACTICAL – 9

Aim: Write a program using Layout Managers
(a). Grid Layout,
(b). Border Layout
Software Required: JDK1.7

Theory:
Layout Managers:

There are various Controls in Java like Button, Checkbox, Lists, Scrollbars, Text Fields, and Text Area etc. All of these components have been positioned by the default layout manager. A layout manager automatically arranges the controls within a window by using some type of algorithm. In GUI environments, such as Windows, you can layout your controls by hand. It is possible to lay out Java controls by hand, too, but we do not do so for two main reasons. First, it is very tedious to manually lay out a large number of components. Second, sometimes the width and height information is not yet available when you need to arrange some control, because the native toolkit components have not been realized. This is a chicken-and-egg situation; it is confusing to figure out when it is okay to use the size of a given component to position it relative to another.

Each Container object has a layout manager associated with it. A layout manager is an instance of any class that implements the LayoutManager interface. The layout manager is set by the setLayout( ) method. If no call to setLayout( ) is made, then the default layout manager is used. Whenever a container is resized or sized for the first time, the layout manager is used to position each of the components within it.

The setLayout( ) method has the following general form:

void setLayout(LayoutManager layoutObj)

Here, layoutObj is a reference to the desired layout manager. If you wish to disable the layout manager and position components manually, pass null for layoutObj. If you do this, you will need to determine the shape and position of each component manually, using the setBounds( ) method defined by Component.

Each layout manager keeps track of a list of components that are stored by their names. The layout manager is notified each time you add a component to a container. Normally, you will want to use a layout manager.Whenever the container needs to be resized, the layout manager is consulted via its minimumLayoutSize( ) and preferredLayoutSize( ) methods. Each component that is being managed by a layout manager contains the getPreferredSize( ) and getMinimumSize( ) methods. These return the preferred and minimum size required to display each component.. You may override these methods for controls that you subclass. Default values are provided otherwise.

Java has several predefined LayoutManager classes.

- FlowLayout
- BorderLayout
- Insets
- GridLayout
- CardLayout

i. FlowLayout

FlowLayout is the default layout manager. FlowLayout implements a simple layout style, which is similar to how words flow in a text editor. Components are laid out from the upper-left corner, left to right and top to bottom. When no more components fit on a line, the next one appears on the next line. A small space is left between each component, above and below, as well as left and right.

The constructors for FlowLayout are shown below:

1. FlowLayout( )
2. FlowLayout(int how)
3. FlowLayout(int how, int horz, int vert)

The first form creates the default layout, which centers components and leaves five pixels of space between each component.

The second form lets you specify how each line is aligned. Valid values for how are as follows:

- FlowLayout.LEFT
- FlowLayout.CENTER
- FlowLayout.RIGHT

These values specify left, center, and right alignment, respectively. The third form allows specifying the horizontal and vertical space left between components in horz and vert, respectively.

Here is another type of the CheckboxDemo applet shown in the previous articles, such that it uses left-aligned flow layout.

Code:

```
import java.awt.*;
import java.awt.event.*;
import java.applet.*;
/*
<applet code="FlowLayoutTest" width=250 height=200>
</applet>
*/
public class FlowLayoutTest extends Applet implements ItemListener
{
    String str = "";
    Checkbox Go4expert, codeitwell,mbaguys;
    Label l1;
```

```
    public void init()
    {
      // set left-aligned flow layout
      setLayout(new FlowLayout(FlowLayout.LEFT));
      l1=new Label("Select the Best site:");
      Go4expert = new Checkbox("Go4expert.com", null, true);
      codeitwell = new Checkbox("codeitwell.com ");
      mbaguys = new Checkbox("mbaguys.net");
      add(l1);
      add(Go4expert);
      add(codeitwell);
      add(mbaguys);
      // register to receive item events

      Go4expert.addItemListener(this);
      codeitwell.addItemListener(this);
      mbaguys.addItemListener(this);
    }
    // Repaint when status of a check box changes.
    public void itemStateChanged(ItemEvent ie)
    {
      repaint();
    }
    // Display current state of the check boxes.
    public void paint(Graphics g)
    {
      str = "Go4expert.com : " + Go4expert.getState();
      g.drawString(str, 6, 100);
      str = "codeitwell.com: " + codeitwell.getState();
      g.drawString(str, 6, 120);
      str = "mbaguys.net : " + mbaguys.getState();
      g.drawString(str, 6, 140);
    }
}
```

Output would be as shown below:-

ii. BorderLayout

The BorderLayout class implements a common layout style for top-level windows. It has four narrow, fixed-width components at the edges and one large area in the center. The four sides are referred to as north, south, east, and west. The middle area is called the center.

The constructors defined by BorderLayout are shown below:

1. BorderLayout( )
2. BorderLayout(int horz, int vert)

The first form creates a default border layout.

The second allows you to specify the horizontal and vertical space left between components in horz and vert, respectively.

BorderLayout defines the following constants that specify the regions:

- BorderLayout.CENTER
- BorderLayout.SOUTH
- BorderLayout.EAST
- BorderLayout.WEST
- BorderLayout.NORTH

When adding components, you will use these constants with the following form of add( ), which is defined by Container:

void add(Component compObj, Object region)

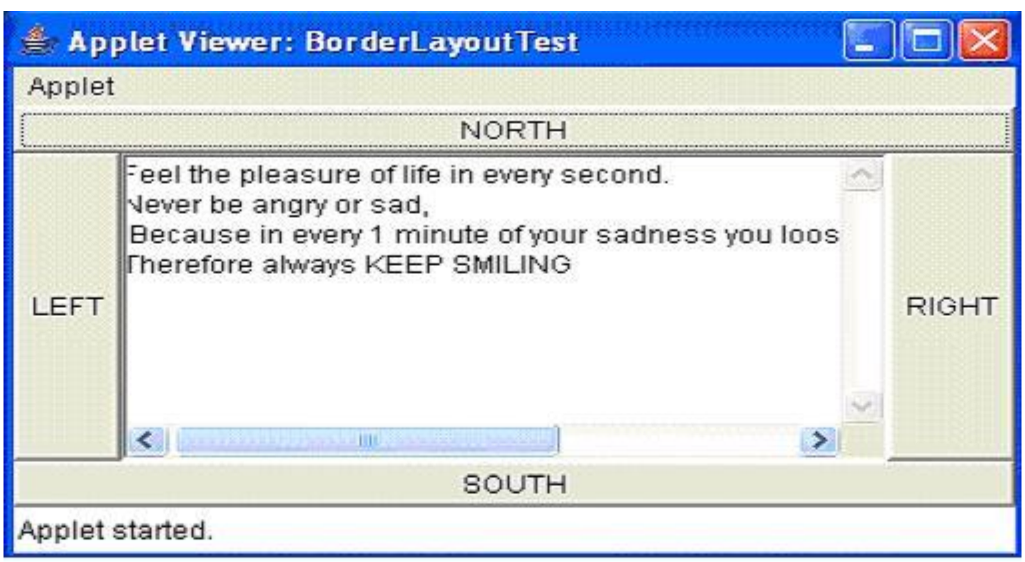
Here, compObj is the component to be added, and region specifies where the component will be added.

Example:

Here is an example of a BorderLayout with a component in each layout area:
Code:

```
import java.awt.*;
import java.applet.*;
import java.util.*;
/*
<applet code="BorderLayoutTest" width=400 height=200>
</applet>
*/
public class BorderLayoutTest extends Applet
{
   public void init()
   {
      setLayout(new BorderLayout());
      add(new Button("NORTH"), BorderLayout.NORTH);
      add(new Button("SOUTH"), BorderLayout.SOUTH);
      add(new Button("RIGHT"), BorderLayout.EAST);
      add(new Button("LEFT"), BorderLayout.WEST);
      String str = "Feel the pleasure of life in every second.\n" +
      "Never be angry or sad, \n " +
      "Because in every 1 minute of your sadness " +
      "you loose 60 seconds of your hapiness.\n" +
      "Therefore always KEEP SMILING \n\n";
      add(new TextArea(str), BorderLayout.CENTER);
   }
}
```

Output would be as shown below:



iii. Insets:

iv. Sometimes you will want to leave a small amount of space between the container that holds your components and the window that contains it. To do this, override the getInsets( ) method that is defined by Container. This function returns an Insets object that contains the top, bottom, left, and right inset to be used when the container is displayed. These values are used by the layout manager to inset the components when it lays out the window.

The constructor for Insets is Insets(int top, int left, int bottom, int right)

The values passed in top, left, bottom, and right specify the amount of space between the container and its enclosing window.

The getInsets( ) method has this general form:

Insets getInsets( )

When overriding one of these methods, you must return a new Insets object that contains the inset spacing you desire.

Example:

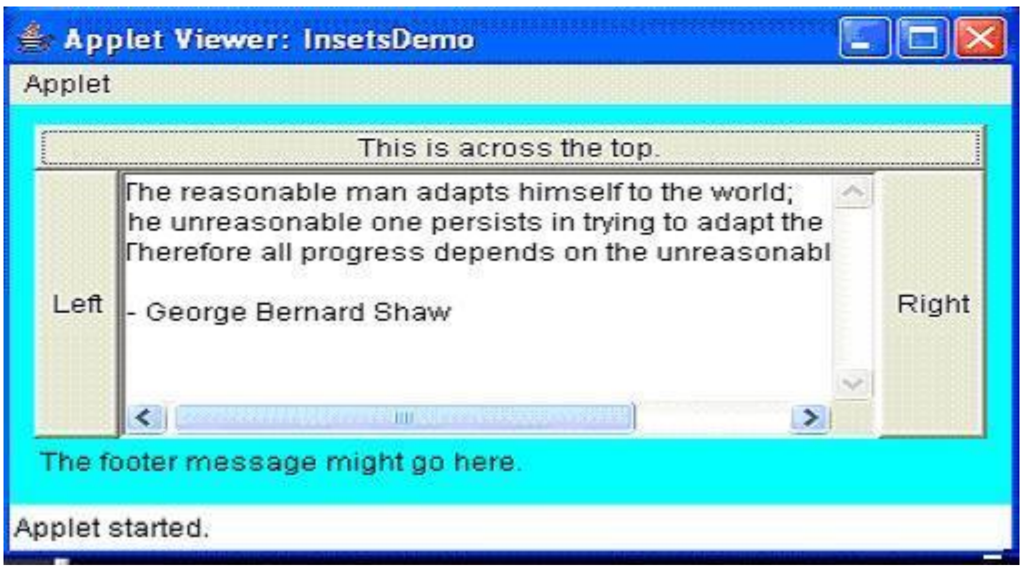
Here is the preceding BorderLayout example modified so that it insets its components ten pixels from each border. The background color has been set to cyan to help make the insets more visible.

Code:

```
// Demonstrate BorderLayout with insets.
import java.awt.*;
import java.applet.*;
import java.util.*;
/*
<applet code="InsetsDemo" width=400 height=200>
</applet>
*/
public class InsetsDemo extends Applet
{
   public void init()
   {
      // set background color so insets can be easily seen
      setBackground(Color.cyan);
      setLayout(new BorderLayout());
      add(new Button("This is across the top."),
      BorderLayout.NORTH);
      add(new Label("The footer message might go here."),
      BorderLayout.SOUTH);
      add(new Button("Right"), BorderLayout.EAST);
      add(new Button("Left"), BorderLayout.WEST);
```

```
        String msg = "The reasonable man adapts " +
        "himself to the world;\n" +
        "the unreasonable one persists in " +
        "trying to adapt the world to himself.\n" +
        "Therefore all progress depends " +
        "on the unreasonable man.\n\n" +
        " - George Bernard Shaw\n\n";
        add(new TextArea(msg), BorderLayout.CENTER);
    }
      // add insets
    public Insets getInsets()
    {
        return new Insets(10, 10, 10, 10);
    }
}
```

Output would be as shown below:



v. GridLayout

GridLayout lays out components in a two-dimensional grid. When you instantiate a GridLayout, you define the number of rows and columns.

The constructors supported by GridLayout are shown below:

1. GridLayout( )
2. GridLayout(int numRows, int numColumns)
3. GridLayout(int numRows, int numColumns, int horz, int vert)

The first form creates a single-column grid layout.

The second form creates a grid layout with the specified number of rows and columns. The third form allows you to specify the horizontal and vertical space left between components in horz and vert, respectively. Either numRows or numColumns can be zero. Specifying numRows as zero allows for unlimited-length columns. Specifying numColumns as zero allows for unlimited-length rows.

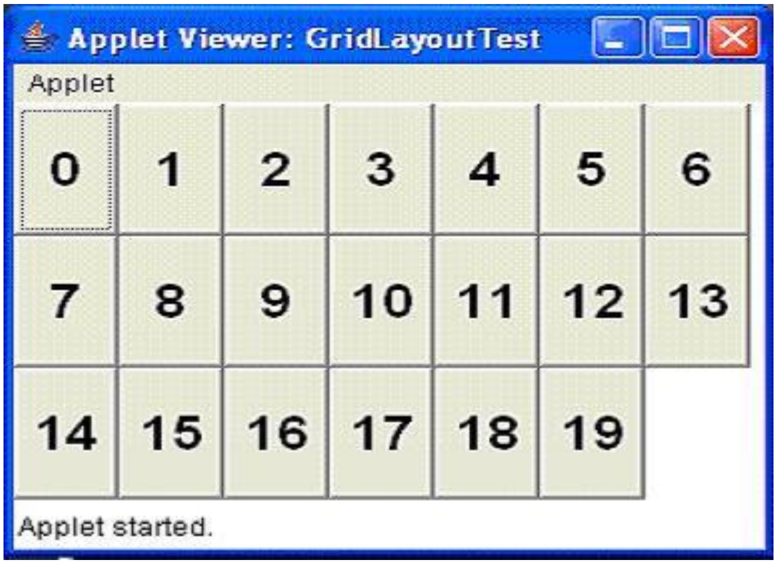
Example:

Here is a sample program that creates a 7×3 grid and fills it in with 20 buttons, each labeled with its index:

Code:

```
// Demonstrate GridLayout
import java.awt.*;
import java.applet.*;
/*
<applet code="GridLayoutTest" width=300 height=200>
</applet>
*/
public class GridLayoutTest extends Applet
{
  static final int row =3;
  static final int col =7;
  public void init()
  {
    setLayout(new GridLayout(row,col));
    setFont(new Font("SansSerif", Font.BOLD, 24));
    for(int i = 0; i < 20; i++)
    {
    add(new Button("" + i));
    }
  }
}
```

Output would be as shown below:



vi.
vii.
viii.
ix.
x.

PRACTICAL – 10

Aim: Write a program using Layout Manager as **CardLayout**
Software Required: JDK1.7

Theory:

The CardLayout class is unique among the other layout managers in that it stores several different layouts. Each layout can be thought of as being on a separate index card in a deck that can be shuffled so that any card is on top at a given time. This can be useful for user interfaces with optional components that can be dynamically enabled and disabled upon user input. We can prepare the other layouts and have them hidden, ready to be activated when needed.

CardLayout provides the following two constructors:

1. CardLayout( )
2. CardLayout(int horz, int vert)

The first form creates a default card layout.

The second form allows you to specify the horizontal and vertical space left between components in horz and vert, respectively.

Use of a card layout requires a bit more work than the other layouts. The cards are typically held in an object of type Panel. This panel must have CardLayout selected as its layout manager. The cards that form the deck are also typically objects of type Panel.

Thus, you must create a panel that contains the deck and a panel for each card in the deck. Next, you add to the appropriate panel the components that form each card. You then add these panels to the panel for which CardLayout is the layout manager. Finally, you add this panel to the main applet panel. Once these steps are complete, you must provide some way for the user to select between cards. One common approach is to include one push button for each card in the deck.

When card panels are added to a panel, they are usually given a name. Most of the time, you will use this form of add( ) when adding cards to a panel:

void add(Component panelObj, Object name);

Here, name is a string that specifies the name of the card whose panel is specified by panelObj.

After you have created a deck, your program activates a card by calling one of the following methods defined by CardLayout:

1. void first(Container deck)
2. void last(Container deck)
3. void next(Container deck)
4. void previous(Container deck)

5. void show(Container deck, String cardName)

   Here, deck is a reference to the container (usually a panel) that holds the cards, and cardName is the name of a card.

   Calling first( ) causes the first card in the deck to be shown.

   To show the last card, call last( ).
   To show the next card, call next( ).
   To show the previous card, call previous( ).
   Both next( ) and previous( ) automatically cycle back to the top or bottom of the deck, respectively.
   The show( ) method displays the card whose name is passed in cardName.

   Example:

   The following example creates a two-level card deck that allows the user to select a language. Procedural languages are displayed in one card. Object Oriented languages are displayed in the other card.
   Code:

```
// Demonstrate CardLayout.
import java.awt.*;
import java.awt.event.*;
import java.applet.*;
/*
<applet code="CardLayoutTest" width=400 height=100>
</applet>
*/
public class CardLayoutTest extends Applet implements ActionListener, MouseListener
{
  Checkbox Java, C, VB ;
  Panel langCards;
  CardLayout cardLO;
  Button OO, Other;
  public void init()
  {
    OO = new Button("ObjectOriented Languages");
    Other = new Button("Procedural Languages");
    add(OO);
    add(Other);
    cardLO = new CardLayout();
    langCards = new Panel();
    langCards.setLayout(cardLO); // set panel layout to card layout
    Java = new Checkbox("Java", null, true);
    C = new Checkbox("C");
    VB = new Checkbox("VB");
```

```
    // add OO languages check boxes to a panel
    Panel OOPan = new Panel();
    OOPan.add(Java);
    OOPan.add(VB);
    // Add other languages check boxes to a panel
    Panel otherPan = new Panel();
    otherPan.add(C);
    // add panels to card deck panel
    langCards.add(OOPan, "Object Oriented Languages");
    langCards.add(otherPan, "Procedural Languages");
    // add cards to main applet panel
    add(langCards);
    // register to receive action events
    OO.addActionListener(this);
    Other.addActionListener(this);
    // register mouse events
    addMouseListener(this);
  }
  // Cycle through panels.
  public void mousePressed(MouseEvent me)
  {
    cardLO.next(langCards);
  }
  // Provide empty implementations for the other MouseListener methods.
  public void mouseClicked(MouseEvent me)
  {
  }
  public void mouseEntered(MouseEvent me)
  {
  }
  public void mouseExited(MouseEvent me)
  {
  }
  public void mouseReleased(MouseEvent me)
  {
  }
  public void actionPerformed(ActionEvent ae)
  {
     if(ae.getSource() == OO)
     {
        cardLO.show(langCards, "Object Oriented Languages");
     }
     else
     {
        cardLO.show(langCards, "Procedural Languages");
```

```
      }
   }
}
```

Output would be as shown below:

On clicking the Procedural languages Button the following output would be displayed:

Another Example:

```
import java.awt.*;
import java.applet.Applet;
import java.awt.event.*;
/* This applet demonstrates using the CardLayout manager.
Pressing one of three buttons will cause a different "card" to be
displayed.
*/
public class Card1 extends Applet implements ActionListener
{
  Panel cardPanel;        // the container that will hold the various "cards"
  Panel firstP, secondP, thirdP;  // each of these panels will constitute the "cards"
  Panel buttonP;                  // panel to hold three buttons
  Button first, second, third;    // the three buttons
  CardLayout ourLayout;           // the card layout object

  public void init()
```

```
{
  //create cardPanel which is the panel that will contain the three "cards"
  cardPanel = new Panel();
  //create the CardLayout object
  ourLayout = new CardLayout();
  //set card Panel's layout to be our Card Layout
  cardPanel.setLayout (ourLayout);

  //create three dummy panels (the "cards") to show
  firstP = new Panel();
  firstP.setBackground(Color.blue);

  secondP = new Panel();
  secondP.setBackground(Color.yellow);

  thirdP = new Panel();
  thirdP.setBackground(Color.green);

  //create three buttons and add ActionListener
  first = new Button("First");
  first.addActionListener(this);

  second = new Button("Second");
  second.addActionListener(this);

  third = new Button("Third");
  third.addActionListener(this);

  //create Panel for the buttons and add the buttons to it
  buttonP = new Panel();   // Panel's default Layout manager is FlowLayout
  buttonP.add(first);
  buttonP.add(second);
  buttonP.add(third);

  //setLayout for applet to be BorderLayout
  this.setLayout(new BorderLayout());
  //button Panel goes South, card panels go Center
  this.add(buttonP, BorderLayout.SOUTH);
  this.add(cardPanel, BorderLayout.CENTER);

  // add the three card panels to the card panel container
  // method takes 1.) an Object (the card) 2.) an identifying String
  // first one added is the visible one when applet appears
  cardPanel.add(firstP, "First");     //blue
  cardPanel.add(secondP, "Second");   //yellow
  cardPanel.add(thirdP, "Third");     //green
```

```
  }

  //------------------------------------
  // respond to Button clicks by showing the so named Panel
  // note use of the CardLayout method show(Container, "identifying string")

  public void actionPerformed(ActionEvent e)
  {
    if (e.getSource() == first)
      ourLayout.show(cardPanel, "First");

    if (e.getSource() == second)
      ourLayout.show(cardPanel, "Second");

    if (e.getSource() == third)
      ourLayout.show(cardPanel, "Third");
  }
} // end class
```

Mixed Layout

```
import java.awt.*;
import java.applet.Applet;
public class Mixed extends Applet
{
  /* This example demonstrates nesting of containers inside other containers.
  Each container can have its own Layout Manager. Thus, we can achieve finer
  degress of control over component placement.  The applet is the main container
  and will use a BorderLayout.  It will use three Panels at North, Center, and
  South.  The Center panel itself will contain 2 other panels inside of it.
  */

  /* Note how all components are declared as class instance variables. Each method
  in this class can have access to them.  Do not redeclare such a component inside
  of init().
  */

  Panel nPanel, sPanel, cPanel, tcPanel, bcPanel;
  Button one, two, three, four, five, six;
  Label bottom, lbl1, lbl2, lbl3;

  public void init()
  {
    nPanel = new Panel();            // north panel will hold three button
```

```
   nPanel.setBackground(Color.blue);  // give it color so you can see it
   one    = new Button("One");
   two    = new Button("Two");
   three  = new Button("Three");
/* Here is a bad idea. If you declared here, inside of init,
  Button three = new Button ("Three"); the class instance Button three would be
  "lost". It would LOOK like you had a button three on the applet, but it would
  not generate any action events.
*/
   // setLayout for North Panel and add the buttons
   nPanel.setLayout (new FlowLayout(FlowLayout.CENTER));
   nPanel.add(one);
   nPanel.add(two);
   nPanel.add(three);

   sPanel = new Panel();       // south Panel will just hold one Label
   sPanel.setBackground(Color.yellow);  // give it color so you can see it
   bottom = new Label("This is South");
   //set Layout and add Label
   sPanel.setLayout (new FlowLayout(FlowLayout.CENTER));
   sPanel.add (bottom);

   cPanel  = new Panel();     // center panel holds two other panels
   tcPanel = new Panel();     // top panel on center panel holds three labels
   tcPanel.setBackground(Color.gray);  // give it color so you can see it
   bcPanel = new Panel();     // bottom panel on center panel hold three buttons
   bcPanel.setBackground(Color.green);  // give it color so you can see it

   lbl1 = new Label("One");   // the labels
   lbl2 = new Label("Two");
   lbl3 = new Label("Three");

   four = new Button("Four"); // the buttons
   five = new Button("Five");
   six = new Button("Six");

   //set Layout for top center Panel and add labels
   tcPanel.setLayout (new GridLayout(1, 3, 5, 5));  // 1 row, 3 columns, 5 pixel gap
   tcPanel.add(lbl1);
   tcPanel.add(lbl2);
   tcPanel.add(lbl3);

   //set Layout for bottom center panel and add buttons
   bcPanel.setLayout (new GridLayout(3, 1, 5, 5));  // 3 rows, 1 col, 5 pixel gap
   bcPanel.add(four);
   bcPanel.add(five);
```

```
    bcPanel.add(six);

    //add two center panels (top and bottom) to the center panel
    cPanel.setLayout(new GridLayout(2, 1));  // 2 rows, 1 col, no gaps
    cPanel.add(tcPanel);
    cPanel.add(bcPanel);

    //set Layout for the Applet itself and add the panels
    this.setLayout (new BorderLayout());
    add(nPanel, BorderLayout.NORTH);
    add(sPanel, BorderLayout.SOUTH);
    add(cPanel, BorderLayout.CENTER);
  }
}
```

VIVA QUESTINS BASED ON JAVA PROGRAMMING

Q1. **What is the difference between an Interface and an Abstract class?**

Ans An abstract class can have instance methods that implement a default behavior. An Interface can only declare constants and instance methods, but cannot implement default behavior and all methods are implicitly abstract. An interface has all public members and no implementation. An abstract class is a class which may have the usual flavors of class members (private, protected, etc.), but has some abstract methods.

Q2. **What is the purpose of garbage collection in Java, and when is it used?**

Ans The purpose of garbage collection is to identify and discard objects that are no longer needed by a program so that their resources can be reclaimed and reused. A Java object is subject to garbage collection when it becomes unreachable to the program in which it is used.

Q3. **Describe synchronization in respect to multithreading.**

Ans With respect to multithreading, synchronization is the capability to control the access of multiple threads to shared resources. Without synchonization, it is possible for one thread to modify a shared variable while another thread is in the process of using or updating same shared variable. This usually leads to significant errors.

Q4. **Explain different way of using thread?**

Ans The thread could be implemented by using runnable interface or by inheriting from the Thread class. The former is more advantageous, 'cause when you are going for multiple inheritance..the only interface can help.

Q5. **What are pass by reference and passby value?**

Ans Pass By Reference means the passing the address itself rather than passing the value. Passby Value means passing a copy of the value to be passed.

Q6. **What is HashMap and Map?**

Ans Map is Interface and Hashmap is class that implements that.

Q7. **Difference between HashMap and HashTable?**

Ans The HashMap class is roughly equivalent to Hashtable, except that it is unsynchronized and permits nulls. (HashMap allows null values as key and value whereas Hashtable doesnt allow). HashMap does not guarantee that the order of the map will remain constant over time. HashMap is unsynchronized and Hashtable is synchronized.

Q8. **Difference between Vector and ArrayList?**

Ans Vector is synchronized whereas arraylist is not.

Q9. **Difference between Swing and Awt?**

Ans AWT are heavy-weight componenets. Swings are light-weight components. Hence swing works faster than AWT.

Q10. **What is an Iterator?**

Ans A constructor is a member function of a class that is used to create objects of that class. It has the same name as the class itself, has no return type, and is invoked using the newoperator. A method is an ordinary member function of a class. It has its own name, a return type (which may be void), and is invoked using the dot operator.

Section 2

HTML

HTML stands for **H**yper **T**ext **M**arkup **L**anguage

An HTML file is a text file containing small **markup tags**

The markup tags tell the Web browser **how to display** the page

An HTML file must have an **htm** or **html** file extension

An HTML file can be created using a **simple text editor**

Basic HTML Tags

Tag	**Description**
<A ...> Anchor	HREF: URL you are linking to
	NAME: name a section of the page
	TARGET = "_blank" | "_parent" | "_self" | "_top" | *window name* which window the document should go in
	TITLE: suggested title for the document to be opened
	onClick: script to run when the user clicks on this anchor
	onMouseOver: when the mouse is over the link
	onMouseOut: when the mouse is no longer over the link
	ACCESSKEY
<BODY ...>	BGCOLOR: background color of the page
	BACKGROUND: background picture for the page
	TEXT: color of the text on the page
	LINK: color of links that haven't been followed yet
	VLINK: color of links that have been followed
	ALINK: color of links while you are clicking on them
	BGPROPERTIES = FIXED if the background image should not scroll
	TOPMARGIN: size of top and bottom margins
	LEFTMARGIN: size of left and right margins

	MARGINHEIGHT: size of top and bottom margins
	MARGINWIDTH: size of left and right margins
	onLoad: Script to run once the page is fully loaded
	onUnload
	onFocus
	onBlur
	STYLESRC: MS FrontPage extension
	SCROLL = YES | NO If the document should have a scroll bar
<BR ...> Line Break	CLEAR = LEFT | RIGHT | ALL | BOTH go past a picture or other object
<CAPTION ...>	ALIGN = TOP | BOTTOM | LEFT | RIGHT alignment of caption to table
	VALIGN = TOP | BOTTOM if caption should be above or below table
<CENTER ...>	
<CITE> Citation	
<CODE>	
<FONT ...>	SIZE: size of the font
	COLOR: color of the text
	FACE: set the typestyle for text
	POINT-SIZE
	WEIGHT
<FORM ...>	ACTION: URL of the CGI program
	METHOD = GET | POST how to transfer the data to the CGI
	NAME: name of this form
	ENCTYPE = "multipart/form-data" | "application/x-www-form-urlencoded" | "text/plain" what type of form this is
	TARGET = "_blank" | "_parent" | "_self" | "_top" | *frame name* what frames to put the results in
	onSubmit: script to run before the form is submitted

	onReset: script to run before the form is reset
<HR ...> Horizontal Rule	NOSHADE: don't use shadow effect
	SIZE: height
	WIDTH: horizontal width of the line
	ALIGN = LEFT | RIGHT | CENTER horizontal alignment of the line
	COLOR: color of the line
<HTML>	
<MARQUEE ...>	WIDTH: how wide the marquee is
	HEIGHT: how tall the marquee is
	DIRECTION = LEFT | RIGHT which direction the marquee should scroll
	BEHAVIOR = SCROLL | SLIDE | ALTERNATE what type of scrolling
	SCROLLDELAY: how long to delay between each jump
	SCROLLAMOUNT: how far to jump
	LOOP = INFINITE | *number of loops* how many times to loop
	BGCOLOR: background color
	HSPACE: horizontal space around the marquee
	VSPACE: vertical space around the marquee
<TABLE ...>	BORDER: size of border around the table
	CELLPADDING: space between the edge of a cell and the contents
	CELLSPACING: space between cells
	WIDTH: width of the table as a whole
	BGCOLOR: color of the background
	BACKGROUND: picture to use as background
	ALIGN = LEFT | RIGHT alignment of table to surrounding text
	HSPACE: horizontal space between table and surrounding text
	VSPACE: vertical space between table and surrounding text
	HEIGHT: height of the table as a whole

	FRAME = VOID | BOX | BORDER | ABOVE | BELOW | LHS | RHS | HSIDES | VSIDES parts of outside border that are visible
	RULES = NONE | ALL | COLS | ROWS | GROUPS if there should be internal borders
	BORDERCOLOR: color of border around the table
	BORDERCOLORLIGHT: color of "light" part of border around the table
	BORDERCOLORDARK: color of "dark" part of border around the table
	SUMMARY: Summary of the purpose of the table
<TBODY ...> Table Body Section	
<TD ...> Table Data	ALIGN = LEFT | CENTER | MIDDLE | RIGHT horizontal alignment of cell contents
	VALIGN = TOP | MIDDLE | CENTER | BOTTOM | BASELINE vertical alignment of cell contents
	WIDTH: width of cell
	HEIGHT: height of cell
	COLSPAN: number of columns to cover
	ROWSPAN: number of rows to cover
	NOWRAP: don't word wrap
	BGCOLOR: color of the background
	BORDERCOLOR: color of border around the table
	BORDERCOLORDARK: color of "dark" part of border around the table
	BORDERCOLORLIGHT: color of "light" part of border around the table
	BACKGROUND: picture to use as background
<TR ...> Table Row	ALIGN = LEFT | CENTER | RIGHT horizontal alignment of cell contents
	HALIGN = LEFT | CENTER | RIGHT
	VALIGN = TOP | MIDDLE | BOTTOM | BASELINE vertical alignment of cell contents

	BGCOLOR: background color
	BACKGROUND: background image
	BORDERCOLOR: color of border around each cell
	BORDERCOLORLIGHT: color of "light" part of border around each cell
	BORDERCOLORDARK: color of "dark" part of border around each cell

Sample Program

Table cells that span more than one row/column
(This sample program demonstrates how to define table cells that span more than one row or one column.)

Steps to write HTML Program :

1. Create HTML file by using vi editor.

2. Define Tags HTML, body tags.

3. Inside body tag ,define the table tags.

4. Save the file with .html extension

5.Open the file in the browser. Browser will show the output as shown in the sample program output**.**

Program Source Code :

```
<html>
```

```
<body>

<h4>Cell that spans two columns:</h4>

<table border="1">

<tr>

  <th>Name</th>

  <th colspan="2">Telephone</th>

</tr>

<tr>

  <td>Bill Gates</td>

  <td>555 77 854</td>

  <td>555 77 855</td>

</tr>

</table>

<h4>Cell that spans two rows:</h4>

<table border="1">

<tr>

  <th>First Name:</th>

  <td>Bill Gates</td>

</tr>

<tr>

  <th rowspan="2">Telephone:</th>

  <td>555 77 854</td>

</tr>
```

```
<tr>
  <td>555 77 855</td>
</tr>
</table>
</body>
</html>
```

Output :

Cell that spans two columns:

<table>
<tr><th>Name</th><th colspan="2">Telephone</th></tr>
<tr><td>Bill Gates</td><td>555 77 854</td><td>555 77 855</td></tr>
</table>

Cell that spans two rows:

<table>
<tr><th>First Name:</th><td>Bill Gates</td></tr>
<tr><th rowspan="2">Telephone:</th><td>555 77 854</td></tr>
<tr><td>555 77 855</td></tr>
</table>

DHTML

DHTML stands for **D**ynamic **HTML**.

DHTML is not a standard defined by the World Wide Web Consortium (W3C). DHTML is a "marketing term" - used by Netscape and Microsoft to describe the new technologies the 4.x generation browsers would support.

DHTML is a combination of technologies used to create dynamic Web sites.

DHTML Technologies

With DHTML a Web developer can control how to display and position HTML elements in a browser window.

HTML 4.0

With HTML 4.0 all formatting can be moved out of the HTML document and into a separate **style sheet**. Because HTML 4.0 separates the presentation of the document from its structure, we have total control of presentation layout without messing up the document content.

Cascading Style Sheets (CSS)

With CSS we have a style and layout model for HTML documents.

CSS was a breakthrough in Web design because it allowed developers to control the style and layout of multiple Web pages all at once. As a Web developer you can define a style for each HTML element and apply it to as many Web pages as you want. To make a global change, simply change the style, and all elements in the Web are updated automatically.

The Document Object Model (DOM)

DOM stands for the **D**ocument **O**bject **M**odel.

The HTML DOM is the Document Object Model for HTML.

The HTML DOM defines a standard set of objects for HTML, and a standard way to access and manipulate HTML objects.

"The W3C Document Object Model (DOM) is a platform and language neutral interface that allows programs and scripts to dynamically access and update the content, structure, and style of a document".

JavaScript allows you to write code to control all HTML elements.

DHTML Technologies in Netscape 4.x and Internet Explorer 4.x

Netscape 4.x	Cross-Browser DHTML	Internet Explorer 4.x

JSS (JavaScript Style Sheets) (allows you to control how different HTML elements will be displayed) Layers (allows you to control element positioning and visibility)	CSS1 CSS2 (allows you to control how different HTML elements will be displayed) CSS Positioning (allows you to control element positioning and visibility) JavaScript	Visual Filters (allow you to apply visual effects to text and graphics) Dynamic CSS (allows you to control element positioning and visibility

Sample Program

How to make an element invisible. Do you want the element to show or not?

Steps to write DHTML Program :

1. Create HTML file by using vi editor.

2. Define Tags HTML,body tags.

3. Inside body tag ,define the table tags.

4. Save the file with .html extension

5.Open the file in the browser.Browser will show the output as shown in the sample program output**.**

Program Source Code :

```
<html>

<head>
```

```
<style>

h1.one

{

visibility:visible;

}

h1.two

{

visibility:hidden;

}

</style>


</head>

<body>

<h1 class="one">Heading one</h1>

<h1 class="two">Heading two</h1>

<p>Where is heading two?</p>


</body>

</html>
```

Output :

Heading one

Heading two

Where is heading two?

Questions :

What is CSS?

CSS stands for Cascading Style Sheets and is a simple styling language which allows attaching style to HTML elements. Every element type as well as every occurance of a specific element within that type can be declared an unique style, e.g. margins, positioning, color or size.

What are Style Sheets?

Style Sheets are templates, very similar to templates in desktop publishing applications, containing a collection of rules declared to various selectors (elements).

What is external Style Sheet? How to link?

External Style Sheet is a template/document/file containing style information which can be linked with any number of HTML documents. This is a very convenient way of formatting the entire site as well as restyling it by editing just one file.

The file is linked with HTML documents via the LINK element inside the HEAD element. Files containing style information must have extension .css, e.g. style.css.

```
<HEAD>
<LINK REL=STYLESHEET HREF="style.css" TYPE="text/css">
</HEAD>
```

What is embedded style? How to link?

Embedded style is the style attached to one specific document. The style information is specified as a content of the STYLE element inside the HEAD element and will apply to the entire document.

```
<HEAD>
<STYLE TYPE="text/css">
<!--
P {text-indent: 10pt}
-->
```

```
</STYLE>
</HEAD>
```

Note: The styling rules are written as a HTML comment, that is, between <!-- and --> to hide the content in browsers without CSS support which would otherwise be displayed.

What is inline style? How to link?

Inline style is the style attached to one specific element. The style is specified directly in the start tag as a value of the STYLE attribute and will apply exclusively to this specific element occurance.

```
<P STYLE="text-indent: 10pt">Indented paragraph</P>
```

What is imported Style Sheet? How to link?

Imported Style Sheet is a sheet that can be imported to (combined with) another sheet. This allows creating one main sheet containing declarations that apply to the whole site and partial sheets containing declarations that apply to specific elements (or documents) that may require additional styling. By importing partial sheets to the main sheet a number of sources can be combined into one.

To import a style sheet or style sheets include the @import notation or notations in the STYLE element. The @import notations must come before any other declaration. If more than one sheet is imported they will cascade in order they are imported - the last imported sheet will override the next last; the next last will override the second last, and so on. If the imported style is in conflict with the rules declared in the main sheet then it will be overridden.

```
<LINK REL=STYLESHEET HREF="main.css" TYPE="text/css">

<STYLE TYPE="text=css">

<!--

@import url(http://www.and.so.on.partial1.css);

@import url(http://www.and.so.on.partial2.css);

 .... other statements

-->

</STYLE>
```

What is alternate Style Sheet? How to link?

Alternate Style Sheet is a sheet defining an *alternate* style to be used in place of style(s) declared as *persistent and/or preferred* .

Persistent style is a default style that applies when style sheets are enabled but can disabled in favor of an alternate style, e.g.:

<LINK REL=Stylesheet HREF="style.css" TYPE="text/css">

Preferred style is a default style that applies automatically and is declared by setting the TITLE attribute to the LINK element. There can only be one preferred style, e.g.:

<LINK REL=Stylesheet HREF="style2.css" TYPE="text/css" TITLE="appropriate style description">

Alternate style gives an user the choice of selecting an alternative style - a very convenient way of specifying a media dependent style. Note: Each group of alternate styles must have unique TITLE, e.g.:

<LINK REL="Alternate Stylesheet" HREF="style3.css" TYPE="text/css" TITLE="appropriate style description" MEDIA=screen>
<LINK REL="Alternate Stylesheet" HREF="style4.css" TYPE="text/css" TITLE="appropriate style description" MEDIA=print>

Alternate stylesheets are not yet supported.

JavaScript

JavaScript was designed to add interactivity to HTML pages

JavaScript is a scripting language (a scripting language is a lightweight programming language)

A JavaScript consists of lines of executable computer code

A JavaScript is usually embedded directly into HTML pages

JavaScript is an interpreted language (means that scripts execute without preliminary compilation)

Everyone can use JavaScript without purchasing a license

Are Java and JavaScript the Same?

NO!

Java and JavaScript are two completely different languages in both concept and design!

Java (developed by Sun Microsystems) is a powerful and much more complex programming language - in the same category as C and C++.

What can a JavaScript Do?

JavaScript gives HTML designers a programming tool - HTML authors are normally not programmers, but JavaScript is a scripting language with a very simple syntax! Almost anyone can put small "snippets" of code into their HTML pages

JavaScript can put dynamic text into an HTML page - A JavaScript statement like this: document.write("<h1>" + name + "</h1>") can write a variable text into an HTML page

JavaScript can react to events - A JavaScript can be set to execute when something happens, like when a page has finished loading or when a user clicks on an HTML element

JavaScript can read and write HTML elements - A JavaScript can read and change the content of an HTML element

JavaScript can be used to validate data - A JavaScript can be used to validate form data before it is submitted to a server. This saves the server from extra processing

JavaScript can be used to detect the visitor's browser - A JavaScript can be used to detect the visitor's browser, and - depending on the browser - load another page specifically designed for that browser

JavaScript can be used to create cookies - A JavaScript can be used to store and retrieve information on the visitor's computer

The Advantages and Disadvantages of JavaScript

We've already seen some of the advantages of JavaScript, like cross-browser support, validating data on the client, and being able to create more sophisticated user interfaces.

JavaScript effects are also much faster to download than some other front-end technologies like Flash and Java applets. In fact, unless you're writing a massive JavaScript application, it's quite likely that no significant extra download time will be added to a page by using JavaScript on it. Nor do users need to download a plugin before they can view your JavaScript, as they would with Flash for example, they simply need a browser that supports it – and, of course, most modern browsers do.

Other advantages include the fact that you don't need any extra tools to write JavaScript, any plain text or HTML editor will do, so there's no expensive development software to buy. It's also an easy language to learn, and there's a thriving and supportive online community of JavaScript developers and information resources.

The disadvantages of JavaScript, as with most web development, are almost entirely related to browser compatibility.

While the advances in browser programmability we've seen over recent years are, generally speaking, a good thing, if you don't implement them with care you can create a lot of inconsistencies and broken pages quite unintentionally using JavaScript. Code that works just great on IE4 might not work at all on Netscape 4, what works in NN6 doesn't always work in NN 4, and so on.

In essence, there are two main problems with JavaScript and browsers:

· The different JavaScript versions in different browsers.

· Browser programmability: the HTML elements and features of the browser that can be accessed through any scripting language. (IE4 , for example, makes most of the page and HTML accessible to scripts, but Navigator 4 limits what can be accessed and manipulated.)

The HTML <script> tag is used to insert a JavaScript into an HTML page.

How to Put a JavaScript Into an HTML Page

```
<html>

<body>

<script type="text/javascript">

document.write("Hello World!")

</script>
```

```
</body>

</html>
```

The code above will produce this output on an HTML page:

```
Hello World!
```

Example Explained

To insert a JavaScript into an HTML page, we use the <script> tag (also use the type attribute to define the scripting language).

So, the <script type="text/javascript"> and </script> tells where the JavaScript starts and ends:

```
<html>

<body>

<script type="text/javascript">

...

</script>

</body>

</html>
```

The word **document.write** is a standard JavaScript command for writing output to a page.

By entering the document.write command between the <script type="text/javascript"> and </script> tags, the browser will recognize it as a JavaScript command and execute the code line. In this case the browser will write Hello World! to the page:

```
<html>

<body>

<script type="text/javascript">

document.write("Hello World!")
```

```
</script>

</body>

</html>
```

Note: If we had not entered the <script> tag, the browser would have treated the document.write("Hello World!") command as pure text, and just write the entire line on the page.

Ending Statements With a Semicolon?

With traditional programming languages, like C++ and Java, each code statement has to end with a semicolon.

Many programmers continue this habit when writing JavaScript, but in general, semicolons are **optional**! However, semicolons are required if you want to put more than one statement on a single line.

Sample Program

Function with arguments, that returns a value

1. Create HTML file by using vi editor.

2. Define Tags HTML,body tags.

3. Inside body tag ,define the table tags.

4. Save the file with .html extension

5.Open the file in the browser.Browser will show the output as shown in the sample program output**.**

Program Source Code :

```
<html>

<head>

<script type="text/javascript">

function product(a,b)
```

```
{
return a*b
}
</script>
</head>

<body>
<script type="text/javascript">
document.write(product(4,3))
</script>
<p>The script in the body section calls a function with two parameters (4 and 3).</p>
<p>The function will return the product of these two parameters.</p>
</body>
</html>
```

Output :

12

The script in the body section calls a function with two parameters (4 and 3).

The function will return the product of these two parameters.

Questions :

Q1.How do you get the page background image to stay fixed when the page is scrolled?

Ans.The technique is called *watermarking.*

One simple way is to add bgproperties="fixed" to the body tag, like this:

```
<body bgproperties="fixed">
```

Note that this typically only works in Internet Explorer browsers.

Another way of doing it that also works in later Netscape browsers (6.x & up) is to add this style script to the <head> of your page:

```
<style>
body {background-attachment:fixed}
</style>
```

Q2.How do you call more than one JavaScript function in a body tag (or other) event handler?

Ans.Simple. End each function call with a semi-colon ;
Like this:

```
<body onload="someFunction();otherFunction();">
```

or, say, in a mouseover...

```
onMouseOver="someFunction();otherFunction();"
```

In JavaScript, the semi-colon is essentially an end-of-line marker. Within reasonable limits, you can actually write a whole script inside of an event handler.

The same thing applies to the href="javascript:etc" structure. For instance:

```
<a  href="javascript:someFunction();otherFunction();">
Click Here
</a>
```

Q3. How do you make a window "pop under" when it is opened?

Put this as early in the <head> of the page as possible:

```
<script>
self.blur();
</script>
```

That tells the window to "lose focus" as soon as it reads the self.blur(); -- which makes the window "jump behind" the window that is currently in focus.

Q4. How can you set a window's size when it is opened?

Put this as early in the <head> of the page as possible:

```
<script>
self.resizeTo(100,200);
</script>
```

Set the dimensions in the parentheses. The first number is the width; the second is the height.

Q5.How can you make certain a window will "come to the front" when it is loaded?

Add onload="self.focus();" to the body tag, like this:

```
<body onload="self.focus();">
```

As soon as the window is fully loaded, it will "take focus" and move in front of any other open windows.

<u>Some additional Viva Questions</u>

Viva Voce:

1. What is the most important feature of Java?

Java is a platform independent language.

2. What do you mean by platform independence?

Platform independence means that we can write and compile the java code in one platform (eg Windows) and can execute the class in any other supported platform eg (Linux,Solaris,etc).

3. What is a JVM?

JVM is Java Virtual Machine which is a run time environment for the compiled java class files.

4. Are JVM's platform independent?

JVM's are not platform independent. JVM's are platform specific run time implementation provided by the vendor.

5. What is the difference between a JDK and a JVM?

JDK is Java Development Kit which is for development purpose and it includes execution environment also. But JVM is purely a run time environment and hence you will not be able to compile your source files using a JVM.

6. What is a pointer and does Java support pointers?

Pointer is a reference handle to a memory location. Improper handling of pointers leads to memory leaks and reliability issues hence Java doesn't support the usage of pointers.

7. What is the base class of all classes?

java.lang.Object

8. Does Java support multiple inheritance?

Java doesn't support multiple inheritance.

9. Is Java a pure object oriented language?

Java uses primitive data types and hence is not a pure object oriented language.

10. Are arrays primitive data types?

In Java, Arrays are objects.

11. What is difference between Path and Classpath?

Path and Classpath are operating system level environment variales. Path is used define where the system can find the executables(.exe) files and classpath is used to specify the location .class files.

12. What are local variables?

Local varaiables are those which are declared within a block of code like methods. Local variables should be initialised before accessing them

25. Can a class declared as private be accessed outside it's package?

Not possible.

26. Can a class be declared as protected?

A class can't be declared as protected. only methods can be declared as protected.

27. What is the access scope of a protected method?

A protected method can be accessed by the classes within the same package or by the subclasses of the class in any package.

28. What is the purpose of declaring a variable as final?

A final variable's value can't be changed. final variables should be initialized before using them.

29. What is the impact of declaring a method as final?

A method declared as final can't be overridden. A sub-class can't have the same method signature with a different implementation.

30. I don't want my class to be inherited by any other class. What should i do?

You should declared your class as final. But you can't define your class as final, if it is an abstract class. A class declared as final can't be extended by any other class.

31. Can you give few examples of final classes defined in Java API?

java.lang.String, java.lang.Math are final classes.

32. How is final different from finally and finalize()?

final is a modifier which can be applied to a class or a method or a variable. final class can't be inherited, final method can't be overridden and final variable can't be changed.

finally is an exception handling code section which gets executed whether an exception is raised or not by the try block code segment.

finalize() is a method of Object class which will be executed by the JVM just before garbage collecting object to give a final chance for resource releasing activity.

33. Can a class be declared as static?

We can not declare top level class as static, but only inner class can be declared static.

```
public class Test
{
   static class InnerClass
   {
      public static void InnerMethod()
      { System.out.println("Static Inner Class!"); }
   }
   public static void main(String args[])
```

```
    {
      Test.InnerClass.InnerMethod();
    }
}
//output: Static Inner Class!
```

34. When will you define a method as static?

When a method needs to be accessed even before the creation of the object of the class then we should declare the method as static.

35. What are the restriction imposed on a static method or a static block of code?

A static method should not refer to instance variables without creating an instance and cannot use "this" operator to refer the instance.

36. I want to print "Hello" even before main() is executed. How will you acheive that?

Print the statement inside a static block of code. Static blocks get executed when the class gets loaded into the memory and even before the creation of an object. Hence it will be executed before the

main() method. And it will be executed only once.

37.How-will-you-communicate-between-two-Applets

The simplest method is to use the static variables of a shared class since there's only one instance of the class and hence only one copy of its static variables. A slightly more reliable method relies on the fact that all the applets on a given page share the same AppletContext. We obtain this applet context as follows:

AppletContext ac = getAppletContext();

AppletContext provides applets with methods such as getApplet(name), getApplets(),getAudioClip, getImage, showDocument and showStatus().

38. Which classes can an applet extend?

An applet can extend the java.applet.Applet class or the java.swing.JApplet class. The java.applet.Applet class extends the java.awt.Panel class and enables you to use the GUI tools in the AWT package. The java.swing.JApplet class is a subclass of java.applet.Applet that also enables you to use the Swing GUI tools.

38. How do you cause the applet GUI in the browser to be refreshed when data in it may have changed?

You invoke the repaint() method. This method causes the paint method, which is required in any applet, to be invoked again, updating the display.

39.For what do you use the start() method?

You use the start() method when the applet must perform a task after it is initialized, before receiving user input. The start() method either performs the applet's work or (more likely) starts up one or more threads to perform the work.

40. True or false: An applet can make network connections to any host on the Internet.

False: An applet can only connect to the host that it came from.

41. How do you get the value of a parameter specified in the APPLET tag from within the applet's code?

You use the getParameter("*Parameter name*") method, which returns the String value of the parameter.

42. True of false: An applet can run multiple threads.

True. The paint and update methods are always called from the AWT drawing and event handling thread. You can have your applet create additional threads, which is recommended for performing time-consuming tasks.