# LAB MANUAL

| | | |
|---|---|---|
| **Lab Name** | **:** | JAVA PROGRAMMING LAB |
| **Lab Code** | **:** | 4CS4-25 |
| **Branch** | **:** | Computer Science and Engineering |
| **Year** | **:** | **2nd** Year [IV SEM] |


**Jaipur Engineering College and Research Center, Jaipur**

Department of Computer Science& Engineering

(Rajasthan Technical University, KOTA)

# INDEX

| | | |
|---|---|---|
| | 5. Write a java program to develop a simple calculator using scanner class | |
| Assignment 2 | 1. Write a java program to sort elements of one dimensional array<br>2. Write a java program to perform Linear Search of one dimensional array<br>3. Write a java program to perform Matrix Multiplication and Transpose of a Matrix | |
| Assignment 3 | 1. Write a java program to implement class and object concept<br>2. Write a java program to implement class and object concept using Scanner class<br>3. Write a java program to implement Constructor concept | |
| Assignment 4 | 1. Write a java program to implement operators concept<br>2. Write a java program to implement decision making and control statements using break and continue keywords<br>3. Write a java program to implement iteration statements in java using break and continue keywords<br>4. Write a java program to implement single or multilevel inheritance<br>5. Write a java program to implement hierarchical or hybrid inheritance | |
| Assignment 5 | 1. Write a java program to implement data abstraction (2 programs)<br>2. Write a java program to implement interface (2 programs)<br>3. Write a java program to implement polymorphism(method overloading)<br>4. Write a java program to implement polymorphism(method overriding) | |

| | | |
|---|---|---|
| Assignment 6 | 1. Case study of Strings<br>2. Case study of Multithreading<br>3. Case study of IOStreams | |
| Assignment 7 | 1. Case study of Exception handling<br>Note: it should cover understanding of Exception handling fundamentals, Exception types, uncaught exceptions, try, catch and multiple catch statements. Usage of throw, throws and finally.<br>2. Case study of Applet | |

JAVA LAB MANUAL

SUBJECT CODE: 4CS4-25 SUBJECT NAME : JAVA LAB

1. Vision

To become renowned Centre of excellence in computer science and engineering and make competent engineers & professionals with high ethical values prepared for lifelong learning.

Mission

**M1:** To impart outcome based education for emerging technologies in the field of computer science and engineering.
**M2:** To provide opportunities for interaction between academia and industry.
**M3:** To provide platform for lifelong learning by accepting the change in technologies
**M4:** To develop aptitude of fulfilling social responsibilities.

**2. PEO**

**1.**To provide students with the fundamentals of Engineering Sciences with more emphasis in Computer Science &Engineering by way of analyzing and exploiting engineering challenges.
**2.**To train students with good scientific and engineering knowledge so as to comprehend, analyze, design, and create novel products and solutions for the real life problems.
**3.**To inculcate professional and ethical attitude, effective communication skills, teamwork skills, multidisciplinary approach, entrepreneurial thinking and an ability to relate engineering issues with social issues.

**4.**To provide students with an academic environment aware of excellence, leadership, written ethical codes and guidelines, and the self motivated life-long learning needed for a successful professional career.
**5**. To prepare students to excel in Industry and Higher education by Educating Students along with High moral values and Knowledge

**3.PO**

- **Engineering Knowledge:** Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems in IT.
- **Problem analysis:** Identify, formulate, research literature, and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences in IT.
- **Design/development of solutions:** Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, societal, and environmental considerations using IT.
- **Conduct investigations of complex problems:** Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions using IT.
- **Modern tool usage:** Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modeling to complex engineering activities with an understanding of the limitations in IT.
- **The engineer and society:** Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional engineering practice using IT.
- **Environment and sustainability:**Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development in IT.
- **Ethics:** Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice using IT.
- **Individual and team work:** Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings in IT.
- **Communication:** Communicate effectively on complex engineering activities with the engineering community and with society at large, such as, being able to comprehend and

write effective reports and design documentation, make effective presentations, and give and receive clear instructions.

- **Project Management and finance:** Demonstrate knowledge and understanding of the engineering and management principles and apply these to one's own work, as a member and leader in a team, to manage IT projects and in multidisciplinary environments.
- **Life –long Learning:** Recognize the need for, and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological changes needed in IT.

## 4. CO

- Develop an in depth understanding of programming & apply by writing Object Oriented programs in Java
- Understand & Develop packages, Interfaces, Strings and exception handling in Java
- Create applications involving file handling, concurrency and applet

## 5. MAPPING OF CO & PO

| Semester | Subject | Code | L/T/P | CO | PO1 | PO2 | PO3 | PO4 | PO5 | PO6 | PO7 | PO8 | PO9 | PO10 | PO11 | PO12 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| IV | JAVA PROGRAMMING LAB | 4CS4-25 | P | 1. Develop an in depth understanding of programming & apply by writing Object Oriented programs in Java | H | H | H | H | M | L | - | - | M | L | L | M |
| | | | P | 2. Understand & Develop packages, Interfaces, Strings and exception handling in Java | H | H | H | H | M | L | - | - | M | L | L | M |
| | | | P | 3. Create applications involving file handling, concurrency and applet | H | H | H | H | M | L | - | - | M | L | L | M |

## 6. SYLLABUS

4CS4-25 Java Programming Lab

| **Class: IV Sem. B.Tech.** | **Evaluation** |
|---|---|
| **Branch: Computer Engineering**<br>**Schedule per Week Practical Hrs.: 2** | **Examination Time = Three Hours**<br>**Maximum Marks = 50**<br>**[IA/ETE] : (30)/(20)** |

**Objectives:** At the end of the semester, the students should have clearly understood and implemented the following:

List of Experiment:

1. Develop an in depth understanding of programming in Java: data types, variables, operators, operator precedence, Decision and control statements, arrays, switch statement, Iteration Statements, Jump Statements, Using break, Using continue, return.
2. Write Object Oriented programs in Java: Objects, Classes constructors, returning and passing objects as parameter, Inheritance, Access Control, Using super, final with inheritance Overloading and overriding methods, Abstract classes, Extended classes.
3. Develop understanding to developing packages & Interfaces in Java: Package, concept of CLASSPATH, access modifiers, importing package, Defining and implementing interfaces.
4. Develop understanding to developing Strings and exception handling: String constructors, special string operations, character extraction, searching and comparing strings, string Buffer class. Exception handling fundamentals, Exception types, uncaught exceptions, try, catch and multiple catch statements. Usage of throw, throws and finally.
5. Develop applications involving file handling: I/O streams, File I/O.
6. Develop applications involving concurrency: Processes and Threads, Thread Objects, Defining and Starting a Thread, Pausing Execution with Sleep, Interrupts, Joins, and Synchronization. Indicative List of exercises:
7. Programs to demonstrate basic concepts e.g. operators, classes, constructors, control & iteration statements, recursion etc. such as complex arithmetic, matrix arithmetic, tower of Hanoi problem etc.
8. Development of programs/projects to demonstrate concepts like inheritance, exception handling, packages, interfaces etc. such as application for electricity department, library management, ticket reservation system, payroll system etc.
9. Development of a project to demonstrate various file handling concepts.
10. Develop applications involving Applet: Applet Fundamentals, using paint method and drawing polygons. It is expected that each laboratory assignments to given to the students with an aim to In order to achieve the above objectives.

7. BOOKS:-

- Text books:-

- Head First Java, 2nd Edition 2nd Editionby Kathy Sierra
- Core Java Volume I--Fundamentals (9th Edition) (Core Series) 9th Editionby Cay S. Horstmann

Reference Books:-

1. Java: A Beginner's Guide, Sixth Edition 6th Editionby Herbert Schild

**8. INSTRUCTIONAL METHODS:-**

**8.1. Direct Instructions:**

- Black board presentation

**8.2. Interactive Instruction:**

- coding

**8.3. Indirect Instructions:**

- Problem solving

**9. LEARNING MATERIALS:-**

9.1. Text/Lab Manual

**10. ASSESSMENT OF OUTCOMES:-**

- End term Practical exam (Conducted by RTU, KOTA)
- Daily Lab interaction.

**OUTCOMES WILL BE ACHIEVED THROUGH FOLLOWING:-**

1. Lab Teaching (through chalk and board).
2. Discussion on website work

## INSTRUCTIONS OF LAB

### DO's

- Please switch off the Mobile/Cell phone before entering Lab.
- Enter the Lab with complete source code and data.
- Check whether all peripheral are available at your desktop before proceeding for program.
- Intimate the lab In charge whenever you are incompatible in using the system or in case software get corrupted/ infected by virus.
- Arrange all the peripheral and seats before leaving the lab.
- Properly shutdown the system before leaving the lab.
- Keep the bag outside in the racks.
- Enter the lab on time and leave at proper time.
- Maintain the decorum of the lab.
- Utilize lab hours in the corresponding experiment.
- Get your CD / Pen drive checked by Labincharge before using it in the lab.

*DON'TS*

- No one is allowed to bring storage devices like Pan Drive /Floppy etc. in the lab.
- Don't mishandle the system.
- Don't leave the system on standing for long
- Don't bring any external material in the lab.
- Don't make noise in the lab.
- Don't bring the mobile in the lab. If extremely necessary then keep ringers off.
- Don't enter in the lab without permission of lab Incharge.
- Don't litter in the lab.
- Don't delete or make any modification in system files.
- Don't carry any lab equipments outside the lab.

## INSTRUCTIONS FOR STUDENT

**BEFORE ENTERING IN THE LAB**

- All the students are supposed to prepare the theory regarding the next program.
- Students are supposed to bring the practical file and the lab copy.
- Previous programs should be written in the practical file.
- Any student not following these instructions will be denied entry in the lab.

**WHILE WORKING IN THE LAB**

- Adhere to experimental schedule as instructed by the lab incharge.
- Get the previously executed program signed by the instructor.
- Get the output of the current program checked by the instructor in the lab copy.
- Each student should work on his/her assigned computer at each turn of the lab.
- Take responsibility of valuable accessories.
- Concentrate on the assigned practical and do not play games.
- If anyone caught red handed carrying any equipment of the lab, then he will have to face serious consequences.

Assignment 1

1. Case study of Object Oriented Programming

**Object-Oriented Programming** is a methodology or paradigm to design a program using classes and objects. It simplifies software development and maintenance by providing some concepts:

- Object
- Class
- Inheritance
- Polymorphism
- Abstraction
- Encapsulation

Object

Any entity that has state and behavior is known as an object. For example, a chair, pen, table, keyboard, bike, etc. It can be physical or logical. An Object can be defined as an instance of a class. An object contains an address and takes up some space in memory. Objects can communicate without knowing the details of each other's data or code. The only necessary thing is the type of message accepted and the type of response returned by the objects.

**Example:** A dog is an object because it has states like color, name, breed, etc. as well as behaviors like wagging the tail, barking, eating, etc.

Class

Collection of objects is called class. It is a logical entity. A class can also be defined as a blueprint from which you can create an individual object. Class doesn't consume any space.

Inheritance

When one object acquires all the properties and behaviors of a parent object, it is known as inheritance. It provides code reusability. It is used to achieve runtime polymorphism.

Polymorphism

If one task is performed in different ways, it is known as polymorphism. For example: to convince the customer differently, to draw something, for example, shape, triangle, rectangle, etc. In Java, we use method overloading and method overriding to achieve polymorphism.

Another example can be to speak something; for example, a cat speaks meow, dog barks woof, etc.

Abstraction

Hiding internal details and showing functionality is known as abstraction. For example phone call, we don't know the internal processing. In Java, we use abstract class and interface to achieve abstraction.

Encapsulation

Binding (or wrapping) code and data together into a single unit are known as encapsulation. For example, a capsule, it is wrapped with different medicines. A java class is the example of encapsulation. Java bean is the fully encapsulated class because all the data members are private here.

2. Case study of Java

   Ans. Java is the newest in a long line of systems programming languages. This paper looks at what makes it special and backs the findings up with three case studies. The projects exercise Java to the full - its features and APis. The first is a WebComputing Skeleton for remote execution of collaborative programs. The second provides open query mechanisms to a spatial database. The third expands adistributed algorithm visualisation system. Issues of performance are discussed, aswell as alternative ways of approaching the solutions. In general the results are positive and Java comes out as a worthy language for undertaking research in distributed systems.

   Since Java emerged into the computing world's consciousness in February 1996, the growth of interest in its use, and the following it has gained, has been phenomenal. It is possibly true to say that the speed and ease with which Java has gained acceptance across the computing community is unparalleled in the history of programming languages. Some of the credit for Java's rise can be attributed to the present level of Internet and Web activity, which has enabled information about the language to reach a wide audience at an immediacy unknown in the past (Sun,1997). It is the view of this paper, however, that the language can stand on its own merits, and would be a
   success, even without the sometimes denigrated "Java hype".

   If we pause to consider the position of Java in 1996, we can see that in systems languages, the time was certainly ripe for a change. Pascal was beginning to look old-fashioned, and had the severe disadvantage of not being object- oriented. C and C++ had a strong hold (or stranglehold) but educators were uncomfortable with them as teaching languages. Visual

Basic was gaining in popularity and usage but as someone said: "People program in VB not because they like programming in Basic, but because of VB's fancy interface." Several other languages such as Ada, Smalltalk and 4GLs such as Natural had their adherents in industry and academia, but there was still a definite sense of unrest.

Java therefore turned out to be the right language at the right time, just as the other strong languages - Fortran, Cobol, Pascal and C++ - were. Continuing this analogy, one could say that Ada and Small talk were the wrong languages for their time. Ada, for all its excellent points, always felt as if it was behind the trends, and Small talk,

for all its failings, was ahead of the pack in many ways. To gain wide acceptance, a language has to get things just right. In essence it needs to:

• espouse the paradigm of the day,
• add something new and significant,
• be readily available.

Java was able to achieve all three. It is firmly object-oriented (currently the major paradigm); it broke new ground in its use of the Internet and browsers as an execution base; and it was immediately downloadable for free. It is evident that Java has become a language of choice among commercial developers of new systems, and those needing to give old systems a face-lift in the Web age (JavaSoft, 1997). It is also clear that teaching institutions are switching over to Java in large numbers, or considering doing so (Schaller, 1997).

3. Write a java program to print hello message

Ans.

```
class Hello{
public static void main(String args[]){
System.out.println("Hello World");}
}
```

Output - Hello World

4. Write a java program to print any string using scanner class

Ans.

```
import java.util.Scanner;
class name{
public static void main(String args[]){
Scanner input = new Scanner(System.in);
System.out.println("Enter Your Name ");
String str = input.next();
System.out.println("MY name is "+str);}
```

}

OUTPUT - Enter Your Name
Amit
My name is Amit

5. Write a java program to develop a simple calculator using scanner class
   Ans.

```
class calculator{
public static void main(String args[]){
Scanner input = new Scanner(System.in);
System.out.println ("Enter a ");
Scanner a = input.nextInt();
System.out.println ("Enter b ");
Scanner b = input.nextInt();
System.out.println (a+b);
System.out.println (a-b);
System.out.println (a*b);
System.out.println (a/b);
System.out.println (a%b);
}}
```

OUTPUT-

15

5

50

2

0

Assignment 2

1. Write a java program to sort elements of one dimensional array

```
import java.util.Scanner;
public class Ascending _Order
{
   public static void main(String[] args)
   {
      int n, temp;
      Scanner s = new Scanner(System.in);
      System.out.print("Enter no. of elements you want in array:");
      n = s.nextInt();
      int a[] = new int[n];
      System.out.println("Enter all the elements:");
      for (int i = 0; i < n; i++)
{
       a[i] = s.nextInt();
}
for (int i = 0; i < n; i++)
 {
          for (int j = i + 1; j < n; j++)
          {
             if (a[i] > a[j])
             {
                temp = a[i];
                a[i] = a[j];
                a[j] = temp;
             }
          }
  }
      System.out.print("Ascending Order:");
      for (int i = 0; i < n - 1; i++)
      {
         System.out.print(a[i] + ",");
      }
      System.out.print(a[n - 1]);
   }
}
```

Output:

```
javac Ascending _Order.java
java Ascending _Order

Enter no. of elements you want in array:5
Enter all the elements:
4
3
2
6
1
Ascending Order: 1,2,3,4,6
```

2. Write a java program to perform Linear Search of one dimensional array

**Ans**. Java array is an object which contains elements of a similar data type. It is a data structure where we store similar elements. We can store only a fixed set of elements in a Java array. Array in java is index-based, the first element of the array is stored at the 0 index.

Advantages

- **Code Optimization:** It makes the code optimized, we can retrieve or sort the data efficiently.
- **Random access:** We can get any data located at an index position.

Disadvantages

- **Size Limit:** We can store only the fixed size of elements in the array. It doesn't grow its size at runtime. To solve this problem, collection framework is used in Java which grows automatically.

**Code-**

```
// Java code for linearly searching x in arr[]. If x

// is present then return its location, otherwise

// return -1

  class GFG {

public static int search(int arr[], int x) {
```

```
    int n = arr.length;
    for(int i = 0; i < n; i++) {
        if(arr[i] == x)
            return i;
}
    return -1;
}
  public static void main(String args[]) {
    int arr[] = { 2, 3, 4, 10, 40 };
    int x = 10;
    Arrays.sort(arr);
       int result = search(arr, x);
    if (result == -1)
        System.out.print("Element is not present in array");
    else
        System.out.print("Element is present at index " + result);
} }
```

**OUTPUT-** Element is present at index 3

3. Write a java program to perform Matrix Multiplication and Transpose of a matrix

```
import java.util.Scanner;
class Matrix{
    void matrixMul(int m, int n, int p, int q){
            int[][] a,b,c,t;
            a = new int[m][n];
            b = new int[p][q];
            c = new int[m][q];
            t = new int[q][m];
```

```
        Scanner s = new Scanner(System.in);
        System.out.println("Enter the elements of matrix A: ");
        for(int i = 0; i < m; i++){
                for(int j = 0; j < n; j++){
                        a[i][j] = s.nextInt();
                }
        }
        System.out.println("Enter the elements of matrix B: ");
        for(int i = 0; i < p; i++){
                for(int j = 0; j < q; j++){
                        b[i][j] = s.nextInt();
                }
        }
        for(int i = 0; i < m; i++){
                for(int j = 0; j < q; j++){
                        for(int k = 0; k < n; k++){
                                c[i][j] += a[i][k]*b[k][j];
                        }
                }
        }
        System.out.println("Elements of result matrix C are: ");
        for(int i = 0; i < m; i++){
                for(int j = 0; j < q; j++){
                        System.out.print(c[i][j]+"\t");
                }
                System.out.print("\n");
        }
        for(int i = 0; i < q; i++){
                for(int j = 0; j < m; j++){
                        t[i][j] = c[j][i];
                }
        }
        System.out.println("Elements of transpose matrix T are: ");
        for(int i = 0; i < q; i++){
                for(int j = 0; j < m; j++){
                        System.out.print(t[i][j]+"\t");
                }
                System.out.print("\n");
        }
}
```

```
}
class Driver{
    public static void main(String[] args){
            Scanner s = new Scanner(System.in);
            System.out.println("Enter no of rows in first matrix: ");
            int m = s.nextInt();
            System.out.println("Enter no of columns in first matrix: ");
            int n = s.nextInt();
            System.out.println("Enter no of rows in second matrix: ");
            int p = s.nextInt();
            System.out.println("Enter no of columns in second matrix: ");
            int q = s.nextInt();
            if(n == p){
                    Matrix obj = new Matrix();
                    obj.matrixMul(m,n,p,q);
            }
            else{
                    System.out.println("Matrix multiplication cannot be performed...");
            }}}
```

**OUTPUT-**

```
Enter no of rows in first matrix:
3
Enter no of columns in first matrix:
3
Enter no of rows in second matrix:
3
Enter no of columns in second matrix:
3
Enter the elements of matrix A:
1 1 1
2 2 2
3 3 3
Enter the elements of matrix B:
1 2 3
1 2 3
1 1 1
Elements of result matrix C are:
3  5  7
6 10 14
9 15  21
```

Elements of transpose matrix T are:
3 6 9
5 10 15
7 14 21

Assignment 3

1. Write a java program to implement class and object concept

```
class Lamp {
   boolean isOn;
   void turnOn() {
      // initialize variable with value true
      isOn = true;
      System.out.println("Light on? " + isOn);
   }
   void turnOff() {
      // initialize variable with value false
      isOn = false;
      System.out.println("Light on? " + isOn);
   }
}
class Main {
   public static void main(String[] args) {

      // create objects l1 and l2
      Lamp l1 = new Lamp();
      Lamp l2 = new Lamp();

      // call methods turnOn() and turnOff()
      l1.turnOn();
      l2.turnOff();
   }
}
```

**Output**:

```
Light on? true
Light on? False
```

2. Write a java program to implement class and object concept using Scanner class

```
class abc1{
void display()
{
Scanner input = new Scanner(System.in);
System.out.println("Enter a ");
Scanner a = input.nextInt();
System.out.println("Enter b ");
Scanner b = input.nextInt();
int z=a+b;
System.out.println("The sum of two numbers is" + z);
}
}
Public class Main
{
public static void main(String args[]){
abc a1=new abc1();
a1.display();
}
```

OUTPUT - Enter a
10
Enter b
5
The sum of two number is 15

3. Write a java program to implement Constructor concept

**Ans.**

```
class Bike{
Bike(){
  System.out.println("Bike Details are");
  }}
class const2{
public static void main(String args[]){
Bike b = new Bike();
}}
```

OUTPUT - Bike details are

Assignment 4

1. Write a java program to implement operators concept

**Ans**. Operator in java is a symbol that is used to perform operations. For example: +, -, *, / etc. There are many types of operators in java which are given below:
Unary Operator , Arithmetic Operator, Shift Operator, Relational Operator, Bitwise Operator, Logical Operator, Ternary Operator and Assignment Operator.

**Code-**

```
class oper{
public static void main(String args[]){
int a=10;
int b=5;
System.out.println (a+b);
System.out.println (a-b);
System.out.println (a*b);
System.out.println (a/b);
System.out.println (a%b);
}}
```

**OUTPUT-** 15
5
50
2
0

2. Write a java program to implement decision making and control statements using break and continue keywords

**Ans .**
The switch statement is a multiway branch statement. It provides an easy way to dispatch execution to different parts of code based on the value of the expression.

**Code-**

```
class SwitchCaseDemo
```

```
{
   public static void main(String args[])
   {
      int i = 9;
      switch (i) {
      case 0:
         System.out.println("i is zero.");
         break;
      case 1:
         System.out.println("i is one.");
         break;
      case 2:
         System.out.println("i is two.");
         break;
      default:
         System.out.println("i is greater than 2.");
      } } }
```

**OUTPUT -** i is greater than 2.

Break –

The break statement in Java programming language has the following two usages –

When the break statement is encountered inside a loop, the loop is immediately terminated and the program control resumes at the next statement following the loop. It can be used to terminate a case in the switch statement

**Code-**

```
public class Test {
  public static void main(String args[]) {
    int [] numbers = {10, 20, 30, 40, 50};
    for(int x : numbers ) {
      if( x == 30 ) {
        break;
      }
      System.out.print( x );
      System.out.print("\n");
    }}}
```

**OUTPUT -**

10

20

Continue –

The continue keyword can be used in any of the loop control structures. It causes the loop to immediately jump to the next iteration of the loop. In a for loop, the continue keyword causes control to immediately jump to the update statement.

3. Write a java program to implement iteration statements in java using break and continue keywords

In a while loop or do/while loop, control immediately jumps to the Boolean expression.

**Code-**

```
public class Test {
  public static void main(String args[]) {
    int [] numbers = {10, 20, 30, 40, 50};
    for(int x : numbers ) {
      if( x == 30 ) {
        continue;
      }
      System.out.print( x );
      System.out.print("\n");
    }}}
```

**OUTPUT-**

```
10
20
40
50
```

4. Write a java program to implement single or multilevel inheritance

**Ans.**

```
class abc{
void addition(int x,int y){
int z=x+y;
system.out.println("the sum of two no. is" +z);
}
void subtraction(int a,int b){
intz=a+b;
system.out.println("the difference of two no. is" +z);
}}
class derived extends abc{
void multiply(int x,int y){
int z=x*y;
system.out.println("the multiply of two no. is" +z)
}
void division(int x,int y){
int z=x/y;
system.out.println("the division of two no. is" +z)
}
```

```
public static void main(string args[]){
derived d=new derived();
d.addition(10,20);
d.subtraction(20,30)
d.multiply(10,10)
d.division(10,10)
}}
```

OUTPUT - The sum of two no. is 30
The subtraction of two no. is -10
The multiply of two no. is 100
The division of two no. is 1

5. Write a java program to implement hierarchical or hybrid inheritance

```
class C
{
  public void disp()
  {
        System.out.println("C");
  }
}

class A extends C
{
  public void disp()
  {
        System.out.println("A");
  }
}

class B extends C
{
  public void disp()
  {
        System.out.println("B");
  }

}

class D extends A
{
  public void disp()
```

```
  {
        System.out.println("D");
  }
  public static void main(String args[]){

        D obj = new D();
        obj.disp();
  }
}
```

Output:
D

Assignment 5

1. Write a java program to implement data abstraction

   **Ans.** Abstraction is a process of hiding the implementation details and showing only functionality to the user. Another way, it shows only essential things to the user and hides the internal details, for example, sending SMS where you type the text and send the message. You don't know the internal processing about the message delivery. Abstraction lets you focus on what the object does instead of how it does it.

   **Code -**

```
import java.util.Scanner;
abstract class Bike{
abstract void run();
}
 class Bajaj extends Bike{
void run(){
Scanner input = new Scanner (System.in);
System.out.println ("Enter the km");
int km = input.nextInt();
System.out.println("Km is"+km);
 }}
class test{
public static void main(String args[ ]){
Bike b = new Bajaj();
b.run();
 }}
```

   **Output -**

```
Enter the km
30
```

Km is 30

2. Write a java program to implement interface

Interface looks like a class but it is not a class. An interface can have methods and variables just like the class but the methods declared in interface are by default abstract (only method signature, no body, see: Java abstract method). Also, the variables declared in an interface are public, static & final by default.

```
interface MyInterface
{
  /* compiler will treat them as:
   * public abstract void method1();
   * public abstract void method2();
   */
  public void method1();
  public void method2();
}
class Demo implements MyInterface
{
  /* This class must have to implement both the abstract methods
   * else you will get compilation error
   */
  public void method1()
  {
        System.out.println("implementation of method1");
  }
  public void method2()
  {
        System.out.println("implementation of method2");
  }
  public static void main(String arg[])
  {
        MyInterface obj = new Demo();
        obj.method1();
  }
}
```

Output:
implementation of method1

3. Write a java program to implement polymorphism(method overloading)

   **Ans.** If a class has multiple methods having same name but different in parameters, it is known as Method Overloading. If we have to perform only one operation, having same name of the

methods increases the readability of the program.

Suppose you have to perform addition of the given numbers but there can be any number of arguments, if you write the method such as a(int,int) for two parameters, and b(int,int,int) for three parameters then it may be difficult for you as well as other programmers to understand the behavior of the method because its name differs. So, we perform method overloading to figure out the program quickly.

**Code -**

```
 import java.util.Scanner;
class Adder{
 static int add(int a, int b){
  return (a+b);
 }
 static int add(double a, double b, double c){
  return (int)(a+b+c);
 }}
class over{
public static void main(String args[ ]) {
 System.out.println(Adder.add(31,19));
 System.out.println(Adder.add(31,19,10));
 } }
```

**Output -**

50
60

4. Write a java program to implement polymorphism(method overriding)

**Ans.** If subclass (child class) has the same method as declared in the parent class, it is known as method overriding in Java.

In other words, If a subclass provides the specific implementation of the method that has been declared by one of its parent class, it is known as method overriding.

**Code -**

```
import java.util.Scanner;
class Vehicle{
 void run( ){
  System.out.println("Your vehicle is Running at 70km/hr");
 }}
class Bike extends Vehicle{
 void run( ){
  System.out.println("Your vehicle is Running at 90km/hr");
 }}
class overr {
```

```
 static public void  main(String args[ ]){
   Bike b = new Bike();
    b.run();
  }}
```

**Output -**

Your vehicle is Running at 90km/hr

Assignment 6

1. Case study of Strings

   (Note: it should cover understanding to developing Strings and String constructors, special string operations,
   character extraction, searching and comparing strings, string Buf er class.)

   Ans-

   · Java String-: In Java, string is basically an object that represents sequence of char values. An array of
   characters works same as Java string.

   For example:

   ```
   char[] ch={'j','a','v','a','t','p','o','i','n','t'};
   String s=new String(ch);
   String s="Amit";
   ```

   2- Java String class provides a lot of methods to perform operations on string such as compare(), concat(),
   equals(), split(), length(), replace(), compareTo(), intern(), substring() etc.

   · How to create a string object?

   There are two ways to create String object:

   · By string literal

   · By new keyword

   1) String Literal

Java String literal is created by using double quotes. For Example:

· String s="welcome";

Each time you create a string literal, the JVM checks the "string constant pool" first. If the string already exists in
the pool, a reference to the pooled instance is returned. If the string doesn't exist in the pool, a new string instance is
created and placed in the pool. For example:

· String s1="Welcome";

· String s2="Welcome"

2) By new keyword

String s=new String("Welcome");//creates two objects and one reference variabl

In such case, JVM will create a new string object in normal (non-pool) heap memory, and the literal "Welcome"
will be placed in the string constant pool. The variable s will refer to the object in a heap (non-pool).

Java String Example

```
public class StringExample{
public static void main(String args[]){
String s1="java";
char ch[]={'s','t','r','i','n','g','s'};
String s2=new String(ch);//converting char array to string
String s3=new String("example");//creating java string by new keyword
System.out.println(s1);
System.out.println(s2);
System.out.println(s3);
}}
```

OUTPUT-:

```
Java
Hello
World
```

3-Java String class methods

The java.lang.String class provides a lot of methods to work on string. By the help of these methods, we can
perform operations on string such as trimming, concatenating, converting, comparing, replacing strings etc.

Java String is a powerful concept because everything is treated as a string if you submit any form in window based,
web based or mobile application.
Let's see the important methods of String class.
A-Java String toUpperCase() and toLowerCase() method
The java string toUpperCase() method converts this string into uppercase letter and string toLowerCase() method
into lowercase letter.

```
String s="Amit";
System.out.println(s.toUpperCase());
System.out.println(s.toLowerCase());
System.out.println(s);
```

OUTPUT-:
AMIT
Amit
Amit

4-Java StringBuffer class
Java StringBuffer class is used to create mutable (modifiable) string. The StringBuffer class in java is same as
String class except it is mutable i.e. it can be changed.
· Important Constructors of StringBuffer class
Constructor Description
StringBuffer() creates an empty string buffer with the initial capacity of 16.
StringBuffer(String str) creates a string buffer with the specified string.
StringBuffer(int capacity) creates an empty string buffer with the specified capacity as length.
5-Java string compare
We can compare string in java on the basis of content and reference.
It is used in authentication (by equals() method), sorting (by compareTo() method), reference matching (by ==
operator) etc.
There are three ways to compare string in java:
· By equals() method
· By = = operator
· By compareTo() method
1) String compare by equals() method

The String equals() method compares the original content of the string. It compares values of string for equality.

String class provides two methods:

· public boolean equals(Object another) compares this string to the specified object.

· public boolean equalsIgnoreCase(String another) compares this String to another string, ignoring case.

```
class Teststringcomparison1{
public static void main(String args[]){
String s1="Amit";
String s2="Amit";
String s3=new String("Amit");
String s4="Saurav";
System.out.println(s1.equals(s2));//true
System.out.println(s1.equals(s3));//true
System.out.println(s1.equals(s4));//false
}
}
```

Output:

true

true

false

2) String compare by == operator

The = = operator compares references not values.

```
class Teststringcomparison3{
public static void main(String args[]){
String s1="Amit";
String s2="Amit";
String s3=new String("Amit");
System.out.println(s==s2);//true (because both refer to same instance)
System.out.println(s==s3);//false(because s3 refers to
instance created i n nonpool)
}
}
```

Output:

true

false

3) String compare by compareTo() method

The String compareTo() method compares values lexicographically and returns an integer value that describes if

first string is less than, equal to or greater than second string.

Suppose s1 and s2 are two string variables. If:

· s1 == s2 :0

· s1 > s2 :positive value

· s1 < s2 :negative value

```
class Teststringcomparison4{
public static void main(String args[]){
String s1="Sachin";
String s2="Sachin";
String s3="Ratan";
System.out.println(s1.compareTo(s2));//0
System.out.println(s1.compareTo(s3));//1(because s1>s3)
System.out.println(s3.compareTo(s1));//-1(because s3 < s1 )
}
}
```

Output:

0

1

-1

6 Difference between String and StringBuffer

There are many differences between String and StringBuffer. A list of differences between String and StringBuffer are given below:

| No. | String | StringBuffer |
|---|---|---|
| 1 | String class is immutable | StringBuffer class is mutable |
| 2 | String is slow and consumes more memory when you concat too many strings because every time it creates new instance. | StringBuffer is fast and consumes less memory when |

| | | |
|---|---|---|
| 3 | String class overrides the equals() method of Object class. So you can compare the contents of two strings by equals() method. | StringBuffer class doesn't override the equals() method of Object class |

Performance Test of String and StringBuffer

```
public class ConcatTest{
public static String concatWithString() {
String t = "Java";
for (int i=0; i<10000; i++){
t = t + "Tpoint";
}
return t;
}
public static String concatWithStringBuffer(){
StringBuffer sb = new StringBuffer("Java");
for (int i=0; i<10000; i++){
sb.append("Tpoint");
}
return sb.toString();
}
public static void main(String[] args){
long startTime = System.currentTimeMillis();
concatWithString();
System.out.println("Time taken by Concating with String: "+(System.curre
ntTimeMillis()-
startTime)+"ms");
startTime = System.currentTimeMillis();
concatWithStringBuffer();
System.out.println("Time taken by Concating with StringBuffer:
"(System.currentTimeMillis()-
startTime)+"ms");
}
}
```

Output-:
Time taken by Concating with String: 578ms
Time taken by Concating with StringBuffer: 0ms

2. Case study of Multithreading

**Ans. Multitasking**

Multitasking has the same meaning of multiprogramming but in a more general sense, as it refers to having multiple (programs, processes, tasks, threads) running at the same time. This term is used in modern operating systems when multiple tasks share a common processing resource (e.g., CPU and Memory). At any time the CPU is executing one task only while other tasks waiting their turn. The illusion of parallelism is achieved when the CPU is reassigned to another task (i.e. process or thread context switching).

There are subtle differences between multitasking and multiprogramming. A task in a multitasking operating system is not a whole application program but it can also refer to a "thread of execution" when one process is divided into sub-tasks. Each smaller task does not hijack the CPU until it finishes like in the older multiprogramming but rather a fair share amount of the CPU time called quantum.

Just to make it easy to remember, both multiprogramming and multitasking operating systems are (CPU) time sharing systems. However, while in multiprogramming (older OSs) one program as a whole keeps running until it blocks, in multitasking (modern OSs) time sharing is best manifested because each running process takes only a fair quantum of the CPU time.

**Multiprogramming**

In a multiprogramming system there are one or more programs loaded in main memory which are ready to execute. Only one program at a time is able to get the CPU for executing its instructions (i.e., there is at most one process running on the system) while all the others are waiting their turn.

The main idea of multiprogramming is to maximize the use of CPU time. Indeed, suppose the currently running process is performing an I/O task (which, by definition, does not need the CPU to be accomplished). Then, the OS may interrupt that process and give the control to one of the other in-main-memory programs that are ready to execute (i.e. process context switching). In this way, no CPU time is wasted by the system waiting for the I/O task to be completed, and a running process keeps executing until either it voluntarily releases the CPU or when it blocks for an I/O operation. Therefore, the ultimate goal of multiprogramming is to keep the CPU busy as long as there are processes ready to execute.

**Multithreading**

Multithreading in java is a process of executing multiple threads simultaneously.

A thread is a lightweight sub-process, the smallest unit of processing. Multiprocessing and multithreading, both are used to achieve multitasking.

However, we use multithreading than multiprocessing because threads use a shared memory area.

They don't allocate separate memory area so saves memory, and context-switching between the threads takes less time than process.
Java Multithreading is mostly used in games, animation, etc.

**Code -**

```
class A extends Thread{
public void run(){
 System.out.println("thread is running ");
}
public static void main(String args[]){
A obj = new A();
obj.start();
}}
```

**OUTPUT -** thread is running

**2. Write a java program to create , start and run a thread using Runnable intetrface ?**

**Ans. PROCESS-**

An executing program is called a process.
Every process has its separate address space.Process-based multitasking allows a computer to run two or more than two programs concurrently.Communication between two processes is expensive and limited.Context switching from one process to another process is expensive.A process has its own address space, global variables, signal handlers, open files, child processes, accounting information.Process are also called heavyweight task.Process-based multitasking is not under the control of Java.You are working on text editor it refers to the execution of a process.

**THREAD -**

A thread is a small part of a process.
All the threads of a process share the same address space cooperatively as that of a process.Thread-based multitasking allows a single program to run two or more threads concurrently.Communication between two threads is less expensive as compared to process.Context switching from one thread to another thread is less expensive as compared to process.A thread has its own register, state, stack, program counter.Thread are also called lightweight task.Thread-based multitasking is under the control of Java.You are printing a file from text editor while working on it that resembles the execution of a thread in the process.

**Code -**

```
class Run1 implements Runnable{
public void run(){
 System.out.println("thread is running ");
}
public static void main(String args[]){
A a1 = new A();
Thread obj = new Thread(a1);
```

```
obj.start();
}}
```

**OUTPUT -** thread is running

**3.Write a java program to implement multithreading (create 3 threads and print number from 1-5 using these threads ) ?**

**Ans. Thread scheduler**

Thread scheduler in java is the part of the JVM that decides which thread should run.
There is no guarantee that which runnable thread will be chosen to run by the thread scheduler.
Only one thread at a time can run in a single process.
The thread scheduler mainly uses preemptive or time slicing scheduling to schedule the threads.

**Code -**

```
class A extends Thread{
public void run(){
  for(int i = 1; i<=5;i++) {
   try
    {Thread.sleep(300);}
   catch(InterruptedException e)
    {System.out.println(e);}
   System.out.println(i);
    }}}
class check{
 public static void main(String args[]){
 A obj1 = new A();
 obj1.start();
 A obj2 = new A();
 obj2.start();
 A obj3 = new A();
 obj3.start();
  }}
```

**OUTPUT -** 1 1 1 2 2 2 3 3 3 4 4 4 5 5 5

**4.Write a java program to implement methods of Thread class (getName(), setName(), currentThread(), join()) ?**

**Ans. Code -**

```
class A extends Thread{
public void run(){
   for(int i = 0; i<3;i++) {
   try
    {Thread.sleep(300);}
```

```
  catch(InterruptedException e)       {System.out.println(e);}
  System.out.println(i);
   }}}
class namecheck{
 public static void main(String args[]){
 String tn = Thread.currentThread().getName();
 System.out.println(tn);
A obj1 = new A();
 System.out.println("Name of obj1:"+obj1.getName());
 obj1.start();
 try{
  obj1.join();   }
catch(Exception e){System.out.println(e);}
  A obj2 = new A();
 obj2.setName("obj3"));
 System.out.println("After changing name of obj2:"+obj2.getName());
 obj2.start();
  }  }
```

**OUTPUT -**

```
main
Thread-0
1
2
3
4
5
```

3. Case study of IOStreams

Java I/O (Input and Output) is used *to process the input* and *produce the output*.
Java uses the concept of a stream to make I/O operation fast. The java.io package contains all the classes required for input and output operations.
We can perform file handling in Java by Java I/O API.
Stream
A stream is a sequence of data. In Java, a stream is composed of bytes. It's called a stream because it is like a stream of water that continues to flow.
In Java, 3 streams are created for us automatically. All these streams are attached with the console.
1) System.out: standard output stream
2) System.in: standard input stream
3) System.err: standard error stream

Let's see the code to print output and an error message to the console.
System.out.println("simple message");
System.err.println("error message");
Let's see the code to get input from console.
int i=System.in.read();//returns ASCII code of 1st character
System.out.println((char)i);//will print the character

Java FileOutputStream is an output stream used for writing data to a file.

```
import java.io.FileOutputStream;
public class FileOutputStreamExample {
   public static void main(String args[]){
        try{
          FileOutputStream fout=new FileOutputStream("D:\\testout.txt");
          fout.write(65);
          fout.close();
          System.out.println("success...");
         }catch(Exception e){System.out.println(e);}
    }
}
```

Output:

```
Success...
```

Java FileInputStream class obtains input bytes from a file.

```
import java.io.FileInputStream;
public class DataStreamExample {
    public static void main(String args[]){
        try{
          FileInputStream fin=new FileInputStream("D:\\testout.txt");
          int i=fin.read();
          System.out.print((char)i);
          fin.close();
        }catch(Exception e){System.out.println(e);}
       }
      }
```

**Note:** Before running the code, a text file named as **"testout.txt"** is required to be created. In this file, we are having following content:
Welcome to javatpoint.

Assignment 7

1. Case study of Exception handling
   Ans-
   An exception (or exceptional event) is a problem that arises during the execution of a program. When an Exception occurs the normal flow of the program is disrupted and the program/Application terminates abnormally, which is not recommended, therefore, these exceptions are to be handled.
   An exception can occur for many different reasons. Following are some scenarios where an exception occurs.

A user has entered an 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.

Based on these, we have three categories of Exceptions. You need to understand them to know how exception handling works in Java.

Checked exceptions – A checked exception is an exception that is checked (notified) by the compiler at compilation-time, these are also called as compile time exceptions. These exceptions cannot simply be ignored, the programmer should take care of (handle) these exceptions.

For example, if you use FileReader class in your program to read data from a file, if the file specified in its constructor doesn't exist, then a FileNotFoundException occurs, and the compiler prompts the programmer to handle the exception.
Example

```
import java.io.File;
import java.io.FileReader;

public class FilenotFound_Demo {

 public static void main(String args[]) {
File file = new File("E://file.txt");
    FileReader fr = new FileReader(file);
  }
}
```

If you try to compile the above program, you will get the following exceptions.
Output

```
C:\>javac FilenotFound_Demo.java
FilenotFound_Demo.java:8: error: unreported exception FileNotFoundException; must be caught or declared to be thrown
    FileReader fr = new FileReader(file);
1 error
```

Note – Since the methods read() and close() of FileReader class throws IOException, you can observe that the compiler notifies to handle IOException, along with FileNotFoundException.

Unchecked exceptions – An unchecked exception is an exception that occurs at the time of execution. These are also called as Runtime Exceptions. These include programming bugs, such as logic errors or improper use of an API. Runtime exceptions are ignored at the time of compilation.
For example, if you have declared an array of size 5 in your program, and trying to call the 6th element of the array then an ArrayIndexOutOfBoundsExceptionexception occurs.

Example

```
public class Unchecked_Demo {

  public static void main(String args[]) {
    int num[] = {1, 2, 3, 4};
    System.out.println(num[5]);
  }
}
```

If you compile and execute the above program, you will get the following exception.

Output

```
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: 5
        at Exceptions.Unchecked_Demo.main(Unchecked_Demo.java:8)
```

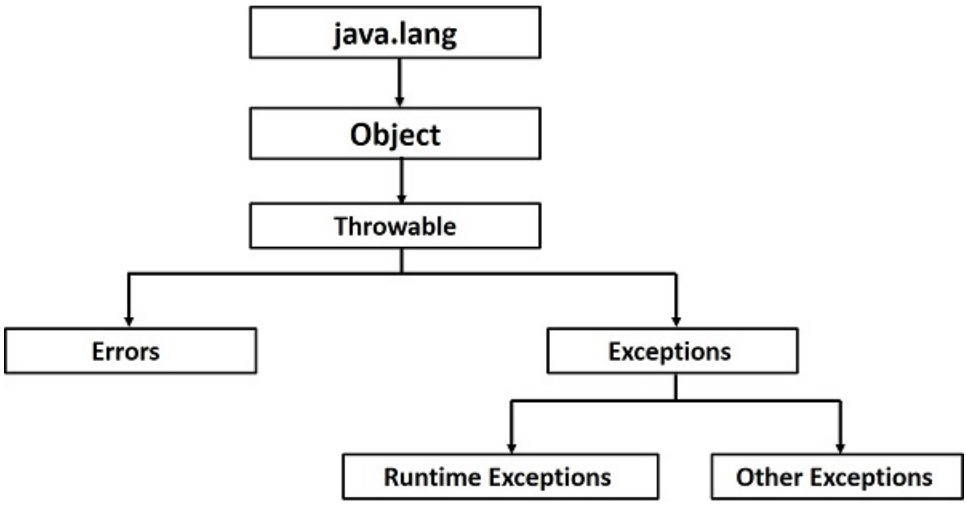
Errors – These are not exceptions at all, but problems that arise beyond the control of the user or the programmer. Errors are typically ignored in your code because you can rarely do anything about an error. For example, if a stack overflow occurs, an error will arise. They are also ignored at the time 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 abnormal conditions that happen in case of severe failures, these 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.




Following is a list of moCatching 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 –

Syntax

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

The code which is prone to exceptions is placed in the try block. When an exception occurs, that exception occurred is handled by catch block associated with it. Every try block should be immediately followed either by a catch block or finally 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 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 will produce the following result –

Output

```
Exception thrown  :java.lang.ArrayIndexOutOfBoundsException: 3
Out of the blockst common checked and unchecked Java's Built-in Exceptions.
```

Multiple Catch Blocks

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

Syntax

```
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;
}
```

Catching Multiple Type of Exceptions

Since Java 7, you can handle more than one exception using a single catch block, this feature simplifies the code. Here is how you would do it −

```
catch (IOException|FileNotFoundException ex) {
   logger.log(ex);
   throw ex;
```

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 difference between throws and throw keywords, throws is used to postpone the handling of a checked exception and throw is used to invoke an exception explicitly.

The following method declares that it throws a RemoteException −

Example
```
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 −

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

  public void withdraw(double amount) throws RemoteException,
    InsufficientFundsException {
    // Method implementation
  }
  // Remainder of class definition
}
```
The Finally Block
The finally block follows a try block or a catch block. A finally block of code always executes, irrespective of occurrence of an Exception.

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 −

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 will produce the following result –

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

2. Case study of Applet

**Ans.**

Applet is a special type of program that is embedded in the webpage to generate the dynamic content. It runs inside the browser and works at client side.

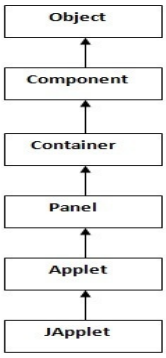
**Advantage of Applet**

1.There are many advantages of applet. They are as follows:
2.It works at client side so less response time.
3.Secured
4.It can be executed by browsers running under many plateforms, including Linux, Windows, Mac Os etc.

**Drawback of Applet**

1.Plugin is required at client browser to execute applet.

**Hierarchy of Applet**




As displayed in the above diagram, Applet class extends Panel. Panel class extends Container which is the subclass of Component.

**Lifecycle of Java Applet**

1.Applet is initialized.
2.Applet is started.
3.Applet is painted.
4.Applet is stopped.
5.Applet is destroyed.


**Lifecycle methods for Applet:**

The java.applet.Applet class 4 life cycle methods and java.awt.Component class provides 1 life cycle methods for an applet.

**java.applet.Applet class**

For creating any applet java.applet.Applet class must be inherited. It provides 4 life cycle methods of applet.

1.public void init(): is used to initialized the Applet. It is invoked only once.
2.public void start(): is invoked after the init() method or browser is maximized. It is used to start the Applet.
3.public void stop(): is used to stop the Applet. It is invoked when Applet is stop or browser is minimized.
4.public void destroy(): is used to destroy the Applet. It is invoked only once.

**java.awt.Component class**
The Component class provides 1 life cycle method of applet.
public void paint(Graphics g): is used to paint the Applet. It provides Graphics class object that can be used for drawing oval, rectangle, arc etc.

**Who is responsible to manage the life cycle of an applet?**
Java Plug-in software.

**How to run an Applet?**
There are two ways to run an applet
1.By html file.
2.By appletViewer tool (for testing purpose).

**Simple example of Applet by html file:**
To execute the applet by html file, create an applet and compile it. After that create an html file and place the applet code in html file. Now click the html file.

```
//First.java
import java.applet.Applet;
import java.awt.Graphics;
public class First extends Applet{

public void paint(Graphics g){
g.drawString("welcome",150,150);
}

}
```

Note: class must be public because its object is created by Java Plugin software that resides on the browser.
myapplet.html

```
<html>
<body>
<applet code="First.class" width="300" height="300">
</applet>
</body>
</html>
```

Simple example of Applet by appletviewer tool:
To execute the applet by appletviewer tool, create an applet that contains applet tag in comment and compile it. After that run it by: appletviewer First.java. Now Html file is not required but it is

for testing purpose only.

```
//First.java
import java.applet.Applet;
import java.awt.Graphics;
public class First extends Applet{

public void paint(Graphics g){
g.drawString("welcome to applet",150,150);
}

}
/*
<applet code="First.class" width="300" height="300">
</applet>
*/
```

To execute the applet by appletviewer tool, write in command prompt:

```
c:\>javac First.java
c:\>appletviewer First.java
```

**Method paint()** is automatically called whenever there is a need to display the applet window, this need could arise in certain situations -
When the applet window is brought up on the screen for the first time.
When the applet window is brought up on the screen from a minimized state, this leads the redrawing of the applet window and hence paint() method is automatically called.
When the applet window is stretched to a new size, this leads to redrawing of applet window to a new size and hence paint() method is automatically called.

**Signature of paint() method**

public void paint(Graphics g)

The method paint() gives us an access to an object of Graphics class type in our applet code. Using the object of Graphics class, we can call drawString() method of Graphics class to write a text message in the applet window.

**Example -**

```
import java.awt.*;
import java.applet.*;
/*
<applet  code="Applet24" width=400 height=200>
</applet>
*/
public class Applet24 extends Applet
{
String str ="";

public void init()
{
```

```
str="init-";
}
public void start()
{
str=str+"start-";
}
public void stop()
{
str=str+"stop-";
}
public void paint(Graphics g)
{
str=str+"paint-";;
g.drawString(str,40,100);
g.drawString("Hello from the Applet.", 40,40);
g.drawString("How are you doing?", 40, 60);
g.drawString("We wish you a pleasant day today.", 40, 80);
}

}
```

**Output**