Master of Computer Applications (MCA)

Object Oriented Programming with C++ and JAVA Lab

(OMCACO107P24)

Self-Learning Material (SEM 1)

Jaipur National University
Centre for Distance and Online Education

Established by Government of Rajasthan
Approved by UGC under Sec 2(f) of UGC ACT 1956
&
NAAC A+ Accredited

TABLE OF CONTENTS

EXPERT COMMITTEE

Prof. Sunil Gupta
(Department of Computer and Systems Sciences, JNU Jaipur)

Dr. Deepak Shekhawat
(Department of Computer and Systems Sciences, JNU Jaipur)

Dr. Shalini Rajawat
(Department of Computer and Systems Sciences, JNU Jaipur)

COURSE COORDINATOR

Ms. Heena Shrimali
(Department of Computer and Systems Sciences, JNU Jaipur)

UNIT PREPARATION

Unit Writer(s)	**Assisting & Proofreading**	**Unit Editor**
Ms. Heena Shrimal (Department of Computer and Systems Sciences, JNU Jaipur)	Mr. Hitendra Agarwal (Department of Computer and Systems Sciences, JNU Jaipur)	Mr. Shish Dubey (Department of Computer and Systems Sciences, JNU Jaipur)

Secretarial Assistance

Mr. Mukesh Sharma

COURSE INTRODUCTION

"Object-oriented programming is an exceptionally bad idea which could only have originated in California."

- Edsger W. Dijkstra

The course starts with an overview of the object-oriented paradigm and its elements, discussing the advantages and disadvantages of the OO methodology. Students will delve into C++ fundamentals, including data types, operators, expressions, and control flow. Essential topics covered include arrays, strings, pointers, and functions, along with the creation and management of classes and objects. The course also addresses constructors and destructors, operator overloading, inheritance, virtual functions, and polymorphism to provide a comprehensive understanding of OOP principles in C++. File handling in C++ is another significant aspect of this course. Students will learn about console streams and console stream classes, including formatted and unformatted console I/O operations and manipulators. The course covers file streams, classes, file modes, file pointers, and file manipulations, as well as file input and output operations. Additionally, students will learn about exception handling in C++, enabling them to write robust and error-resistant programs.

The course also offers an introduction to Java, a versatile and widely-used programming language. Students will learn about Java's data types, variables, and arrays, as well as operators and control statements. The course covers the creation and management of classes, objects, and methods, along with key OOP concepts such as inheritance, packages, and interfaces. Exception handling, multithreaded programming, strings, and input/output operations are also discussed, providing students with the skills needed to develop efficient Java applications.

Multithreading is a powerful concept that allows for the concurrent execution of code. This course introduces students to the differences between non-threaded and threaded applications, focusing on the creation and management of threads. Students will learn to implement the Runnable interface, gaining a practical understanding of multithreading and its applications in real-world scenarios.

Course Outcomes:

At the completion of the course, a student will be able to:

1. Acquire profound knowledge of object oriented programming.
2. Demonstrate the difference between the solutions offered by traditional imperative problem solving method and object-oriented method by class inheritance, data encapsulation, and polymorphism as fundamental building blocks to generate reusable code.
3. Understand and implement error handling and file handling routines.
4. Explain the Internet Programming, using Java Applets.
5. Create and design a full set of UI widgets and other components, including windows, menus, buttons, checkboxes, text fields, scrollbars and scrolling lists, using Abstract Windowing Toolkit (AWT).
6. Describe to access database through Java programs, using Java Database Connectivity (JDBC)
7. Develop Mini Projects using constructs of OOPs and Java.

Acknowledgements:

The content we have utilized is solely educational in nature. The copyright proprietors of the materials reproduced in this book have been tracked down as much as possible. The editors apologize for any violation that may have happened, and they will be happy to rectify any such material in later versions of this book.

OOPs Using C++ Lab

Assignment 1: Implementing a Class and Object

Program Statement: Write a C++ program to make a class **Rectangle** with attributes length and width. Include member functions to:

1. Set the dimensions of the rectangle.
2. Compute and return the area of the rectangle.
3. Calculate and go back the perimeter of the rectangle.
4. Display the dimensions, area, and perimeter.

Solution Description: This program will help students understand how to define a class with private attributes and public member functions. The **Rectangle** class will encapsulate the properties of a rectangle and provide methods to manipulate and access these properties. The program will include a constructor to initialize the rectangle's dimensions, methods to calculate the area and perimeter, and a display function to output the rectangle's details. This assignment reinforces concepts of encapsulation, data hiding, and basic object manipulation in C++.

Assignment 2: Constructors and Destructors

Program Statement: Create a class **Complex** to represent complex numbers. Implement:

1. A default constructor to initialize the real and imaginary parts to zero.
2. A “parameterized constructor” to initialize the real and imaginary parts to given values.
3. A “destructor” to display a message when an object is destroyed.
4. A “member function” to display the complex number in the form **a + bi**.

Solution Description: This assignment emphasizes the use of constructors and destructors in a class. The **Complex** class will have attributes for the real and imaginary parts. Constructors will initialize these attributes, either to default values or to user-provided values. The destructor will be used to demonstrate when an object goes out of scope and is destroyed. The display function will format and print the complex number. This task helps in understanding object lifecycle management and resource cleanup in C++.

Assignment 3: Implementing Inheritance

Program Statement: Create a "base class" **Shape** with a pure "virtual function" **area()**. Derive two classes **Circle** and **Square** from **Shape**. Implement:

1. The constructor for each derived class.
2. The **area()** function to calculate and return the area for each shape.
3. A function to display the area.

Solution Description: This assignment introduces the concept of inheritance and polymorphism. The **Shape** class serves as an "abstract base class" with a pure "virtual function" **area()**. The "derived classes" **Circle** and **Square** implement the **area()** function to calculate the area specific to each shape. The constructors initialize the radius and side length, respectively. By using base class pointers to call the **area()** function, students will understand dynamic binding and polymorphism in C++.

Assignment 4: Function Overloading

Program Statement: Write a C++ program to demonstrate function overloading by creating a class **Math** with multiple **add()** functions to handle:

1. Addition of two integers.
2. Addition of two floating-point numbers.
3. Addition of three integers.

Solution Description: "Function overloading" allows multiple functions with the same name but different parameters. The **Math** class will have overloaded **add()** functions to handle different types and numbers of arguments. This program will show how the same function name can be used to do different operations based on the input parameters. This assignment helps in understanding the concept of function overloading and its applications in C++.

Assignment 5: Operator Overloading

Program Statement: Create a class **Complex** to represent complex numbers. Overload the **+** operator to add two complex numbers. The program should:

1. Include a constructor to initialize the real and imaginary parts.
2. Overload the **+** operator.
3. Display the result of the addition.

Solution Description: "Operator overloading" allow operators to be redefined for user-defined types. The **Complex** class will include a constructor for initialization and an

overloaded + operator to add two **Complex** objects. The program will create **Complex** objects, perform the addition using the overloaded operator, and display the result. This assignment covers operator overloading and custom behavior for operators, allowing students to extend the functionality of existing operators to work with user-defined types.

Assignment 6: Inheritance with Function Overriding

Program Statement: Create a "Base class" **Animal** with a "Virtual function" **sound()**. Derive two classes **Dog** and **Cat** from **Animal** and override the **sound()** function in each derived class. The program should:

1. Create objects of **Dog** and **Cat**.
2. Call the **sound()** function by a pointer to the "Base class".

Solution Description: "Function overriding" allows a derived class to provide a definite implementation of a function already defined in its base class. The **Animal** class will have a virtual **sound()** function, which will be overridden in the **Dog** and **Cat** classes to provide specific sounds. By using base class pointers to call the **sound()** function, the program will demonstrate polymorphism. This assignment helps in understanding inheritance, function overriding, and runtime polymorphism in C++.

Assignment 7: Implementing a Copy Constructor

Program Statement: Create a "class" **Book** with attributes title, author, and price. Implement:

1. A "parameterized constructor" to initialize the attributes.
2. A copy constructor to create a copy of a **Book** object.
3. A function to display the book details.

Solution Description: A copy constructor is used to create a new object as a copy of an existing object. The **Book** class will have attributes for the title, author, and price, and a parameterized constructor to initialize them. The copy constructor will create a new **Book** object with the same attribute values as an existing object. The display function will print the details of the book. This assignment helps in understanding deep copying and the role of copy constructors in C++.

Assignment 8: Implementing Dynamic Memory Allocation

Program Statement: Create a class **Student** with attributes name and marks. Implement:

1. A constructor to dynamically allocate memory for the name.

2. A destructor to deallocate the memory.
3. A function to display the student details.

Solution Description: Dynamic memory allocation involves allocating memory at runtime using pointers. The **Student** class will have a constructor that allocates memory for the name attribute and a destructor that deallocates this memory to prevent memory leaks. The display function will output the student's details. This assignment helps in understanding dynamic memory management, constructors, destructors, and the importance of resource management in C++.

Assignment 9: Friend Function

Program Statement: Make a class **Box** with private attributes length, width, and height. Implement:

1. A "constructor" to initialize the attributes.
2. A "friend function" to calculate and return the volume of the box.
3. A function to display the dimensions and volume of the box.

Solution Description: A "Friend function" is a non-member function that has access to the private and protected member of a class. The **Box** class will have a constructor to initialize its dimensions and a friend function to calculate the volume. The display function will print the box's dimensions and volume. This assignment helps in understanding friend functions and their use cases in C++.

Assignment 10: Template Class

Program Statement: Write a template class **Array** that can store elements of any data type. Implement:

1. A constructor to initialize the array with a given size.
2. A function to add elements to the array.
3. A function to show the elements of the array.

Solution Description: Templates allow classes and functions to operate with generic types. The **Array** template class will support any data type, provide flexibility and reusability. The constructor will initialize the array with a specified size, the function to add elements will store values in the array, and the display function will print all elements. This assignment helps in understanding templates and their benefits in creating generic and reusable code in C++.

Assignment 11: Implementing a Linked List

Program Statement: Create a class **Linked List** to represent a "singly linked list" of integers. Implement:

1. A constructor to initialize an empty list.
2. A function to add a node at the end.
3. A function to cross out a node from the beginning.
4. A function to show the elements of the list.

Solution Description: The **Linked List** class will encapsulate the properties of a singly linked list. The class will have a nested **Node** structure by an integer data field and a pointer to the next node. The constructor will initialize an empty list. The **add Node** function will add nodes to the end of the list, while the **deleteNode** function will remove the node at the beginning. The **display** function will traverse and print the list elements. This assignment reinforces concepts of dynamic memory allocation and pointer manipulation in C++.

Assignment 12: Implementing a Stack Using Class

Program Statement: Create a class **Stack** to represent a stack of integers. Implement:

1. A constructor to initialize an empty stack.
2. A function to push an element on the stack.
3. A function to pop an element from the stack.
4. A function to show the elements of the stack.

Solution Description: The **Stack** class will use an array or a linked list to store stack elements. The constructor will initialize the stack, and the **push** function will insert elements to the top. The **pop** function will remove the top element, and the "**display** function" will print all elements from the top to the bottom. This assignment helps in understanding stack operations and their implementation in C++.

Assignment 13: Implementing a Queue Using Class

Program Statement: Create a class **Queue** to represent a queue of integers. Implement:

1. A constructor to initialize an empty queue.
2. A function to enqueue an element on the end.
3. A function towards dequeue an element from the front.
4. A function to show the elements of the queue.

Solution Description: The **Queue** class will use an array or a linked list to manage queue elements. The constructor will initialize the queue, and the **enqueue** function will add elements to the end. The **dequeue** function will remove elements from the front, and the **display** function will print all elements from the front to the end. This assignment covers the concept of queue operations and their implementation in C++.

Assignment 14: Implementing Polymorphism

Program Statement: Create a "Base class" **Vehicle** with a "virtual function" **display()**. Derive two classes **Car** as well as **Bike** from **Vehicle**. Override the **display()** function in each derived class. The program should:

1. Create objects of **Car** and **Bike**.
2. Call the **display()** function using a pointer towards the base class.

Solution Description: Polymorphism allows methods to be used interchangeably based on the object type at runtime. The **Vehicle** class will have a virtual **display()** function, which will be overridden in the **Car** and **Bike** classes to provide specific implementations. The program will use base class pointers to demonstrate polymorphism by calling the **display()** function on **Car** and **Bike** objects. This assignment helps understand runtime polymorphism and dynamic binding in C++.

Assignment 15: Implementing Abstract Classes

Program Statement: Create an abstract "Base class" **Employee** with a pure virtual function **calculateSalary()**. Derive 2 classes **FullTimeEmployee** and **PartTimeEmployee** from **Employee**. Implement:

1. The **calculateSalary()** function in each derived class.
2. A function to display the salary details.

Solution Description: Abstract classes cannot be instantiated and are used to define interfaces for derived classes. The **Employee** class will have a pure virtual **calculateSalary()** function, making it abstract. The **FullTimeEmployee** and **PartTimeEmployee** classes will provide concrete implementations of the **calculateSalary()** function. The display function will print the salary details. This assignment helps understand abstract classes, pure virtual functions, and their role in defining interfaces in C++.

Assignment 16: Implementing a Template Function

Program Statement: Create a template "Function" **findMax** to find the supreme of two elements. The program should:

1. Use the **findMax** function with different data types (int, float, char).
2. Display the results.

Solution Description: Template functions allow a function to operate with generic types. The **findMax** function will compare two elements of any data type and return the maximum. The program will demonstrate the function with different data types, showcasing its versatility and reusability. This assignment helps understand the concept of templates and their benefits in creating generic functions in C++.

Assignment 17: Exception Handling

Program Statement: Create a C++ program to demonstrate exception handling. The program should:

1. Implement a function that performs division of two numbers.
2. Throw an exception if the divisor is zero.
3. Catch the exception and display an appropriate error message.

Solution Description: Exception handling allows a program to handle runtime errors gracefully. The division function will throw an exception if an attempt is made to divide by zero. The program will catch the exception and display an error message, preventing the program from crashing. This assignment helps in understanding the try, catch, and throw mechanisms in C++ for robust error handling.

Assignment 18: Implementing Copy Constructor and Assignment Operator

Program Statement: Create a class **String** to represent a dynamic string. Implement:

1. A parameterized constructor to initialize the string.
2. A copy constructor to create a copy of a **String** object.
3. An overloaded assignment operator to assign one **String** object to another.
4. A function to display the string.

Solution Description: The **String** class will manage a dynamically allocated character array. The copy constructor will ensure a deep copy of the string, and the assignment operator will handle assignment between objects, preventing memory leaks. The display function will print the string. This assignment covers dynamic memory management, copy constructors, and assignment operators in C++.

Assignment 19: Implementing Static Members

Program Statement: Create a class **Counter** with a static data member to keep pathway of the number of objects created. Implement:

1. A constructor to increment the counter.
2. A static member function to display the count of objects created.

Solution Description: Static members are joint among all objects of a class. The **Counter** class will have a static data member to count the number of objects. The constructor will increment this counter, and the static member function will display the count. This assignment helps in understanding static data members and functions, and their shared nature across all instances of a class in C++.

Assignment 20: File Handling

Program Statement: Create a class **File Handler** to perform basic file operations. Implement:

1. A function to write data to a file.
2. A function to read data from the file and display it.

Solution Description: File handling allows programs to read from and write to files. The **FileHandler** class will use file streams to perform these operations. The write function will output data to a file, and the read function will input data from the file and display it. This assignment helps in understanding file I/O operations and their implementation using file streams in C++.

Assignment 21:

Object : Write a java program to find the given single digit number using switch case.

ALGORITHM:

1. Start the program.
2. Read a string with inputstreamReader(System.in).
3. convert the string into Integer.parseInt(stdin.readLine());
4. By using switch case (multi way decision statement) when a match is found, that case is executed.
5. Default it is a break statement exit the switch statement.
   Stop the program

PROGRAM:

```
import java.io.*;
class digit
{
```

```
public static void main(String s[]) throws IOException
{
 int n;
 DataInputStream stdin=new DataInputStream(System.in);
 System.out.println("Enter Any positive single digit number :");
 n=Integer.parseInt(stdin.readLine());
 Switch(n)
  {
  case 0:
        System.out.println("Zero");
        break;
  case 1:
        System.out.println("One");
        break;
  case 2:
        System.out.println("Two");
        break;
  case 3:
        System.out.println("Three");
        break;
  case 4:
        System.out.println("Four");
        break;
  case 5:
        System.out.println("Five");
        break;
  case 6:
        System.out.println("Six");



    break;
      case 7:
            System.out.println("Seven");
            break;
      case 8:
            System.out.println("Eight");
            break;
      case 9:
            System.out.println("Nine");
            break;
      default:
             System.out.println("Invalid Number");
            break;
     }
    }
  }
```

OUTPUT:

```
Enter any positive single digit number: 6
Six
Enter any positive single digit number: 5
Five
```

Assignment 22:

Object :- Write a java program to find the given factorial numbers.

ALGORITHM:

1. Start the program. Import the packages.
2. Read a string with inputstreamReader(System.in).
3. convert the string into Integer.parseInt(stdin.readLine());
4. By using for loop rotating the integer value.
5. Repeat enter the value until end of loop.
6. End of class and main method.
   stop the program

PROGRAM:

```
import java.io.*;            //importing  io package
import java.lang.*;         //importing lang package
class Factorial
{
    public static void main(String  args[])   throws IOException
     {
        int i,n,f=1;
        System.out.println("Enter the numbert you want to calculate the factorial");
        BufferedReader stdin=new BufferedReader(new InputStreamReader(System.in));
        n=Integer.parseInt(stdin.readLine());
         for(i=1;i<=n;i++)
          {
             f=f*i;
          }
        System.out.println("The   factorial of " + n + " is " +  f);
     }                                                    //End of main
}                                                         //End of class Factorial
```

//End of class Factorial

OUTPUT:

```
Enter the number for which you want the factorial  3
The factorial of 3 is 6
```

Assignment 23:

Object: To check whether the first number is a multiple of second number.

ALGORITHM:

1. Start the program, import the packages.
2. Create a class and variables with data types.
3. Read a string with inputstreamReader(System.in).
4. convert the string into Integer.parseInt(stdin.readLine());
5. By using ***if…else*** loop rotating the string.
6. Print the concatenation of arrays.

   Stop the program

PROGRAM:

```
import java.io.*;                    //Importing io package
class Multiple
{
  public static void main(String  args[])   throws IOException
  {
   int m,n;
   BufferedReader stdin=new BufferedReader(new InputStreamReader(System.in));
   System.out.println("Enter the first number" );
   m=Integer.parseInt(stdin.readLine());
   System.out.println("Enter the second number ");
   n=Integer.parseInt(stdin.readLine());
   if ( m%n==0)
   {
     System.out.println("The  first number is  the multiple of  second number" );
   }
    else
    {
       System.out.println("The  first number is not  the multiple  of  second number" );
    }


    }              //End of main
}                  //End of class  Multiple
```

OUTPUT:

```
Enter the first number 10
Enter the second number 5
The first number is  the multiple of  second number
Enter the first number 2
Enter the second number 3
The first number is not the multiple  of second number
```

Assignment 24:

Objective : Write a java program to check given numbers in a sorting order.

ALGORITHM:

1. Start the program.
2. Create a class and variables with data types.
3. Read a string with DatainputstreamReader(System.in).
4. convert the string into Integer.parseInt(stdin.readLine());
5. By using ***for*** loop rotating the array.
6. Print the concatenation of arrays.

   Stop the program

PROGRAM:

```
import java.io.*;
class sorting
{
public static void main(String s[]) throws IOException
    {
          int a[]=new int[10];
          int i,j;
          DataInputStream stdin=new DataInputstream(System.in);
          System.out.println("Enter 10 Elements into Array");
          for(i=0;i<10;i++)
               a[i]=Integer.parseInt(stdin.readLine());
          for(i=0;i<9;i++)
            for(j=0;j<9-i;j++)
              {
               if (a[j+1]<a[j])
                 {
                  int temp=a[j+1];
                    a[j+1]=a[j];
                    a[j]=temp;
                 }
              }



          System.out.println("Required Order is ");
          for(i=0;i<10;i++)
            System.out.println(a[i]);
         }
}
```

OUTPUT:

```
Enter 10 elements into Array:10 9 8 7 6 5 4 3 2 1
Required order is:1  2 3 4 5 6  7 8 9 10
```

Assignment 25:

Object : Write a java program to generate the Armstrong number.

ALGORITHM:

1. Start the program.
2. Read a string with DatainputstreamReader(System.in).
3. convert the string into Integer.parseInt(stdin.readLine());
4. sum=sum+r*r*r formula.
5. Using if else statement.
   Stop the program

PROGRAM:

```
import java.io.*;
class armstrong
{
public static void main(String s[]) throws IOException
{
int n,r,temp,sum;
DataInputStream stdin=new DataInputStream(System.in);
System.out.println("Enter any Positive Integer Number :");
n=Integer.parseInt(stdin.readLine());
temp=n;
sum=0;
while(temp>0)
 {
  r=temp % 10;
  sum=sum+r*r*r;
  temp=temp/10;
 }
if (n==sum)
   System.out.println("Given Number is Armstrong Number");
else
   System.out.println("Given Number is not Armstrong Number");
}
}
```

OUTPUT:

```
Enter any positive Integer number:  153
Given number is Armstrong number.
Enter any positive Integer number:  146
Given number is not Armstrong number.
```

Assignment 26:

Objective : Write a java program to generate the quadratic equation.

ALGORITHM:

1. Start the program. Import the packages.
2. Create a class and variables with data types.
3. Declaration of the main class.
4. Read a string with inputstreamReader(System.in).
5. convert the string into Integer.parseInt(stdin.readLine());
6. By using if(d==0) "Roots are Equalent" loop rotating the integer value.
7. if(d>0) "Roots are Real" otherwise ("Roots are Imaginary");
8. Repeats enter the value until end of loop.
9. End of class and main method.
10. Stop the program.

PROGRAM:

```
import java.io.*;
import java.math.*;
class quadratic
{
 public static void main(String s[]) throws IOException
 {
 int a,b,c,d;
 double r1,r2;
 BufferedReader stdin=new BufferedReader(new InputStreamReader(System.in));
 System.out.println("Enter value of a:");
 a=Integer.parseInt(stdin.readLine());
 System.out.println("Enter value of b:");
 b=Integer.parseInt(stdin.readLine());
 System.out.println("Enter value of c:");
 c=Integer.parseInt(stdin.readLine());
 d=b*b-4*a*c;
 if(d==0)
   {
   System.out.println("Roots are Equalent");
   r1=-b/(2*a);
   r2=-b/(2*a);
   System.out.println("Root1 = "+r1);
   System.out.println("Root2 = "+r2);
   }


else if (d>0)
     {
     System.out.println("Roots are Real");
     r1=(-b+Math.sqrt(d))/(2*a);
     r2=(-b-Math.sqrt(d))/(2*a);
     System.out.println("Root1 = "+r1);
```

```
  System.out.println("Root2 = "+r2);
    }
 else
   System.out.println("Roots are Imaginary");
}
```

OUTPUT:

```
Enter  value of a:1
Enter value of b:2
Enter value of c:1
Roots are Equalent
Enter  value of a:2
Enter value of b:3
Enter value of c:2
Roots are Imaginary
```

Assignment 27:

Object -To implement a calculator with its functionality.

ALGORITHM:

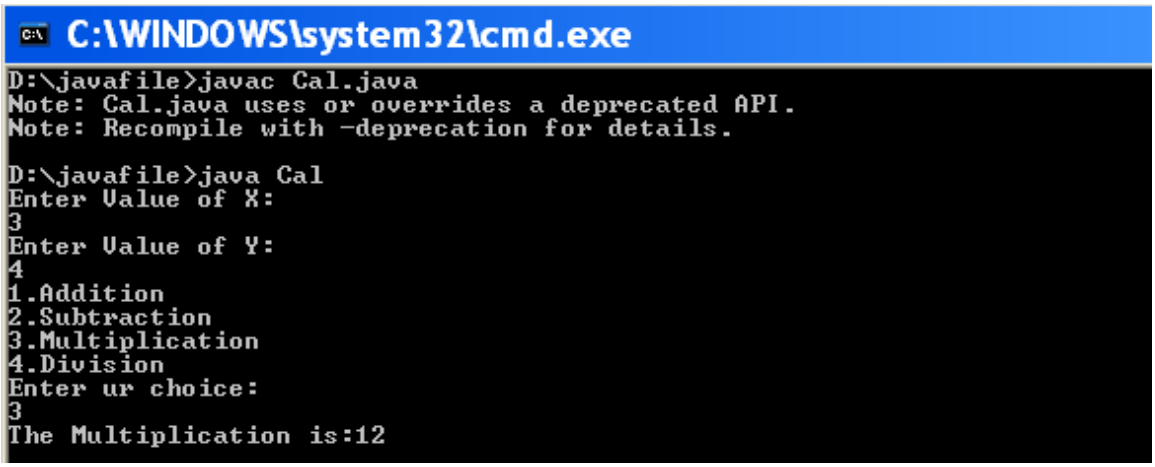
1. Start the program. Import the packages.
2. Create a class and variables with data types.
3. Declaration of the main class.
4. Read a string with inputstreamReader(System.in).
5. convert the string into Integer.parseInt(stdin.readLine());
6. By using *for* loop rotating the integer value.
7. Repeats enter the value until end of loop.
8. End of class and main method.
9. Stop the program.

PROGRAM:

```
import java.io.DataInputStream;
class Cal
{
      public static void main(String args[])
      {
            DataInputStream pt=new
DataInputStream(System.in);
            int x,y,z,ch;
            try
            {
                  System.out.println("Enter Value of X:");
                  x = Integer.parseInt(pt.readLine());
                  System.out.println("Enter Value of Y:");
                  y = Integer.parseInt(pt.readLine());
                  System.out.println("1.Addition \n 2.Subtraction
\n 3. Multiplication \n 4.Division");
                  System.out.println("Enter ur choice:");
                  ch = Integer.parseInt(pt.readLine());
                  switch(ch)


              {
                        case 1:
                              z = x + y;
                              System.out.println("The Addition
 is:"+z);
                              break;
                        case 2:
                              z = x - y;
```

```
System.out.println("The Subtraction is:"+z);
                    break;
               case 3:
                    z = x * y;
                    System.out.println("The Multiplication
is:"+z);
                    break;
               case 4:
                    z = x / y;
                    System.out.println("The Division
is:"+z);
               break;
               default:
                    System.out.println("Sorry Try
Again.....");
                    break;
               }
          }

          catch(Exception e)
          {
          System.out.println("x.getmessage()");
          }
     }

}
```

OUTPUT:



Assignment 28:

Object : To find the largest number of given numbers, by using arrays.

ALGORITHM:

1. Start the program, import the packages.
2. Create a class and variables with data types.
3. Read a string with inputstreamReader(System.in).
4. convert the string into Integer.parseInt(stdin.readLine());
5. By using ***for*** loop rotating the single dimensional arrays value.
6. Repeats enter the value until end of loop.
7. Print the concatenation of string.
8. Stop the program.

PROGRAM:

```
import java.io.*;                    //Importing io package
import java.lang.*;                  //Importing lang package
class Largest
{

   public static void main(String args[])       throws IOException
    {
       int a[]= new int[10];
       int i,j,k;
       System.out.println("Enter the numbers of the array");
       BufferedReader stdin=new BufferedReader(new InputStreamReader(System.in));
        for(i=0;i<10;i++)
        {
           a[i]=Integer.parseInt(stdin.readLine());
        }
        for(i=0;i<10;i++)
        {
           for(j=(i+1);j<10;j++)
            {
               if(a[i]>a[j])
           {
              k=a[i];
                    a[i]=a[j];
                    a[j]=k;
                }               //End of if

              }                 //End of inner for loop
         }                      //End of outer for loop
System.out.println("The first largest number in the array is " + a[9]);
System.out.println("The second largest number in the array is " + a[8]);
  }     //End of main
} //End of class Largest
```

OUTPUT:

```
Enter the numbers of the array 1 2 6 3 78 7 8 45 32 23
The first largest number in the array is 78
The second largest number in the array is 45
Enter the numbers of the array 12 23 34 45 56 67 68 78 89 100
The first largest number in the array is 100
The second largest number in the array is 89
Enter the numbers of the array 11 12 13 14 15 16 17 18 19 20
The first largest number in the array is 20
The second largest number in the array is 19
```

Viva Questions(object oriented programming)

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.

13. What are instance variables?

Instance variables are those which are defined at the class level. Instance variables need not be initialized before using them as they are automatically initialized to their default values.

14. How to define a constant variable in Java?

The variable should be declared as `static` and `final`. So only one copy of the variable exists for all instances of the class and the value can't be changed also.
`static final int PI = 2.14;` is an example for constant.

15. Should a main() method be compulsorily declared in all java classes?

No not required. `main()` method should be defined only if the source class is a java application.

16. What is the return type of the main() method?

`Main()` method doesn't return anything hence declared `void`.

17. Why is the main() method declared static?

`main()` method is called by the JVM even before the instantiation of the class hence it is declared as `static`.

18. What is the arguement of main() method?

`main()` method accepts an array of String object as arguement.

19. Can a main() method be overloaded?

Yes. You can have any number of `main()` methods with different method signature and implementation in the class.

20. Can a main() method be declared final?

Yes. Any inheriting class will not be able to have it's own default `main()` method.

21. Does the order of public and static declaration matter in main() method?

No. It doesn't matter but `void` should always come before `main()`.

22. Can a source file contain more than one class declaration?

Yes a single source file can contain any number of Class declarations but only one of the class can be declared as `public`.

23. What is a package?

Package is a collection of related classes and interfaces. package declaration should be first statement in a java class.

24. Which package is imported by default?

`java.lang package` is imported by default even without a package declaration.

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.