20MCA132

OBJECT ORIENTED PROGRAMMING LAB

CO1

Understand object-oriented concepts and design classes and objects to solve problems

Contents

Preface

This lab manual is prepared primarily for the students pursuing the MCA programme of the APJ Abdul Kalam Technological University. The Curriculum for the programme offers a practical course on object oriented programming using JAVA in the Second Semester with code and name "20MCA132 OBJECT ORIENTED PROGRAMMING LAB". The selection of topics in this manual is guided by the contents of the syllabus for the course and also by the contents in the books cited in the syllabus. The book will also be useful to faculty members who teach the course. This can be used as a reference guide that covers only essential aspects required for each Course Outcomes. The users of these notes are earnestly requested to bring to the notice of the author any errors they find so that a corrected and revised edition can be published at the earliest.

20MCA132	OBJECT ORIENTED PROGRAMMING LAB	CATEGORY	L	T	P	CREDIT
		PRACTICAL	0	1	3	2

Preamble: This course enables the students to understand the concepts of object-oriented programming and to develop skills using these paradigms using Java.

Prerequisite: Knowledge of any programming language preferred.

Course Outcomes: After the completion of the course the student will be able to

CO 1	Understand object-oriented concepts and design classes and objects to solve problems
CO 2	Implement arrays and strings.
CO 3	Implement object-oriented concepts like inheritance, overloading and interfaces
CO 4	Implement packages, exception handling, multithreading and generic programming. Use java.util package and Collection framework
CO 5	Develop applications to handle events using applets
CO 6	Develop applications using files and networking concepts

Mapping of course outcomes with program outcomes

	PO 1	PO 2	PO 3	PO 4	PO 5	PO 6	PO 7	PO 8	PO 9	PO 10	PO 11	PO 12
CO 1	2	2	2	2	3							
CO 2	**3**	2	2		3							
CO 3	**3**	2	2		3							
CO 4	**3**	2	2		3							
CO 5	**3**	3	3		3	2			3		3	
CO 6	**3**	3	3		3	2			3		3	

Assessment Pattern

Bloom's Category	Continuous Assessment Tests		End Semester Examination
	1	2	
Remember(K1)			
Understand(K2)			
Apply(K3)	10	10	10
Analyse(K4)	10	10	10
Evaluate(K5)	10	10	10
Create(K6)	20	20	20

Mark distribution

Total Marks	CIE	ESE	ESE Duration
100	50	50	3 hours

Continuous Internal Evaluation Pattern:

Maximum Marks: 50	
Attendance	15%
Maintenance of daily lab record and GitHub management	20%
Regular class viva	15%
Timely completion of day to day tasks	20%
Tests/Evaluation	30%

End Semester Examination Pattern:

Maximum Marks: 50			
Verification of Daily program record and Git Repository			5 marks
Viva			10 marks
Problem solving (Based on difficulty level, one or more questions may be given)	Flowchart / Algorithm / Structured description of problem to explain how the problem can be solved / Interface Design	15%	35 marks
	Program correctness	50%	
	Code efficiency	15%	
	Formatted output and Pushing to remote Git repository	20%	
Total Marks			50 marks

Course Level Assessment Questions

Course Outcome 1 (CO1):

1. Define a class ‘product’ with data members pcode, pname and price. Create 3 objects of the class and find the product having the lowest price.
2. Read 2 matrices from the console and perform matrix addition.

3. Add complex numbers
4. Read a matrix from the console and check whether it is symmetric or not.
5. Create CPU with attribute price. Create inner class Processor (no. of cores, manufacturer) and static nested class RAM (memory, manufacturer). Create an object of CPU and print information of Processor and RAM.

Course Outcome 2 (CO2)

1. Program to Sort strings
2. Search an element in an array.
3. Perform string manipulations
4. Program to create a class for Employee having attributes eNo, eName eSalary. Read n employ information and Search for an employee given eNo, using the concept of Array of Objects.

Course Outcome 3(CO3):

1. Area of different shapes using overloaded functions
2. Create a class 'Employee' with data members Empid, Name, Salary, Address and constructors to initialize the data members. Create another class 'Teacher' that inherit the properties of class employee and contain its own data members department, Subjects taught and constructors to initialize these data members and also include display function to display all the data members. Use array of objects to display details of N teachers.
3. Create a class 'Person' with data members Name, Gender, Address, Age and a constructor to initialize the data members and another class 'Employee' that inherits the properties of class Person and also contains its own data members like Empid, Company_name, Qualification, Salary and its own constructor. Create another class 'Teacher' that inherits the properties of class Employee and contains its own data members like Subject, Department, Teacherid and also contain constructors and methods to display the data members. Use array of objects to display details of N teachers.
4. Write a program has class Publisher, Book, Literature and Fiction. Read the information and print the details of books from either the category, using inheritance.
5. Create classes Student and Sports. Create another class Result inherited from Student and Sports. Display the academic and sports score of a student.

6. Create an interface having prototypes of functions area() and perimeter(). Create two classes Circle and Rectangle which implements the above interface. Create a menu driven program to find area and perimeter of objects.

7. Prepare bill with the given format using calculate method from interface.

Order No.

Date :

Product Id	Name	Quantity	unit price	Total
101	A	2	25	50
102	B	1	100	100
			Net. Amount	150

Course Outcome 4 (CO4):

1. Create a Graphics package that has classes and interfaces for figures Rectangle, Triangle, Square and Circle. Test the package by finding the area of these figures.
2. Create an Arithmetic package that has classes and interfaces for the 4 basic arithmetic operations. Test the package by implementing all operations on two given numbers
3. Write a user defined exception class to authenticate the user name and password.
4. Find the average of N positive integers, raising a user defined exception for each negative input.
5. Define 2 classes; one for generating multiplication table of 5 and other for displaying first N prime numbers. Implement using threads. (Thread class)
6. Define 2 classes; one for generating Fibonacci numbers and other for displaying even numbers in a given range. Implement using threads. (Runnable Interface)
7. Producer/Consumer using ITC
8. Program to create a generic stack and do the Push and Pop operations.
9. Using generic method perform Bubble sort.
10. Maintain a list of Strings using ArrayList from collection framework, perform built-in operations.
11. Program to remove all the elements from a linked list
12. Program to remove an object from the Stack when the position is passed as parameter
13. Program to demonstrate the creation of queue object using the PriorityQueue class
14. Program to demonstrate the addition and deletion of elements in deque
15. Program to demonstrate the creation of Set object using the LinkedHashset class
16. Write a Java program to compare two hash set

17. Program to demonstrate the working of Map interface by adding, changing and removing elements.
18. Program to Convert HashMap to TreeMap

Course Outcome 5 (CO5):

1. Program to draw Circle, Rectangle, Line in Applet.
2. Program to find maximum of three numbers using AWT.
3. Find the percentage of marks obtained by a student in 5 subjects. Display a happy face if he secures above 50% or a sad face if otherwise.
4. Using 2D graphics commands in an Applet, construct a house. On mouse click event, change the color of the door from blue to red.
5. Implement a simple calculator using AWT components.
6. Develop a program that has a Choice component which contains the names of shapes such as rectangle, triangle, square and circle. Draw the corresponding shapes for given parameters as per user's choice.
7. Develop a program to handle all mouse events and window events
8. Develop a program to handle Key events.

Course Outcome 6 (CO6):

1. Program to list the sub directories and files in a given directory and also search for a file name.
2. Write a program to write to a file, then read from the file and display the contents on the console.
3. Write a program to copy one file to another.
4. Write a program that reads from a file having integers. Copy even numbers and odd numbers to separate files.
5. Client server communication using Socket – TCP/IP
6. Client Server communication using DatagramSocket - UDP

Syllabus:

Classes and Objects, Constructors, Method Overloading, Access Modifiers, Arrays and Strings, Inheritance, Interfaces, Abstract classes, Dynamic Method Dispatch, String, Packages, Introduction to java.util, Collection framework, User defined packages, Exceptions, Multithreading, Applets, Graphics, File, Generic programming, Socket Programming

Reference Books

1. Herbert Schildt, "***Java The Complete Reference***", Seventh Edition, Tata McGraw-Hill Edition

2. C. Thomas Wu, "***An introduction to Object-oriented programming with Java***", Fourth Edition, Tata McGraw-Hill Publishing company Ltd.

3. Cay S. Horstmann and Gary Cornell, "***Core Java: Volume I – Fundamentals***", Eighth Edition, Sun Microsystems Press.

4. K. Arnold and J. Gosling, "***The JAVA programming language***", Third edition, Pearson Education.

5. Paul Deitel and Harvey Deitel, "***Java, How to Program***", Tenth Edition, Pearson Education

6. Rohit Khurana, "***Programming with Java***", Vikas Publishing, 2014.

7. Timothy Budd, "***Understanding Object-oriented programming with Java***", Updated Edition, Pearson Education.

8. Y. Daniel Liang, "***Introduction to Java programming***", Seventh Edition, Pearson Education.

Web Reference

- https://www.hackerrank.com/domains/java
- https://www.geeksforgeeks.org/java-tutorial/
- https://www.w3resource.com/java-tutorial/
- https://www.w3resource.com/java-exercises/
- https://nptel.ac.in/courses/106/105/106105191/
- https://nptel.ac.in/noc/courses/noc20/SEM1/noc20-cs08/
- https://www.coursera.org/learn/object-oriented-java
- https://www.edx.org/course/object-oriented-programming-in-java-2

Course Contents and Lab Schedule

Topic	No. of hours
1. Classes and Objects.	3
2. Constructors, Method Overloading, Access Modifiers	2
3. Arrays and Strings.	4
4. Inner class – static and non-static	2
5. Inheritance, Multiple inheritance - implementation using interfaces	3
6. Method overriding, Abstract classes, Dynamic Method Dispatch	3
7. Interfaces and Packages, StringBuffer class	3
8. Introduction to java.util package – Vector, Scanner, StringTokenizer	2
9. Collection framework – ArrayList, LinkedList, Stack, Queue, Set, Map	3
10. User defined packages	2
11. Exceptions – User defines exceptions	2
12. Multithreading – Thread class	2
13. Inter Thread Communication	2
14. Generic programming	2
15. Applets, Graphics – 2D	3
16. Event handling in Applet	3
17. File	3
18. Socket Programming	3

Chapter 1

Introduction to Java

Java is a programming language and a platform. Java is a high level, robust, object-oriented and secure programming language.

Java was developed by Sun Microsystems (which is now the subsidiary of Oracle) in the year 1995. James Gosling is known as the father of Java. Before Java, its name was Oak. Since Oak was already a registered company, James Gosling and his team renamed Oak as Java.

Any hardware or software environment in which a program runs, is known as a platform. Since Java has a runtime environment (JRE) and API, it is called a platform.

1.1 Java terminologies

Let us get familiarised with some common Java terms.

1.1.1 Java Virtual Machine (JVM)

This is generally referred as JVM. Before, we discuss about JVM lets see the phases of program execution. Phases are as follows: we write the program, then we compile the program and at last we run the program.

1. Writing of the program is of course done by Java programmer.
2. Compilation of program is done by Javac compiler, *Javac* is the primary Java compiler included in Java development kit (JDK). It takes Java program as input and generates Java bytecode as output.
3. In third phase, JVM executes the bytecode generated by compiler. This is called *program run phase.*

So, now that we understood that the primary function of JVM is to execute the bytecode produced by compiler. Each operating system has different JVM, however the output they produce after execution of bytecode is same across all operating systems. That is why we call Java as platform independent language.

1.1.2 bytecode

As discussed above, Javac compiler of JDK compiles the Java source code into bytecode so that it can be executed by JVM. The bytecode is saved in a .class file by compiler.

1.1.3 Java Development Kit(JDK)

As the name suggests this is complete Java development kit that includes JRE (Java Runtime Environment), compilers and various tools like JavaDoc, Java debugger etc. In order to create, compile and run Java program you would need JDK installed on your computer.

1.1.4 Java Runtime Environment(JRE)

JRE is a part of JDK which means that JDK includes JRE. When you have JRE installed on your system, you can run a Java program however you won't be able to compile it. JRE includes JVM, browser plugins and applets support. When you only need to run a Java program on your computer, you would only need JRE.

1.2 Features of JAVA

A list of most important features of Java language is given below.

1. Simple: Java is considered as one of simple language because it does not have complex features like Operator overloading, Multiple inheritance, pointers and Explicit memory allocation.

2. Object-Oriented: Java is an Object Oriented language. Object oriented programming is a way of organizing programs as collection of objects, each of which rrepresents an instance of a class. Basic concepts of OOPs are:

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

3. Portable: As discussed above, Java code that is written on one machine can run on another machine. The platform independent byte code can be carried to any platform for execution that makes Java code portable.

4. Platform independent: Java is a platform independent language. Compiler(Javac) converts source code (.java file) to the byte code(.class file). As mentioned above, JVM executes the bytecode produced by compiler. This byte code can run on any platform such as Windows, Linux, Mac OS etc. Which means a program that is compiled on windows can run on Linux and vice-versa. Each operating system has different JVM, however the output they produce after execution of bytecode is same across all operating systems. That is why we call Java as platform independent language.

5. Secure: We don't have pointers and we cannot access out of bound arrays (you get ArrayIndexOutOfBoundsException if you try to do so) in Java. That's why several security flaws like stack corruption or buffer overflow is impossible to exploit in Java.

6. Robust: Robust means reliable. Java programming language is developed in a way that puts a lot of emphasis on early checking for possible errors. The main features of Java that makes it robust are garbage collection, Exception Handling and memory allocation.

7. Architecture neutral: Java is architecture neutral because there are no implementation dependent features, for example, the size of primitive types is fixed. In C programming, int data type occupies 2 bytes of memory for 32-bit architecture and 4 bytes of memory for 64-bit architecture. However, it occupies 4 bytes of memory for both 32 and 64-bit architectures in Java.

8. High-performance: Java is faster than other traditional interpreted programming languages because Java bytecode is "close" to native code.

9. Multithreading: Java supports multithreading. Multithreading is a Java feature that allows concurrent execution of two or more parts of a program for maximum utilisation of CPU.

10. Distributed: Using Java programming language we can create distributed applications. RMI(Remote Method Invocation) and EJB(Enterprise Java Beans) are used for creating distributed applications in Java. In simple words: The Java programs can be distributed on more than one systems that are connected to each other using internet connection. Objects on one JVM (Java virtual machine) can execute procedures on a remote JVM.

11. Dynamic: Java is a dynamic language. It supports dynamic loading of classes. It means classes are loaded on demand. It also supports functions from its native languages, i.e., C and C++. Java supports dynamic compilation and automatic memory management (garbage collection).

12. Strongly typed: Java is a strongly typed language. The Java compiler enforces type checking of each assignment made in a program at compile time. If the type checking fails, then a compile-time error is issued.

1.3 JAVA IDE

IDE: Integrated Development Environment. There are various free IDEs available for JAVA. A few of them are listed below.

- Eclipse
- NetBeans
- IntelliJ IDEA
- BlueJ
- (Oracle) JDeveloper
- Greenfoot

In the following chapter, we present basic concepts of Java programming, specifically about objects, classes, constructors and so on.

Part I

CO1: Understand object-oriented concepts and design classes and objects to solve problems

Chapter 2

Classes and Objects

Object-Oriented Programming is a methodology or paradigm to design a program using classes and objects and thus, simplifies software development and maintenance.

2.1 Classes

Collection of objects is called class. It is a logical entity, i.e., it doesn't consume any memory space. We create objects from a class. Therefore, a class can also be defined as a blueprint from which you can create an individual object. Multiple objects can be created from a class. Thus, a class is a template for an object, and an object is an instance of a class.

A class is declared using *class* keyword. The data or variables defined within a class are called *instance variables*. Collectively, the methods and variables defined within a class are called *members* of a class. A general form of a class is given below.

```
class classname{
    type instance-variable1;
    type instance-variable2;
    type instance-variable3;
    //...
    type instance-variableN;

    type methodname1(parameter-list){
    //Body of method
    }

    //...
    type methodnameN(parameter-list){
    //Body of method
```

```
    }
}
```

For example, below is a class Car, that has a color, and speed.

```
class Car{
    String color;
    int speed;
}
```

2.2 Objects

Object means a real-world entity such as a pen, chair, table, computer, watch, etc. Technically, any entity that has state and behavior is known as an object. It can be physical or logical.

An Object can also 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.

In lieu of the class Car above, we can consider maruthi, honda, etc., as instances of the class Car.

2.3 Creating objects from classes

As aforementioned, we create (multiple) objects from a class. Every time an object is created, memory is allocated to it. We use ***new*** keyword to create an object of a class.

```
classname objectname = new classname();

Car maruthi = new Car();
```

However, this does not ensure proper initialisation of member data within the class. that is, the color and speed of maruthi (instance created) is unknown. While designing a class, the class designer can define within the class, a special method called "constructor". This *constructor* is automatically invoked whenever an object of the class is created. These constructors can be overloaded to initialise the member data when the object is created.

2.3.1 Constructors

A constructor is invoked to create an object of a class. A constructor function has the same name as the class name [Car() in the above example]. A class can contain more than one constructor. This, thus, facilitates multiple ways of initializing an object. Constructors do not have return types and can

be overloaded. A Java constructor cannot be abstract, static, final, and/or synchronized.

2.3.2 Overloaded constructors

Functions those share same name but has different signatures are said to be overloaded functions.

```
class Car{
    String color;
    int speed;

    //default constructor
    Car(){
    color = "";
    speed = 0;
    }

    //parameterised constructor
    Car(String c, int s){
    this.color = c;
    this.speed = s;
    }
}
```

In the class Car above, we have two overloaded constructors. The number, type and sequence of parameters determine the constructor to be invoked. A *maruthi* car of *red* color with speed *100* can be created as follows:

Method 1 (default constructor):

```
Car maruthi = new Car();
maruthi.color = "red";
maruthi.speed = 100;
```

Method 2 (Overloaded constructor with parameters)

```
Car maruthi = new Car("red",100);
```

Once created, the data members can be accessed using ***dot notation***. A general format is

objectname.instancevariable = variablevalue; E.g., maruthi.color = "red";

However, within the class, we use ***this*** keyword to access it own object. E.g., within the class Car(String c,int s)constructor, we used *this.color* = *c*. The ***this*** reference is created automatically by the compiler. It contains the address of the object through which the function is invoked.

A word of caution: When you assign one object reference to another object reference variable, you are only making a copy of the reference. For instance, consider the following:

```
Car maruthi = new Car();
Car another_maruthi = maruthi;
```

You might think that *maruthi* and *another_maruthi* refer to separate and distinct objects, however, that is wrong. Both, *maruthi* and *another_maruthi* refer to the ***same*** object. Thus, any changes made through *maruthi* and *another_maruthi* will affect the same object!

2.3.3 Accessing data members of a class

Java provides access specifiers to control access to class members. Java offers the designer of the class the flexibility of deciding which member data or methods should be accessible from outside the class, and which should not.

The *public* access specifier

When a class member is preceded by **public**, then that member may be accessed by code outside the class in which it is declared.

The *private* access specifier

When a member of a class is specified as **private** then that member can only be accessed by other members of its class. In other words, a **private** member cannot be accessed by code outside the class in which it is declared.

The *protected* access specifier

The **protected** access specifier applies only when inheritance is involved.

We will explore more on access specifiers later when introducing inheritance and packages.

2.4 Inner Classes

Inner class are defined inside the body of another class (known as outer class). These classes can have access modifier or even can be marked as abstract and final. Inner classes have special relationship with outer class instances. This relationship allows them to have access to outer class members including private members too.

Inner classes can be defined in four different following ways:

1. **Inner class**: An inner class is declared inside the curly braces of another enclosing class. Inner class is coded inside a Top level class as shown below:

```
    //Top level class definition
class MyOuterClassDemo {
   private int myVar= 1;
```

```
    // inner class definition
    class MyInnerClassDemo {
       public void seeOuter () {
          System.out.println("Value of myVar is :" + myVar);
       }
     } // close inner class definition
} // close Top level class definition
```

Inner class acts as a member of the enclosing class and can have any access modifiers: abstract, final, public, protected, private, static. Inner class can access all members of the outer class including those marked private as shown in the above example: inner class is accessing the private variable "myVar" of outer class.

An inner class shares a special relationship with an instance of the enclosing class. An inner class instance can be created only from an outer class instance,i.e., there should be a live instance of the outer class to instantiate an instance of the inner class.

```
class MyOuterClassDemo {
    private int x= 1;
    public void innerInstance()
    {
        MyInnerClassDemo inner = new MyInnerClassDemo();
        inner. seeOuter();
    }
    public static void main(String args[]){
        MyOuterClassDemo obj = new MyOuterClassDemo();
        obj.innerInstance();
    }
    // inner class definition
    class MyInnerClassDemo {
        public void seeOuter () {
           System.out.println("Outer Value of x is :" + x);
        }
    } // close inner class definition
} // close Top level class definition
```

This prints "Outer Value of x is :1".

2. **Method-local inner class**: A method local inner class is defined within a method of the enclosing class. In order to use this inner class, we must instantiate the inner class in the same method, but after the class definition code. Only 'abstract' and 'final' modifiers are allowed. The inner class can use the local variables of the method (in which it is present), only if they are marked final.

```
//Top level class definition
class MyOuterClassDemo {
   private int x= 1;

   public void doThings(){
   //not using final below will cause compilation error.
      final String name ="local variable";
      // inner class defined inside a method of outer class
      class MyInnerClassDemo {
        public void seeOuter () {
           System.out.println("Outer Value of x is :" + x);
           //The below statement causes compilation error,
           //if name is not declared as final.
           System.out.println("Value of name is :" + name);
        } //close inner class method
      } // close inner class definition
   } //close Top level class method
} // close Top level class
```

3. **Anonymous inner class**: It is a type of inner class which has no name and can be instantiated only once. It is usually declared inside a method or a code block. It is accessible only at the point where it is defined. Not that it does not have a constructor and cannot be static. Example is given below.

```
    class Pizza{
   public void eat()
   {
      System.out.println("pizza");
   }
}
class Food {
   /* There is no semicolon(;)
    * semicolon is present at the curly braces of the method end.
    */
   Pizza p = new Pizza(){
      public void eat()
      {
         System.out.println("anonymous pizza");
      }
   };
}
```

4. **Static nested class**: Static nested classes are the inner classes marked with static modifier. A static nested class cannot access non static members of outer class.

```
    class Outer{
   static class Nested{}
}
```

A static nested class can be instantiated as follows:

```
class Outer{// outer class
   static class Nested{}// static nested class
}

class Demo{
   public static void main(string[] args){
      // use both class names
      Outer.Nested n= new Outer.Nested();
   }
}
```

2.5 A Java example

Java program to display "Hello Java":

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

We save the above Java snippet as Simple.java. In Java, or any pure object-oriented language, a *class* is the smallest unit of executable code. This means that every source file must contain only classes, and not a class and a separate *main()* function. main() function is the entry point for execution and hence must be invoked without creating an object of the class where it is defined. Hence, we define main() as a **static** function. A **static** member of a class is independent of instances of the class, and **static** method can be invoked without creating an object of the class. main() is declared as **public** to ensure its visibility outside its class i.e., to the Java Runtime Environment.

Chapter 3

I/O Basics

Java provides strong, flexible support for I/O, although usually they rely upon Java's Abstract Window Toolkit (AWT) or Swing to interact with the user. Java programs perform I/O through streams.

3.1 Streams

A *stream* is an abstraction that either produces or consumes information and is linked to a physical device by the Java I/O system. The same I/O classes can be applied to any kind of device as they abstract the differences between different I/O devices.

Java's stream classes are defined in Java.io package. There are two types of streams:

1. **Byte streams**: handle input and output of bytes, and usually used for reading and writing binary data. Input streams and Output streams are byte streams.

2. **Character streams**: handle input and output of characters, and use **Unicodes**[1]. Reader and Writer streams are character streams.

3.1.1 The predefined streams

The **System** class of Java.lang package contains three predefined stream variables: **in**, **out** and **err**.

- **System.out** refers to standard output stream which is the console.
- **System.in** refers to standard input, which is the keyboard by default.
- **System.err** refers to the standard error stream, which also is the console by default.

[1] Therefore, character streams can be internationalized.

3.2 Reading input from the user

For now we will focus on reading console input. In Java, there are four different ways for reading input from the user in the command line environment(console).

1. **Using Buffered Reader Class**:

   This is the Java classical method to take input. This method is used by wrapping the System.in (standard input stream) in an InputStreamReader which is wrapped in a BufferedReader, we can read input from the user in the command line.

```
// Java program to demonstrate BufferedReader
import Java.io.BufferedReader;
import Java.io.IOException;
import Java.io.InputStreamReader;
public class Test {
    public static void main(String[] args) throws IOException
    {
        // Enter data using BufferReader
        BufferedReader reader = new BufferedReader(
            new InputStreamReader(System.in));

        // Reading data using readLine
        String name = reader.readLine();

        // Printing the read line
        System.out.println(name);
    }
}
```

2. **Using Scanner Class**

   This is probably the most preferred method to take input. The main purpose of the Scanner class is to parse primitive types and strings using regular expressions, however, it is also can be used to read input from the user in the command line.

```
// Java program to demonstrate working of Scanner in Java
import Java.util.Scanner;

class GetInputFromUser {
    public static void main(String args[])
    {
        // Using Scanner for Getting Input from User
        Scanner in = new Scanner(System.in);

        String s = in.nextLine();
        System.out.println("You entered string " + s);
```

```
        int a = in.nextInt();
        System.out.println("You entered integer " + a);

        float b = in.nextFloat();
        System.out.println("You entered float " + b);
    }
}
```

3. **Using Console Class**

   It has been becoming a preferred way for reading user's input from the command line. In addition, it can be used for reading password-like input without echoing the characters entered by the user; the format string syntax can also be used (like System.out.printf()). However, this does not work on IDEs as System.console() may require console.

```
// Java program to demonstrate working of System.console()
// Note that this program does not work on IDEs as
// System.console() may require console
public class Sample {
    public static void main(String[] args)
    {
        // Using Console to input data from user
        String name = System.console().readLine();

        System.out.println("You entered string " + name);
    }
}
```

4. **Using Command line argument**

   Most used user input for competitive coding. The command-line arguments are stored in the String format. The parseInt method of the Integer class converts string argument into Integer. Similarly, for float and others during execution. The usage of args[] comes into existence in this input form. The passing of information takes place during the program run. The command line is given to args[]. These programs have to be run on cmd.

```
// Program to check for command line arguments
class Hello {
    public static void main(String[] args)
    {
        // check if length of args array is
        // greater than 0
        if (args.length > 0) {
            System.out.println(
```

```
                "The command line arguments are:");

            // iterating the args array and printing
            // the command line arguments
            for (String val : args)
                System.out.println(val);
        }
        else
            System.out.println("No command line "
                               + "arguments found.");
    }
}
```

Chapter 4

Course Level Assessment Questions - Course Outcome 1

4.1 Experiment 1: Design and use 'Product' class

Points to recap:

- File naming in Java
- Creating objects from classee - constructors
- Conditional statements including ternary operator

Aim: Define a class 'product' with data members pcode, pname and price. Create 3 objects of the class and find the product having the lowest price.

Algorithm:

1. Create a public class Product with members pname, pcode of String datatype and price of integer datatype.
2. Define both default and paramertized constructors.
3. Define getters and setters for members pname, pcode and price.
4. Define a used-defined method display() to display the product details.
5. End class Product
6. Create public class EXP1
7. Define main(String[] args)

8. Create object p1 using default constructor and objects p2 and p3 using overloaded constructors
9. For object p1 created using default constructor, assign values to p1.pname, p1.pcode, p1.price and call p1.display()
10. Get the product p which has the lowest price among p1, p2 and p3. (Use ternary operator for one line of code!)
11. Call p.display();
12. End main
13. End Class EXP1.

Program:

PRODUCT.JAVA

```
import Java.util.Objects;

public class Product {
    String pname, pcode;
    int price;

    public String getPname() {
        return pname;
    }

    //default constructor
    public Product(){

    }

    // parameterised constructor
    public Product(String pname, String pcode, int price) {
        this.pname = pname;
        this.pcode = pcode;
        this.price = price;
    }

    public void setPname(String pname) {
        this.pname = pname;
    }

    public String getPcode() {
        return pcode;
    }
```

```
    public void setPcode(String pcode) {
        this.pcode = pcode;
    }

    public int getPrice() {
        return price;
    }

    public void setPrice(int price) {
        this.price = price;
    }

    public void display(){
        System.out.println("pcode: " + this.pcode);
        System.out.println("pname: " + this.pname);
        System.out.println("price: " + this.price);
    }
}
```

EXP1.JAVA

```
public class EXP1 {
    public static void main(String[] args){
        //Creating object using no parameters constructor
        Product p1 = new Product();
        p1.pcode = "Car123";
        p1.pname = "Benz";
        p1.price = 10000;
        System.out.println("********Displaying p1**********");
        p1.display();

        //Using parameterised constructors
        Product p2 = new Product("Jaguar","Car426", 25000);
        System.out.println("\n********Displaying p2**********");
        p2.display();

        Product p3 = new Product("Maruthi","Car800",50000);
        System.out.println("\n********Displaying p3**********");
        p3.display();

        Product p = p3.getPrice() < (p1.price < p2.price? p1.price:p2.price)?
                p3:(p1.price < p2.price? p1:p2);

        System.out.println("\n****Displaying product with lowest price******");
        p.display();
    }
```

```
}
```

Expected Output:

```
********Displaying p1**********
pcode: Car123
pname: Benz
price: 10000

********Displaying p2**********
pcode: Car426
pname: Jaguar
price: 25000

********Displaying p3**********
pcode: Car800
pname: Maruthi
price: 50000

********Displaying product with lowest price**********
pcode: Car123
pname: Benz
price: 10000
```

4.2 Experiment 2: Matrix Addition

Points to recap:

- Arrays and multi-dimensional arrays.
- Matrix addition
- Ways to read user input from console

Aim: To read 2 matrices from the console and perform marix addition.

Algorithm:

1. If both matrices are of the same size then only we can add the matrices.
2. Use the double dimensional array to store the matrix elements.
3. Read row number,column number and initialize the double dimensional arrays a[][],b[][],c[][] with same row number,column number.
4. Store the first matrix elements into the two-dimensional array a[][] using two for loops. i indicates row number, j indicates column index.Similarly matrix 2 elements in to b[][].
5. Add the two matrices using for loop

   for i=0 to i¡row
   for j=0 to j¡col
   a[i][j] + b[i][j] and store it in to the matrix res at c[i][j] .

Program:

MATRIXADD.JAVA

```
import Java.util.Scanner;
public class MatrixAdd
{
    public static void main(String[] args)
    {
        int p, q, m, n;
        Scanner s = new Scanner(System.in);
        System.out.print("Enter number of rows in first matrix:");
        p = s.nextInt();
        System.out.print("Enter number of columns in first matrix:");
        q = s.nextInt();
        System.out.print("Enter number of rows in second matrix:");
        m = s.nextInt();
        System.out.print("Enter number of columns in second matrix:");
        n = s.nextInt();
```

```
if (p == m && q == n)
{
    int a[][] = new int[p][q];
    int b[][] = new int[m][n];
    int c[][] = new int[m][n];
    System.out.println("Enter all the elements of first matrix:");
    for (int i = 0; i < p; i++)
        for (int j = 0; j < q; j++)
            a[i][j] = s.nextInt();

    System.out.println("Enter all the elements of second matrix:");
    for (int i = 0; i < m; i++)
        for (int j = 0; j < n; j++)
            b[i][j] = s.nextInt();

    System.out.println("First Matrix:");
    for (int i = 0; i < p; i++)
    {
        for (int j = 0; j < q; j++)
            System.out.print(a[i][j]+" ");
        System.out.println("");
    }

    System.out.println("Second Matrix:");
    for (int i = 0; i < m; i++)
    {
        for (int j = 0; j < n; j++)
        {
            System.out.print(b[i][j]+" ");
        }
        System.out.println("");
    }

    //do the sum
    for (int i = 0; i < p; i++)
        for (int j = 0; j < n; j++)
            for (int k = 0; k < q; k++)
                c[i][j] = a[i][j] + b[i][j];

    System.out.println("Matrix after addition:");
    for (int i = 0; i < p; i++)
    {
        for (int j = 0; j < n; j++)
            System.out.print(c[i][j]+" ");
        System.out.println("");
    }
```

```
        }
        else
        {
            System.out.println("Addition would not be possible");
        }
    }
}
```

Expected Output:

```
Enter number of rows in first matrix:3
Enter number of columns in first matrix:3
Enter number of rows in second matrix:3
Enter number of columns in second matrix:3
Enter all the elements of first matrix:
1 2 3
4 5 6
7 8 9
Enter all the elements of second matrix:
2 3 5
4 8 6
7 4 0
First Matrix:
1 2 3
4 5 6
7 8 9
Second Matrix:
2 3 5
4 8 6
7 4 0
Matrix after addition:
3 5 8
8 13 12
14 12 9
```

4.3 Experiment 3: Add Complex Numbers

Points to recap:

- Member data
- Member methods
- Adding complex numbers of form a + bi

Aim: To add two complex numbers.
Algorithm:

1. Define class ComplexNumber with members real, img of double datatype for the real and imaginary part of the complex number.
2. Define a constructor to initialise real and img.
3. Create a temporary complex number to hold the sum of two numbers
4. Do the summation and return the value.

Program

```
public class ComplexNumber {
    //for real and imaginary parts of complex numbers
    double real, img;

    //constructor to initialize the complex number
    ComplexNumber(double r, double i){
        this.real = r;
        this.img = i;
    }

    public static ComplexNumber sum(ComplexNumber c1, ComplexNumber c2)
    {
        //creating a temporary complex number to hold the sum
        ComplexNumber temp = new ComplexNumber(0, 0);

        temp.real = c1.real + c2.real;
        temp.img = c1.img + c2.img;
        return temp;
    }
    public static void main(String args[]) {
        ComplexNumber c1 = new ComplexNumber(5.5, 4);
        ComplexNumber c2 = new ComplexNumber(1.2, 3.5);
        ComplexNumber temp = sum(c1, c2);
        System.out.printf("Sum is: "+ temp.real+" + "+ temp.img +"i");
    }
}
```

Expected Output

```
Sum is: 6.7 + 7.5i
```

It left to the reader to try console input and formatting output.

4.4 Experiment 4: Symmetric matrix

Points to recap:

- Arrays and multi-dimensional arrays
- Properties of a Symmetric matrix
- Ways to read input from console

Aim:To read a matrix from the console and check whether it is symmetric or not.

Algorithm:

1. Read the no of rows and columns into integer variables rows and cols respectively.
2. Check IF(rows != cols), then print given matrx is not symmetric and End. Otherwise, continue.
3. Declare and initialse an integer array matrix[rows][cols].
4. Read the elements of the matrix[i][j].
5. Display the input matrix[i][j].
6. Check if the matrix not equal to transpose matrix i.e., if(matrix[i][j] != matrix[j][i]) Print not symmetric and End.
7. Print matrix is symmetric accordingly.

Program

```
import Java.util.Scanner;

public class SymmetricMatrixProgram
{
    public static void main(String[] args)
    {
        Scanner sc = new Scanner(System.in);

        System.out.println("Enter the no. of rows : ");
        int rows = sc.nextInt();

        System.out.println("Enter the no. of columns : ");
        int cols = sc.nextInt();

        int matrix[][] = new int[rows][cols];
        System.out.println("Enter the elements :");
        for (int i = 0; i < rows; i++)
            for (int j = 0; j < cols; j++)
```

```
                matrix[i][j] = sc.nextInt();

        sc.close();
        System.out.println("Printing the input matrix :");
        for (int i = 0; i < rows; i++)
        {
            for (int j = 0; j < cols; j++)
                System.out.print(matrix[i][j]+"\t");
            System.out.println();
        }

        //Checking the input matrix for symmetric
        if(rows != cols)
            System.out.println("The given matrix is not a square matrix.");
        else
        {
            boolean symmetric = true;
            for (int i = 0; i < rows; i++)
                for (int j = 0; j < cols; j++)
                    if(matrix[i][j] != matrix[j][i])
                    {
                        symmetric = false;
                        break;
                    }

            if(symmetric)
            {
                System.out.println("The given matrix is symmetric...");
            }
            else
            {
                System.out.println("The given matrix is not symmetric...");
            }
        }
    }
}
```

Expected Output

```
Enter the no. of rows :
3
Enter the no. of columns :
3
Enter the elements :
2 4 4
4 8 8
4 8 8
```

```
Printing the input matrix :
2       4       4
4       8       8
4       8       8
The given matrix is symmetric...
```

4.5 Experiment 5: CPU and its details - Using Inner classes

Points to recap:

- Classes and inner classes
- Instantiating objects of inner classes.

Aim:To create CPU with attribute price. Create inner class Processor (no of cores, manufacturer) and static nested class RAM (memory, manufacturer). Create an object of CPU and print information of Processor and RAM.

Algorithm:

1. Define class CPU with member price of double datatype.
2. Define an inner class Processor with members cores (double) and manufacturer (String).
3. Define another nested protected class RAM with members memory and manufacturer of double and String data types respectively.
4. Define a public class CPUDetails
5. Create object of Outer class CPU i.e., CPU cpu = new CPU();
6. Create an object of inner class Processor using outer class CPU.Processor processor = cpu.new Processor();
7. Create an object of inner class RAM using outer class CPU CPU.RAM ram = cpu.new RAM();
8. Print Processor cache= processor.getCache();
9. Print Ram Clock speed = ram.getClockSpeed();
10. End class CPUDetails.

Program

```
class CPU {
    double price;
    // nested class
    class Processor{
        // members of nested class
        double cores;
        String manufacturer;

        double getCache(){
            return 4.3;
```

```
        }
    }

    // nested protected class
    protected class RAM{
        // members of protected nested class
        double memory;
        String manufacturer;

        double getClockSpeed(){
            return 5.5;
        }
    }
}

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

        // create object of Outer class CPU
        CPU cpu = new CPU();

        // create an object of inner class Processor using outer class
        CPU.Processor processor = cpu.new Processor();

        // create an object of inner class RAM using outer class CPU
        CPU.RAM ram = cpu.new RAM();
        System.out.println("Processor Cache = " + processor.getCache());
        System.out.println("Ram Clock speed = " + ram.getClockSpeed());
    }
}
```

Expected Output

```
Processor Cache = 4.3
RAM Clock speed = 5.5
```