B.M.S. COLLEGE OF ENGINEERING

(An Autonomous Institute, Affiliated to VTU, Belagavi)

DEPARTMENT OF MACHINE LEARNING

JAVA PROGRAMMING CONCEPTS

(Course Code: 24AM4AEJPC)

SOLUTION MANUAL

Lab In-charge

Dr. Sandeep Varma N

Academic Year: 2024-25

B.M.S. COLLEGE OF ENGINEERING

(Autonomous Institute, Affiliated to VTU)

VISION

Promoting Prosperity of mankind by augmenting Human Resource Capital through Quality Technical Education & Training.

MISSION

Accomplish Excellence in the field of Technical Education through Education, Research and Service needs of society.

DEPARTMENT OF MACHINE LEARNING

VISION

To achieve excellent standards of quality education in the field of Artificial Intelligence and Machine Learning.

MISSION

- ✓ To nurture the students with strong fundamentals for a successful carrier in the field of artificial intelligence and machine learning.
- ✓ To motivate the students for post-graduation and research.
- ✓ To create impact in the society with continuous research and innovations.

PREFACE

This laboratory manual is prepared by the Department of Machine Learning Introduction to Artificial Intelligence (22AM4PCIAI). This lab manual can be used as instructional book for students, staff and instructors to assist in performing and understanding the programs in Java. In this manual, the programs are as per syllabus prescribed.

INSTRUCTIONS TO THE STUDENTS

Do's

- ✓ Learn the topics taught in the instruction class and come well prepared to the laboratory session.
- ✓ Update observation & record regularly and get it evaluated by the respective faculty.
- ✓ Practice additional concepts taught in the instruction class in every lab.
- ✓ Be obedient and disciplined during the stay in campus.
- ✓ Maintain cleanliness inside the laboratory.
- ✓ Damages observed in the laboratory to be informed to the concerned staff immediately.

Don'ts

- × Usage of cell phones or any other electronic gadgets inside the laboratory.
- × Eat or drink in the laboratory.
- × Damage the department belongings.
- × Meddle with the software programs that are harmful to the laboratory systems.

SYLLABUS

Course Title	JAVA PROGRAMMING CONCEPTS				
Course Code	24AM4AEJPC	Credits	1	L-T-P	0-0-1
CIE	50 Marks	SEE	100 Marks (50% Weightage)		
Contact Hrs./Week	2	Total Lab Hrs.			12

Sl. No	Topics
1.	Overview of Java Development Kit, Java Virtual Machine, Java Syntax, Execution environment and Primitive Datatypes.
2.	Demonstration and usage of Wrapper Classes, Boxing and Unboxing mechanisms.
3.	Implement Program on Strings, StringBuilder and StringBuffer
4.	Enumerations, Arrays
5.	Vector
6.	Generics
7.	Collections Framework
8.	Exception Handling
9.	Interface
10.	Java Packages

COURSE OUTCOMES	
CO1	Apply the core concepts of Java to build console-based applications.
CO2	Analyze the need of Java collection framework to handle group of objects.
CO3	Design and implement robust and modular Java applications using modern integrated tools.

INDEX

		the inventory, remove products from the inventory and display all products in the inventory.	
6	**6**	Write a Java program to implement a dynamic, growable queue using generics. The queue should be able to store elements of any data type and dynamically resize itself as needed to accommodate more elements.	
7	**7**	Consider a small hospital with a pharmacy that needs a simple inventory management system to keep track of its medications and medical supplies. Each item in the inventory has a unique ID, name, quantity in stock, and price. The hospital wants functionalities to add new items, update existing items, remove items, display the list of all items, and search for a specific item by its ID.	
8	**8**	Write a JAVA program which has Class called Account that creates account with Rs500 minimum balance, a deposit() method to deposit amount, a withdraw() method to withdraw amount and also throws LessBalanceException if an account holder tries to withdraw money which makes the balance become less than Rs500. i. A Class called LessBalanceException which returns the statement that says withdraw amount (Rs) is not valid. ii. A Class which creates 2 accounts, both account deposit money and one account tries to withdraw more money which generates a LessBalanceException take appropriate action for the same.	
9	**9**	Write a JAVA program which has i. An Interface class for Stack Operations ii. A Class that implements the Stack Interface and creates a fixed length Stack. iii. A Class that implements the Stack Interface and creates a Dynamic length Stack. iv. A Class that uses both the above Stacks through Interface reference and does the Stack operations that demonstrates the runtime binding.	
10	**10**	Define one class A in package apack. In class A, four variables are defined of access modifiers default, protected, private and public. Define class B in package bpack which extends A and write display() method which access variables of class A. Define class C in package cpack which has one method display() in that create one object of class A and display its variables. Define class ProtectedDemo in package dpack which contains the main () method. Create objects of class B and C and call display method for both these objects.	

		Analyze the program by interpreting the access modifiers and provide valid conclusion.	

1a. Design a membership management system for a fitness center, where the system should allow gym administrators to store information about gym members- name, age, membership status(true/false), and membership duration (in months). Calculate the membership fee based on the membership duration using a predefined fee. Display detailed information of the member.

```
import java.util.Scanner;
public class GymMembershipManager {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        // Prompt user to enter member information
        System.out.println("Welcome to the Gym Membership Management System!");
        System.out.print("Enter member name: ");
        String name = scanner.nextLine();
        System.out.print("Enter member age: ");
        int age = scanner.nextInt();
        System.out.print("Is the membership active? (true/false): ");
        boolean isActive = scanner.nextBoolean();
        System.out.print("Enter membership duration (in months): ");
        int duration = scanner.nextInt();

        // Calculate membership fee (assuming $50 per month)
        double membershipFee = duration * 50.0;

        // Display member details
        System.out.println("\nMember Details:");
        System.out.println("Name: " + name);
```

```
        System.out.println("Age: " + age);
        System.out.println("Membership Status: " + (isActive ? "Active" : "Inactive"));
        System.out.println("Membership Duration: " + duration + " months");
        System.out.println("Membership Fee: $" + membershipFee);
        scanner.close();
    }
}
```

1b. Design JAVA program to print a chessboard pattern.

```
public class ChessboardPattern {
    public static void main(String[] args) {
        int size = 8; // Size of the chessboard
        // Loop through each row
        for (int i = 0; i < size; i++) {
            // Loop through each column
            for (int j = 0; j < size; j++) {
                // Check if the sum of row and column indices is even
                if ((i + j) % 2 == 0) {
                    // Print black square for even sum
                    System.out.print("**");
                } else {
                    // Print white square for odd sum
                    System.out.print("  ");
                }
            }
            // Move to the next line after printing each row
            System.out.println();
```

```
        }
    }
}
```

Primitive Datatypes --→ Wrapper class

Byte ---→ Byte

Short ---→ Short

int -→ Integer

float---→ Float

double --→ Double

char ---→ Character

2 a . Demonstrate boxing of primitive data types into their corresponding wrapper class objects and unbox them back to their primitive forms

```
public class boxing {
    public static void main(String[] args) {
                    Integer boxedInt = Integer.valueOf(10);
            // Double boxing
        Double boxedDouble = Double.valueOf(20.5);
            // Character boxing
        Character boxedChar = Character.valueOf('A');
            // Boolean boxing
```

```
Boolean boxedBoolean = Boolean.valueOf(true);
// Float boxing
Float boxedFloat = Float.valueOf(30.5f);
// Unboxing: Converting wrapper class objects back to their primitive data types
// Integer unboxing
int unboxedInt = boxedInt.intValue();
// Double unboxing
double unboxedDouble = boxedDouble.doubleValue();


// Character unboxing
char unboxedChar = boxedChar.charValue();
// Boolean unboxing
boolean unboxedBoolean = boxedBoolean.booleanValue();
// Float unboxing
float unboxedFloat = boxedFloat.floatValue();
// Display results
System.out.println("Boxed Integer: " + boxedInt);
System.out.println("Unboxed Integer: " + unboxedInt);
System.out.println("Boxed Double: " + boxedDouble);
System.out.println("Unboxed Double: " + unboxedDouble);
System.out.println("Boxed Character: " + boxedChar);
System.out.println("Unboxed Character: " + unboxedChar);
System.out.println("Boxed Boolean: " + boxedBoolean);
System.out.println("Unboxed Boolean: " + unboxedBoolean);
System.out.println("Boxed Float: " + boxedFloat);
```

```
        System.out.println("Unboxed Float: " + unboxedFloat);
    }
}
```

3a. Develop a text processing tool for a language learning platform. The tool needs to compare strings, concatenate strings, and create copies of strings for various languages. Implement appropriate Java functionalities.

```
public class String_Functions {
    // Function to compare two strings
   public static boolean compareStrings(String str1, String str2) {
      return str1.equals(str2);
   }

   // Function to copy a string
   public static String copyString(String original) {
      return new String(original);
   }

   // Function to concatenate two strings
   public static String concatenateStrings(String str1, String str2) {
      return str1 + str2;
   }
   public static void main(String[] args) {


      // Test compareStrings function
      String str1 = "Hello";
      String str2 = "hello";
      System.out.println("String comparison:");
      System.out.println("Are the strings equal? " + compareStrings(str1, str2));

      // Test copyString function
      String original = "Copy me!";
      String copied = copyString(original);
```

```
        System.out.println("\nString copy:");
        System.out.println("Original string: " + original);
        System.out.println("Copied string: " + copied);

        // Test concatenateStrings function
        String part1 = "Hello, ";
        String part2 = "world!";
        String concatenated = concatenateStrings(part1, part2);
        System.out.println("\nString concatenation:");
        System.out.println("Concatenated string: " + concatenated);
    }
    }
```

3b. Write a program in Java for String handling which performs the following:

i. Checks the capacity of String Buffer objects.

ii. Reverses the contents of string given on console and converts the resultant string in upper case.

iii. Reads a string from console and appends it to the resultant string.

```
import java.util.Scanner;

public class StringBufferHandling {
    public static void main(String[] args) {
        // Task i: Checking capacity of StringBuffer objects
        StringBuffer sb1 = new StringBuffer();
        System.out.println("Capacity of StringBuffer object sb1: " + sb1.capacity());

        StringBuffer sb2 = new StringBuffer("Hello");
        System.out.println("Capacity of StringBuffer object sb2: " + sb2.capacity());

        // Task ii: Reversing the contents of string and converting to uppercase
        Scanner scanner = new Scanner(System.in);
```

```
        System.out.print("Enter a string: ");
        String inputString = scanner.nextLine();

        StringBuffer reversedBuffer = new StringBuffer(inputString);
        reversedBuffer.reverse();
        String reversedUpperCase = reversedBuffer.toString().toUpperCase();
        System.out.println("Reversed string in uppercase: " + reversedUpperCase);

        // Task iii: Reading a string from console and appending it
        System.out.print("Enter a string to append: ");
        String appendString = scanner.nextLine();
        reversedBuffer.append(appendString);
        System.out.println("String after appending: " + reversedBuffer);

        scanner.close();
    }
}
```

4 a . Create an enumeration Day of Week with seven values SUNDAY through SATURDAY. Add a method Workday() to the DayofWeek class that returns true if the value on which it is called is MONDAY through FRIDAY.

```
import java.util.Scanner;
public class Enumeration {
    public enum DayOfWeek {
    SUNDAY,
    MONDAY,
    TUESDAY,
    WEDNESDAY,
    THURSDAY,
    FRIDAY,
```

```
    SATURDAY;

    // Method to check if the day is a workday (Monday to Friday)
    public boolean isWorkday() {
        return (this != SATURDAY && this != SUNDAY);
    }
}
    public static void main(String[] args) {
        // Check if Monday is a workday
        System.out.println("Is Monday a workday? " + DayOfWeek.MONDAY.isWorkday());

        // Check if Saturday is a workday
        System.out.println("Is Saturday a workday? " + DayOfWeek.SATURDAY.isWorkday());
    }
}
```

4b. Design a JAVA program to help a teacher track the exam scores of five students in a class. The teacher wants to input the exam scores of each student and then view statistics such as the sum of all scores and the highest score attained.

```
import java.util.Scanner;

public class ExamScoreTracker {
    public static void main(String[] args) {
        final int NUM_STUDENTS = 5;
        int[] examScores = new int[NUM_STUDENTS];

        // Input exam scores for each student
        Scanner scanner = new Scanner(System.in);
        System.out.println("Enter exam scores for each student:");
        for (int i = 0; i < NUM_STUDENTS; i++) {
            System.out.print("Enter score for student " + (i + 1) + ": ");
            examScores[i] = scanner.nextInt();
```

```
        }

        // Calculate sum of scores and highest score
        int sum = 0;
        int highestScore = examScores[0];
        for (int score : examScores) {
            sum += score;
            if (score > highestScore) {
                highestScore = score;
            }
        }

        // Display statistics
        System.out.println("\nExam score statistics:");
        System.out.println("Sum of all scores: " + sum);
        System.out.println("Highest score attained: " + highestScore);
    }
}
```

5. Develop an online shopping platform to manage a dynamic list of products available for sale.

i. Implement a program called OnlineShoppingManager that uses a Vector to store product objects. Each product should have attributes for name, price and category.

ii. Design methods in the OnlineShoppingManager class to add products to the inventory, remove products from the inventory and display all products in the inventory.

```
import java.util.Vector;
class Product {
    String name;
    double price;
    String category;
```

```
    public Product(String name, double price, String category) {
      this.name = name;
      this.price = price;
      this.category = category;

    }
}

public class OnlineShoppingManager {
    private Vector<Product> inventory;

    public OnlineShoppingManager() {
        inventory = new Vector<>();
    }

    public void addProduct(String name, double price, String category) {
        Product product = new Product(name, price, category);
        inventory.add(product);
    }

    public void removeProduct(String name) {
        for (int i = 0; i < inventory.size(); i++) {
            if (inventory.get(i).name.equalsIgnoreCase(name)) {
                inventory.remove(i);
                break;
            }
        }
    }

    public void displayInventory() {
        System.out.println("Inventory:");
        for (Product product : inventory) {
```

```
            System.out.println("Name: " + product.name + ", Price: " + product.price + ", Category: "
+ product.category );
        }
    }

    public static void main(String[] args) {
        OnlineShoppingManager manager = new OnlineShoppingManager();

        // Add some products to the inventory
        manager.addProduct("Laptop", 40000, "Electronics");
        manager.addProduct("T-shirt", 550, "Clothing");
        manager.addProduct("Book", 920, "Books");

        // Display all products in the inventory
        manager.displayInventory();

        // Remove a product
        manager.removeProduct("Book");

        // Display updated inventory
        manager.displayInventory();

        manager.removeProduct("Laptop");

        // Display updated inventory
        manager.displayInventory();
    }
}
```

6. Write a Java program to implement a dynamic, growable queue using generics. The queue should be able to store elements of any data type and dynamically resize itself as needed to accommodate more elements.

```
import java.util.Arrays;
public class GQueue<T> {
    T[] que;
  static int size;
   public GQueue() {
      que = (T[]) new Object[2];
      size = 0;
   }
   public void enqueue(T element) {
      if (size == que.length) {
         int newCapacity = que.length * 2;
      que = Arrays.copyOf(que, newCapacity);
      }
      que[size++] = element;
   }

   public T dequeue() {
      if (size==0) {
         System.out.println("Queue is empty");
      }
      T del_element = que[0];
      System.arraycopy(que, 1, que, 0, size - 1);
      que[--size] = null;
      return del_element;
   }
 public void display() {
      for (T element : que) {
         System.out.println(element);
      }
```

```
  }
    public static void main(String[] args) {
     // Create a GrowableQueue of integers
     GQueue<Integer> integerQueue = new GQueue<>();
     // Enqueue elements
     integerQueue.enqueue(10);
     integerQueue.enqueue(20);
     integerQueue.enqueue(30);

     System.out.println("ELements in the queue are:");
     integerQueue.display();
     // Dequeue an element
     int dequeuedElement = integerQueue.dequeue();
     System.out.println("Dequeued element: " + dequeuedElement);

     // Display size of the queue
     System.out.println("Queue size: " +(++size));

  }
}
```

7. Consider a small hospital with a pharmacy that needs a simple inventory management system to keep track of its medications and medical supplies. Each item in the inventory has a unique ID, name, quantity in stock, and price. The hospital wants functionalities to add new items, update existing items, remove items, display the list of all items, and search for a specific item by its ID.

```
import java.util.ArrayList;

class Pharmacy {
  public final int id;
  public final String name;
```

```
    public int quantity;
    public double price;

    public Pharmacy(int id, String name, int quantity, double price) {
        this.id = id;
        this.name = name;
        this.quantity = quantity;
        this.price = price;
    }
}

public class PharmacyManagementSystem {
    private ArrayList<Pharmacy> inventory = new ArrayList<>();

     // Add new item to inventory
 public void addItem(Pharmacy item) {
        inventory.add(item);
    }

    // Update existing item in inventory
    public void updateItem(int id, int quantity, double price) {
        for (Pharmacy item : inventory) {
            if (item.id == id) {
                item.quantity = quantity;
                item.price = price;
                break;
            }
        }
    }
    // Remove item from inventory
    public void removeItem(int id) {
        inventory.removeIf(item -> item.id == id);
```

```
    }

    // Display list of all items
    public void displayInventory() {
        System.out.println("Inventory:");
        for (Pharmacy item : inventory) {
            System.out.println(item.id + " | " + item.name + " | Quantity: " + item.quantity + " | Price:
" + item.price);
        }
    }

    // Search for item by ID
    public Pharmacy searchItemByID(int id) {
        for (Pharmacy item : inventory) {
            if (item.id == id) {
                return item;
            }
        }
        return null; // Item not found
    }

    public static void main(String[] args) {
        PharmacyManagementSystem ims = new PharmacyManagementSystem();

        // Adding sample items
        ims.addItem(new Pharmacy(1, "Crocin", 100, 2.5));
        ims.addItem(new Pharmacy(2, "Calpol", 50, 3.0));
        ims.addItem(new Pharmacy(3, "Crickmol", 200, 1.0));

        // Displaying inventory
        ims.displayInventory();
```

```
        // Updating an item
        ims.updateItem(1, 90, 2.0);

        // Displaying inventory after update
        ims.displayInventory();

        // Searching for an item
        System.out.println("Searching for item with ID 2:");
        Pharmacy foundItem = ims.searchItemByID(2);
        if (foundItem != null) {
            System.out.println("Item found: " + foundItem.name);
        } else {
            System.out.println("Item not found.");
        }

        // Removing an item
        ims.removeItem(2);

        // Displaying inventory after removal
        ims.displayInventory();
    }
}
```

8. Write a JAVA program which has Class called Account that creates account with Rs500 minimum balance, a deposit() method to deposit amount, a withdraw() method to withdraw amount and also throws LessBalanceException if an account holder tries to withdraw money which makes the balance become less than Rs500.

i. A Class called LessBalanceException which returns the statement that says withdraw amount (Rs) is not valid.

ii. A Class which creates 2 accounts, both account deposit money and one account tries to withdraw more money which generates a LessBalanceException take appropriate action for the same.

```
class LessBalanceException extends Exception {
   public LessBalanceException(double amount) {
      super("Withdraw amount (" + amount + " Rs) is not possible. ");
   }
}

class Account {
    double balance;
   static final double MIN_BALANCE = 500;

   public Account() {
      balance = MIN_BALANCE;
   }

   public void deposit(double amount) {
      balance += amount;
      System.out.println("Deposited " + amount + " Rs. New balance: " + balance + "
Rs");
   }

   public void withdraw(double amount) throws LessBalanceException {
      if (balance - amount < MIN_BALANCE) {
         throw new LessBalanceException(amount);
      }
      balance -= amount;
      System.out.println("Withdrawn " + amount + " Rs. New balance: " + balance + "
Rs");
   }

   public double getBalance() {
      return balance;
   }
```

```
}

public class TestAccount {
   public static void main(String[] args) {
      Account account1 = new Account();
      Account account2 = new Account();

      // Deposit money into both accounts
      account1.deposit(1000);
      account2.deposit(700);

      // Try to withdraw from account1
      try {
         account1.withdraw(1600);
      } catch (LessBalanceException e) {
         System.out.println("LessBalanceException: " + e.getMessage());
      }

      // Try to withdraw from account2
      try {
         account2.withdraw(200);
      } catch (LessBalanceException e) {
         System.out.println("LessBalanceException: " + e.getMessage());
      }
   }
}
```

Write a JAVA program which has
i. An Interface class for Stack Operations
ii. A Class that implements the Stack Interface and creates a fixed length Stack.
iii. A Class that implements the Stack Interface and creates a Dynamic length Stack.
iv. A Class that uses both the above Stacks through Interface reference and does the Stack operations that demonstrates the runtime binding.

i. An Interface class for Stack Operations

```
interface StackOperations {
    void push(int item);
    int pop();
    boolean isEmpty();
    boolean isFull();
}
```

ii. A Class that implements the Stack Interface and creates a fixed length Stack.

```
// Class that implements the Stack Interface and creates a fixed-length Stack
class FixedLengthStack implements StackOperations {
    private int[] stack;
    private int top;
    private int capacity;

    public FixedLengthStack(int size) {
        stack = new int[size];
        capacity = size;
        top = -1;
    }

    public void push(int item) {
        if (isFull()) {
            System.out.println("Stack Overflow");
            return;
        }
        stack[++top] = item;
    }

    public int pop() {
        if (isEmpty()) {
            System.out.println("Stack Underflow");
            return -1;
        }
        return stack[top--];
```

```
        }

        public boolean isEmpty() {
            return top == -1;
        }

        public boolean isFull() {
            return top == capacity - 1;
        }
}
```

iii. A Class that implements the Stack Interface and creates a Dynamic length Stack.

```
class DynamicLengthStack implements StackOperations {
    private int[] stack;
    private int top;
    private int capacity;

    public DynamicLengthStack(int size) {
        stack = new int[size];
        capacity = size;
        top = -1;
    }
    public void push(int item) {
        if (isFull()) {
            resizeStack();
        }
        stack[++top] = item;
        System.out.println(“Item pushed : “+item);
    }

    public int pop() {
        if (isEmpty()) {
            System.out.println("Stack Underflow");
            return -1;
        }
        return stack[top--];
    }

    public boolean isEmpty() {
        return top == -1;
    }

    public boolean isFull() {
```

```
        return top == capacity - 1;
    }

    private void resizeStack() {
        int[] newStack = new int[capacity * 2];
        System.arraycopy(stack, 0, newStack, 0, capacity);
        capacity *= 2;
       System.out.println("Stach size resized to : " +capacity);
        stack = newStack;
    }
}
```

iv. A Class that uses both the above Stacks through Interface reference and does the Stack operations that demonstrates the runtime binding.

```
public class StackDemo {
    public static void main(String[] args) {
        StackOperations fixedStack = new FixedLengthStack(5);
        StackOperations dynamicStack = new DynamicLengthStack(5);

        System.out.println("Using Fixed Length Stack:");
        testStack(fixedStack);

        System.out.println("\nUsing Dynamic Length Stack:");
        testStack(dynamicStack);
    }

    public static void testStack(StackOperations stack) {
        stack.push(1);
        stack.push(2);
        stack.push(3);
        stack.push(4);
        stack.push(5);
        stack.push(6); // This will show overflow for fixed stack and resize for dynamic stack

        System.out.println("Popping elements from stack:");
        while (!stack.isEmpty()) {
            System.out.println(stack.pop());
        }
    }
}
```

- **Define one class A in package apack. In class A, four variables are defined of access modifiers default, protected, private and public.**
- **Define class B in package bpack which extends A and write display() method which access variables of class A.**
- **Define class C in package cpack which has one method display() in that create one object of class A and display its variables.**
- **Define class ProtectedDemo in package dpack which contains the main () method.**
- **Create objects of class B and C and call display method for both these objects. Analyze the program by interpreting the access modifiers and provide valid conclusion.**

Define one class A in package apack. In class A, four variables are defined of access modifiers default, protected, private and public.

```
package apack;
public class A{
int defaultvar = 10;
protected int protectedVar = 20;
private int privateVar = 30;
public int publicVar =40;
public A(){
   System.out.println("Inside Constructor of class A");
}
}
```

Define class B in package bpack which extends A and write display() method which access variables of class A.

```
package bpack;
import apack.A;
public class B extends A {
   public void display(){

      System.out.println("Inside Constructor of class B");
     // System.out.println("Default Variable in class A" +defaultvar);
      System.out.println("Protected variable in class A"+protectedVar);
      //System.out.println("Private variable in class A"+privateVar);
      System.out.println("Public variable in class A"+publicVar);

   }
}
```

Define class C in package cpack which has one method display() in that create one object of class A and display its variables.

```
package cpack;
import apack.A;
```

```
public class C {
   public void display(){
      A obj = new A();

      System.out.println("Inside Constructor of class C");
      //System.out.println("Default Variable in class A" +obj.defaultvar);
      //System.out.println("Protected variable in class A"+obj.protectedVar);
      //System.out.println("Private variable in class A"+privateVar);
      System.out.println("Public variable in class A"+obj.publicVar);

   }
}
```

Define class ProtectedDemo in package dpack which contains the main () method. Create objects of class B and C and call display method for both these objects. Analyze the program by interpreting the access modifiers and provide valid conclusion.

```
package com.mycompany.mavenproject2;
import bpack.B;
import cpack.C;
public class Mavenproject2 {

   public static void main(String[] args) {
      B objB = new B();
      objB.display();
      C objC = new C();
      objC.display();

   }
}
```