MANIPAL ACADEMY OF HIGHER EDUCATION

III SEMESTER B.TECH. (CSE and AIML)
MID SEMESTER EXAMINATIONS, SEP 2023

OBJECT ORIENTED PROGRAMMING [CSE 2124]

Marks: 30 **Duration: 120 mins.**

A

Answer all the questions.

Q1. What is the output of this program?

```
interface compute
{
   void multiply(int no);
}
class show implements compute
{
            int n = 1;
            public void multiply(int no)
            {
                        n = n * no;
            }
}
public class  Demo
{
   public static void main(String args[])
   {
    compute c  = new show();
    c.multiply(5);
    System.out.print(c.n);
   }
} (0.5)
```

1. 0
2. 1
3. 5
4. **Compilation Error

Q2. What is the output of the following program?

```
interface iA{
        int i = 10;
}

public class  Demo implements iA
{
  public static void main(String args[])
  { i = 12;
    System.out.println(i);

  }}
```
(0.5)

1. 10
2. 12
3. **Compilation Error
4. Runtime Error

Q3. When is the object created with new keyword? (0.5)

1. **At run time
2. At compile time
3. Depends on the code
4. When program is loaded

Q4. Which of the following is true in Java?

i) Enables creation of cross-platform programs

ii) Can resolve type information at run time

iii) Does not support memory management

iv) Execution is confined to JVM (0.5)

1. All are true
2. i, ii. iii
3. **i, ii, iv
4. ii ,iii, iv

Q5. What is the output(if any) of the following?

```
class UDExc extends Exception{

        int val;

        UDExc(int v){

                super("The value resulting in error is: "+v);
```

```
            val = v;
        }
}
class ExcGen{
        static void excGen(int i) throws UDExc{
                if(i%10==0)
                        throw new UDExc(i);
        }
}
class ExcGenDemo{
        public static void main(String[] args){
                try{
                        ExcGen.excGen(20);
                        ExcGen.excGen(30);
                }
                catch(UDExc u){
                        System.out.println(u);
                }
        }
}
```

(0.5)

1. Compilation Error as toString() is not overridden in UDExc class
2. **UDExc: The value resulting in error is: 20
3. UDExc: The value resulting in error is: 30
4. UDExc: The value resulting in error is: 20

Q6. What is the output(if any) of the following program?

```
class ExcTestDemo{
        static void method(){
        try{throw new RuntimeException();}
        finally{
```

```
        System.out.println("Leaving from finally...");
        return;
        }
}
        public static void main(String[] args){
                try{
                method();
                }
        catch(RuntimeException e){
                System.out.println(e);
                }
        }
} (0.5)
```

1. ** Leaving from finally
2. Leaving from finally
java.lang.RuntimeException
3. java.lang.RuntimeException
Leaving from finally
4. Compile-time Error as try in method() is not associated with catch()

Q7. Which of the following statement(s) about finalize() method are correct? (0.5)

1. Used for specific actions that will occur when an object is just about to be reclaimed by the garbage collector
2. can be added to a class, by simply defining the finalize( ) method
3. used to free non-java resources such as file handle or window character font
4. ** All the options

Q8. Which of the following features are resolved at compile time ?

A) Method overloading B) Method overriding C) Constructor overloading (0.5)

1. Both A and B
2. **Both A and C
3. Both B and C
4. All three features (A, B and C)

Q9. How many static blocks can a Java class have? (0.5)

1. Only one static block is allowed per class
2. Multiple static blocks are allowed, but they must be placed at the beginning of the class
3. **Multiple static blocks are allowed, and their placement within the class doesn't matter
4. A class can have any number of static blocks but put immediately after data declaration

Q10. Write the output of the following program.

```
class Superclass {
   static {  System.out.print("L1 ");        }
}

class Subclass extends Superclass {
   static {   System.out.print("L2 ");   }
}

public class Main {

          static { System.out.print("L3 ");}
   public static void main(String[] args) {
      Subclass obj = new Subclass();
   }
   static {System.out.print("L4 ");      }
}
```

(0.5)

1. ** L3 L4 L1 L2
2. L1 L2 L3 L4
3. L3 L1 L2 L4
4. Compile error

Answer all the questions. Section Duration: 100 mins

Answer all the questions

11. Write a method ComputeArmstrong(int) to check if a number is armstrong or not. Create a class ComputeDemo to input lower and upper limits and display all Armstrong numbers between these limits using the ComputeArmstrong(int) method and find the sum of these Armstrong numbers in main() method.Note: Consider any n-digit number.

Ans:

Ans: Main Class with function call 1M; Logic: 1M; Declaration of static method: 0.5M, Sum calculation:0.5M

```
public class Armstrong {
        public static void main(String[] args) {
       int low=0 , high=100,sum=0;
        for(int number = low + 1; number < high; ++number) {
          if (checkArmstrong(number)){
              System.out.println(number + " ");
              sum=sum+number;}
  }
   System.out.println("sum is" +sum);
 }

   public static boolean checkArmstrong(int num) {
      int digits = 0;
      int result = 0;
      int originalNumber = num;

      // number of digits calculation
      while (originalNumber != 0) {
         originalNumber /= 10;
         ++digits;
      }

      originalNumber = num;

      // result contains sum of nth power of its digits
      while (originalNumber != 0) {
         int remainder = originalNumber % 10;
         result += Math.pow(remainder, digits);
         originalNumber /= 10;
      }

      if (result == num)
         return true;

      return false;
   }
}
```

Ans: Main Class with function call 1M; Logic: 1M; Declaration of static method: 0.5M, Sum calculation:0.5M

```
   public class Armstrong {
        public static void main(String[] args) {
       int low=0 , high=100,sum=0;
        for(int number = low + 1; number < high; ++number) {
            if (checkArmstrong(number)){
                System.out.println(number + " ");
                sum=sum+number;}
    }
     System.out.println("sum is" +sum);
  }

    public static boolean checkArmstrong(int num) {
        int digits = 0;
        int result = 0;
        int originalNumber = num;

        // number of digits calculation
        while (originalNumber != 0) {
            originalNumber /= 10;
            ++digits;
        }

        originalNumber = num;

        // result contains sum of nth power of its digits
        while (originalNumber != 0) {
            int remainder = originalNumber % 10;
            result += Math.pow(remainder, digits);
            originalNumber /= 10;
        }

        if (result == num)
            return true;

        return false;
    }
}
```

12.

Create a class Movie with data members mov_id(short), screen_num(short), pdate(String) to store screening date (e.g., 09-JAN-2020) and ptime(float) to store screening time (e.g., (hh.mm)18.45 or 22.10). Provide a parameterised constructor to initialise all the data members and a display() method to output the movie details. Provide a checkSchedule(Movie[] m) that checks for any conflicting screenings. Any two movies have a conflicting screening if they are screened on the same date and screen with a gap of less than 3 hours. Raise user-defined ScreeningException from checkSchedule() if there are conflicts with an appropriate message to the user such as: "Schedules of movies with Ids 10 and 40 are conflicting". Create an array of movies, display them, and handle ScrreningException in main() written in the class MovieDemo.
Ans:

```
import java.util.Scanner;
```

```
class ScreeningException extends Exception{
	int mov_1, mov_2;
	ScreeningException(int a, int b){
		mov_1 = a;
		mov_2 = b;
	}--------------------------------------------------------------------------------------------------------0.5
	public String toString(){
		return "Schedules of movies "+mov_1+" and "+mov_2+" are conflicting.";
	}--------------------------------------------------------------------------------------------------------0.5
}
class Movie{
	short screen_num;
	short mov_id;
	String pdate;//dd-mm-yyyy format 01-JAN-2020
	float ptime;//

	Movie(short s, short m, String d, float t){
		screen_num=s;
		mov_id=m;
		pdate=d;
		ptime=t;
	}
	void display(){
		System.out.println("Movie Id:"+mov_id+"Screen: "+screen_num+"Date: "+pdate+"Time: 
"+ptime);
	}----------------------------------------------------------------------------------------------------0.5
	void checkSchedule(Movie[] m) throws ScreeningException{
		for(int i =0; i<m.length; i++)
				for(int j=i+1;j<m.length;j++)
					if((m[i].pdate.equals(m[j].pdate))    &&    (m[i].screen_num    ==
m[j].screen_num) && (Math.abs(m[i].ptime - m[j].ptime) < 3))
```

```
                                        throw      new      ScreeningException(m[i].mov_id,
m[j].mov_id);
        }---------------------------------------------------------------------------------------------------------0.5
}
class MovieDemo{
        public static void main(String[] args){
                Scanner sc = new Scanner(System.in);
                Movie[] m = new Movie[5];
                int count = 0;
                for(int i =0; i<m.length; i++){
                        System.out.println("Enter the screen_num");
                        short sn = sc.nextShort();
                        System.out.println("Enter the movie id");
                        short mov_id = sc.nextShort();
                        System.out.println("Enter the screening time");
                        float st = sc.nextFloat();
                        sc.nextLine();
                        System.out.println("Enter the date");
                        String d = sc.nextLine();
                        m[i] = new Movie(sn,mov_id,d,st);
                }
                for(Movie mov:m)
                        mov.display();-------------------------------------------------------------0.5

                try{
                        m[0].checkSchedule(m);
                }
                catch(ScreeningException e){
                        System.out.println(e);
                }-------------------------------------------------------------------------------------------0.5
```

```
		}
	}
	import java.util.Scanner;
	class ScreeningException extends Exception{
		int mov_1, mov_2;
		ScreeningException(int a, int b){
			mov_1 = a;
			mov_2 = b;
		}--------------------------------------------------------------------------------------------------------0.5
		public String toString(){
			return "Schedules of movies "+mov_1+" and "+mov_2+" are conflicting.";
		}-------------------------------------------------------------------------------------------------------0.5
	}
	class Movie{
		short screen_num;
		short mov_id;
		String pdate;//dd-mm-yyyy format 01-JAN-2020
		float ptime;//

		Movie(short s, short m, String d, float t){
			screen_num=s;
			mov_id=m;
			pdate=d;
			ptime=t;
		}
		void display(){
			System.out.println("Movie Id:"+mov_id+"Screen: "+screen_num+"Date: "+pdate+"Time:
"+ptime);
		}---------------------------------------------------------------------------------------------------0.5
		void checkSchedule(Movie[] m) throws ScreeningException{
```

```
                for(int i =0; i<m.length; i++)
                        for(int j=i+1;j<m.length;j++)
                            if((m[i].pdate.equals(m[j].pdate))  &&  (m[i].screen_num  ==
m[j].screen_num) && (Math.abs(m[i].ptime - m[j].ptime) < 3))
                                throw    new    ScreeningException(m[i].mov_id,
m[j].mov_id);
        }-----------------------------------------------------------------------------------------------------------0.5
}
class MovieDemo{
        public static void main(String[] args){
                Scanner sc = new Scanner(System.in);
                Movie[] m = new Movie[5];
                int count = 0;
                for(int i =0; i<m.length; i++){
                        System.out.println("Enter the screen_num");
                        short sn = sc.nextShort();
                        System.out.println("Enter the movie id");
                        short mov_id = sc.nextShort();
                        System.out.println("Enter the screening time");
                        float st = sc.nextFloat();
                        sc.nextLine();
                        System.out.println("Enter the date");
                        String d = sc.nextLine();
                        m[i] = new Movie(sn,mov_id,d,st);
                }
                for(Movie mov:m)
                        mov.display();--------------------------------------------------------------0.5

                try{
                        m[0].checkSchedule(m);
                }
```

```
            catch(ScreeningException e){
                System.out.println(e);
            }-----------------------------------------------------------------------------------------0.5
        }
}
```

13. **Write a Java program to create a package named “Mypackage” which is a sub package of “Allpackage”. Create a class “Macbook” with two instance variables *RAM_Capacity* and *Processor.* Create an interface “Browsing” with method *AccessInternet(String bandwidth).* Keep the “Macbook” class and the “Browsing” interface in the package “Mypackage”. Import this package in a new file and create a subclass “Macbook_air” from class “Macbook” also it implements the interface. Demonstrate the object creation to invoke the constructors and to call the methods of the interface.**

Browsing.java (inside "Mypackage"):

```
package Allpackage.Mypackage;

public interface Browsing {
   void AccessInternet(String bandwidth);
}
```

Macbook.java (inside "Mypackage"):

```
package Allpackage.Mypackage;

public class Macbook {
   protected int RAM_Capacity;
   protected String Processor;

   public Macbook(int ramCapacity, String processor) {
      this.RAM_Capacity = ramCapacity;
      this.Processor = processor;
   }
}
```

Macbook_air.java (inside "Mypackage"):

```
package Allpackage.Mypackage;

public class Macbook_air extends Macbook implements Browsing {
    public Macbook_air(int ramCapacity, String processor) {
        super(ramCapacity, processor);
    }

    @Override
    public void AccessInternet(String bandwidth) {
        System.out.println("Accessing the internet with bandwidth: " + bandwidth);
    }
}
```

Main.java (outside the packages):

```
import Allpackage.Mypackage.*;

public class Main {
    public static void main(String[] args) {
        Macbook_air macbookAir = new Macbook_air(8, "Intel i7");
        macbookAir.AccessInternet("High-speed");
    }
}
```

14. What order must be followed when both subclass and superclass exceptions are listed in catch clause? Justify.

Ans: The superclass exceptions must always follow the subclass exceptions and hence must be present towards the end of the catch clause list.---1M

This is because if a superclass exception is present before a subclass exception, it will always be used whenever an exception occurs. However, subclass exception is never reached. As Java does not support unreachable code correct ordering must be followed. ---------------1M

15. Write a Java program to execute the following main method. Define Book class with private members: a unique book id, title, author, and availability status (Boolean). Book may be a Textbook with additional member subject. Book may also be a Novel with additional member genre. Demonstrate method overriding and use super keyword appropriately. Define a Library class with a Book array of size MAX_BOOKS as data member and a method addBook() to take a Book type as parameter and add that to the library. Initialize Book array with 5 books appropriately.

```
public class Main {
public static void main(String[] args) {
Library.addBook(new Novel(6, "The Lord of the Rings", "J.R.R. Tolkien", "Fantasy"));
Book[] books = Library.getAllBooks();
System.out.println("Library Book Information:");
for (int i=0;i< Library.bookCount;i++)
books[i].display();
}

}
```

Ans:

```
class Book {
          private int id;
          private String title;
          private String author;
          private boolean available;

          public Book(int id, String title, String author) {
            this.id = id;
            this.title = title;
            this.author = author;
            this.available = true;
          }
          public int getId() {
            return id;
          }
          public String getTitle() {
            return title;
          }
          public String getAuthor() {
            return author;
          }

          public boolean isAvailable() {
            return available;
          }
          public void setAvailable(boolean available) {
            this.available = available;
          }
          public void display() {
            System.out.print("ID: " + id + ", Title: " + title + ", Author: " + author);
          }
}//-----------------------------1M
class Novel extends Book {
     String genre;
    public Novel(int id, String title, String author, String genre) {
        super(id, title, author);
        this.genre = genre;
    }
       public void display() {
       System.out.println("Novel :");
        super.display();
        System.out.println("Genre: " + genre);
    }
}//------------------------0.5M

class Textbook extends Book {
    String subject;
    public Textbook(int id, String title, String author, String subject) {
        super(id, title, author);
        this.subject = subject;
    }
    public void display() {
       System.out.println("Textbook :");
        super.display();
        System.out.println("Subject: " + subject);
    }
}}//------------------------0.5M

class Library {
    private static final intMAX_BOOKS = 100;
    private static Book[] bookDatabase = new Book[MAX_BOOKS];
    static int bookCount = 0;
    static {
     bookDatabase[bookCount++] = new Novel(1, "The Catcher in the Rye", "J.D. Salinger", "Fiction");
     bookDatabase[bookCount++] = new Novel(2, "To Kill a Mockingbird", "Harper Lee", "Fiction");
     bookDatabase[bookCount++] = new Novel(3, "1984", "George Orwell", "Science Fiction");
     bookDatabase[bookCount++] = new Textbook(4, "Calculus", "James Stewart", "Maths");
     bookDatabase[bookCount++] = new Textbook(5, "Introduction to Java", "John Smith", "CS");
    } }//------------------------1M

    public static void addBook(Book book) {
        if (bookCount < MAX_BOOKS) {
            bookDatabase[bookCount++] = book;
        } else {
            System.out.println("Library is at maximum capacity. Cannot add more books.");
        }
    }
public static Book[] getAllBooks() {
    return bookDatabase;
}
}
}//------------------------1M
```

16. Write a Java program to define a class called Matrix that has a 2D array of integers and the following methods.

a) A parameterized constructor to receive two integers(m and n)as parameter and allocate memory

for the 2D array of size m*n.
b) Another parameterized constructor to receive an integer 2D array and allocate memory to the instance variable and initialize the array with the contents of the array received.
c) Method to display the Matrix using for each loop
d) Method find() to return a 1D array containing number of palindromic prime numbers [Example Palindromic prime numbers: 11, 101, 131, 151, etc.,].
Test class Matrix with class MatrixDemo having main method to test all the methods of the class appropriately.

Ans:

```
import java.util.*;
class Matrix {
  int row, col, a[][];

  Matrix(int m, int n) {
        row = m;
        col = n;
        a=new int[m][n];
  }                                         //0.5M

  Matrix(int a[][]) {
     row = a.length;
        col = a[0].length;
        this.a = new int[row][col];
        for(int i=0;i<row;i++)
          for(int j=0;j<col;j++)
        this.a[i][j]=a[i][j];
  }                                         // 0.5M

  int[] find() {
     int[] palprimes = new int[row * col];
     int count = 0;

     for (int i = 0; i < row; i++) {
       for (int j = 0; j < col; j++) {
          int num = a[i][j];
          int save = num, n = num;
          int prime = 1;
          for(int k=2; k<num/2; k++) {
             if( num % k == 0) {
                prime=0;
                break;
             }
          }
          int dig, rev=0;
          while(n > 0) {
                   dig = n % 10;
                   rev = rev * 10 + dig;
                   n = n / 10;
                }
                if (prime == 1 && rev==save) {
                        palprimes[count] = num;
                        count++;
                }



               }
            }
            return palprimes;
    }                                                          // 1.5M

    void dispMatrix() {
            for ( int row[]: a) {
               for ( int e: row)
                        System.out.print(e+" ");
               System.out.println();
            }
    }                                                            //0.5M
}

public class MatrixDemo {
    public static void main(String[] args) {
            Scanner sc = new Scanner(System.in);
            int i, j, row, col;

        // Taking input from the user
        System.out.print("Enter the number of rows:");
        row = sc.nextInt();

        // Display message
        System.out.print("Enter the number of columns:");

        // Reading matrix elements individually using nextInt() method
        col = sc.nextInt();

        // Declaring a 2D array(matrix)
        int[][] mat = new int[row][col];
        System.out.print("Enter the matrix elements:");
        for (i = 0; i < row; i++)
               for (j = 0; j < col; j++)
                        mat[i][j] = sc.nextInt();

        Matrix m1=new Matrix(mat);
        System.out.println("The matrix is ");
        m1.dispMatrix();

        // calling find() function to return array of palindromic prime numbers
        int pp[] = m1.find();
           for(int num : pp) {
              if(num!=0)
           System.out.print(num + " ");
        }
   }
}                                        //1M
```

17. Design a base class called Employee with 3 data members: (i) EmpName (ii) EmpID and (iii) BasicPay. Include parameterized constructor to initialize the data members. Define an Interface called Research which contains a method named Incentives() which computes and returns the Incentive.[Note: Incentive will be ResearchPoints X 1000] . Derive a class called Researcher which inherits Employee class and implements the interface Research with data members: DA(10% of BasicPay), HRA(15% of BasicPay) and ResearchPoints. Include a method TotalPay() to compute and display the total pay of the researcher [Total pay= Basic+DA+HRA+ Incentive]. Write a demo class which includes the main() to demonstrates the working of Researcher class.

Ans:

```
import java.util.Scanner;
interface Research
{         double Incentive();
}  0.5M

class Employee
{
  int EmpID;
 String Empname;
double basic;
 Employee(String n, int x,  double b)
{
      Empname=n;
      EmpID =x;basic=b;
}
} 0.5M

class Researcher extends Employee implements Research
{
            double DA;
            double HRA;
            int Researchpoint;
 Researcher(String n, int x,  double b, int rp)
{
      super(n,x,b);
            DA= 0.1*basic; HRA=0.15*basic;
            Researchpoint=rp;}
public double Incentive()
{   return Researchpoint*1000;
}
double pay ()
{ return (basic+DA+HRA+Incentive());
}
} 1.5M
class PayDemo
{     public static void main(String[] args)
      {
         Researcher p1= new Researcher("john",1200,45000.0,60);
         System.out.println("Total pay:"+p1.pay());

     }   }        0.5M
```

18. Discuss the difference between checked and unchecked exceptions and write example programs to show the way to handle checked and unchecked exceptions in Java.

Ans:

Checked Exceptions:

Checked exceptions are exceptions that are checked at compile time. This means the Java compiler ensures that you either handle these exceptions using try-catch blocks or declare them in the method's throws clause.

Checked exceptions typically represent external factors that can go wrong and are beyond the control of your program, such as I/O errors, network errors, and database errors.

You must explicitly handle or declare checked exceptions, making the code more robust and helping developers anticipate and deal with potential issues.

Here's an example of handling a checked exception (IOException) using a try-catch block:

```
import java.io.*;


public class CheckedExceptionExample {

    public static void main(String[] args) {

        try {

            FileReader fileReader = new FileReader("nonexistent-file.txt");

            BufferedReader bufferedReader = new BufferedReader(fileReader);

            String line = bufferedReader.readLine();
            bufferedReader.close();
        } catch (IOException e) {
            System.err.println("IOException occurred: " + e.getMessage());
        }
    }
}
```

Unchecked Exceptions (Runtime Exceptions):

Unchecked exceptions, also known as runtime exceptions, are not checked by the compiler at compile time. These exceptions typically indicate programming errors or bugs in your code, such as dividing by zero, accessing an array out of bounds, or calling a method on a null object.

You are not required to handle or declare unchecked exceptions, which can lead to unexpected crashes if not handled properly.

While unchecked exceptions don't need explicit handling, it's good practice to handle them or use proper defensive programming techniques to prevent them.

Here's an example of an unchecked exception (ArithmeticException):

```
public class UncheckedExceptionExample {
    public static void main(String[] args) {
        int numerator = 10;
        int denominator = 0;

        try {
            int result = numerator / denominator; // This will throw an ArithmeticException
        } catch (ArithmeticException e) {
            System.out.println("ArithmeticException occurred: " + e);
        }
    }
}
```

In summary, the key difference between checked and unchecked exceptions in Java is the requirement to handle or declare them. Checked exceptions are enforced to be handled or declared, whereas unchecked exceptions are not. Handling checked exceptions is essential for robust code, especially when dealing with external resources or operations, while unchecked exceptions are typically indicative of code logic errors and should be fixed during development.