Q No	Questions	Marks	CO	BT

1	Find the output of the following code snippet.
	class access
	{ static int x;
	void increment()
	{ x++; } }
	class static_use
	{
	public static void main(String args[])
	{
	access obj1 = new access();
	access obj2 = new access();
	obj1.x = 0;
	obj1.increment();
	obj2.increment();
	System.out.println(obj1.x + " " + obj2.x);
	}
	}
	1. 1 2
	2. 1 1
	3. ** 2 2
	4. Compilation Error
		0.5	CO1	3

2	Identify the output of the following code snippet.
	class output
	{
	public static void main(String args[])
	{
	String c = "Hello i love java";
	int start = 2;
	int end = 9;
	char s[]=new char[end-start];
	c.getChars(start,end,s,0);
	System.out.println(s);
	}
	}
	1. Hello, i love java
	2. i love ja
	3. lo i lo
	4. **llo i l
		0.5	CO1	3

3	Determine the output for the following. class Test {	0.5	CO1	3

	static int a;
	static
	{
	a = 4;
	System.out.println ("inside static block\\n");
	System.out.println ("a = " + a);
	}
	Test()
	{
	System.out.println ("\\ninside constructor\\n");
	a = 10;
	}
	public static void func()
	{
	a = a + 1;
	System.out.println ("a = " + a);
	}
	public static void main(String[] args)
	{
	Test obj = new Test();
	obj.func();
	}
	}
	
	1. **inside static block
	a=4
	inside constructor
	a=11
	2. inside static block
	a=4
	inside constructor
	a=5
	3. inside static block
	a=10
	inside constructor
	a=11
	4. Run time Error			

4	Find the output for the following
	class Base {
	public final void display() {
	System.out.println("Display from Base");
	}
	}
	class Derived extends Base {
		0.5	CO1	3

	public void display() {
	System.out.println("Display from Derived");
	}
	}
	public class Test {
	public static void main(String[] args) {
	Base b = new Derived();
	b.display();
	}
	}
	
	1. Display from Derived
	2. Display from Base
	3. **Compilation Error
	4. Runtime Error			

5	Predict the output of the following program?
	class A
	{
	public void print() { System.out.print("A"); }
	}
	class B extends A
	{
	public void print() { System.out.print("B"); }
	}
	class Main
	{
	public static void Doprint(A O)
	{ O.print(); }
	public static void main(String args[])
	{
	A x = new A();
	B y = new B();
	B z= new B();
	Doprint(x);
	Doprint(y);
	Doprint(z);
	}
	}
	
	1. BBA
	2. BAB
	3. ** ABB
	4. AAA
		0.5	CO2	3

6	Q6. The _____method allows run-time polymorphism in Java. 1. Overloaded	0.5	CO2	2

	2. ** Overridden 3. Same class 4. Different parameter types but with the same name			
7	Q7. When is the object created with new keyword? 1. **At run time 2. At compile time 3. Depends on the code 4. When the program is loaded	0.5	CO1	2
8	Q8. Identify the true statements in Java. i) Enables the 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 1. All are true 2. i, ii. iii 3. **i, ii, iv 4. ii, iii, iv	0.5	CO1	2
9	Q9. Identify the correct statement(s) about the finalize() method. 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	0.5	CO1	2

10	Q10. Determine the output of the following code.
	class Test {
	public static void main(String args[]) {
	int arr[2];
	System.out.print(arr[0]);
	System.out.println(arr[1]);
	}
	}
	1. **Compile Error
	2. garbage value garbage value
	3. Exception
	4. 0 0
		0.5	CO1	3

11	A company pays its employees on a weekly basis (Consider company as an abstract class and empName as a private variable). The employees are of three types: i) Salaried employees are paid a fixed weekly salary regardless of the number of hours worked, ii) hourly employees are paid by the hour and receive overtime pay (i.e., 1.5 times their hourly salary	4	CO2	4

rate) for all hours worked in excess of 40 hours, iii) commission employees are paid a percentage of their sales. Design the class hierarchy, include the constructors in each class with appropriate instance variables, make the instance variables private and method to print the employee name and salary. Write a test class to show all the functionalities. Use dynamic method dispatch. (Note: make suitable assumptions for data).

Scheme: For each class – 0.5 M except PartEmp (1M), for dynamic method dispatch - 1 M = 4M

```
abstract class Emp{
private String name;
Emp(String s){
name= s; }
String getName( ) { return name; }
abstract void computeSal( );
 }
class RegEmp extends Emp {
int salary;
RegEmp( String n,int s ) { super(n); salary = s; }
void computeSal( ){
 System.out.println( "name :" + getName( ) + " salary :"+ salary); }
 }
class PartEmp extends Emp {
static int rate = 500;
int no_hrs;
PartEmp(String s,int n ) { super(s); no_hrs = n; }
void computeSal( ){
 double sal;
 if(no_hrs>40) sal=( (no_hrs-40) * 1.5 + 40 ) * rate;
 else sal= no_hrs * rate;
 System.out.println( "name :" + getName( ) + " salary :"+ sal); }
 }
class CommEmp extends Emp {
static double comm_rate = 0.05;
int totalsales;
CommEmp(String s,int n ) { super(s); totalsales = n; }
void computeSal( ){
System.out.println( "name :" + getName( ) + " salary :"+
(comm_rate*totalsales)); }
 }
class Test {
public static void main(String args[ ]) {
 Emp e[]=new Emp[3];
 e[0]=new RegEmp( "Anand",80000);
 e[1]=new PartEmp("Arun",45);
```

	e[2]=new CommEmp("Sushma",2000000);
	for(int i=0;i<3;i++)
	e[i].computeSal();
	}
	}			

12	Write a Java program that reads a student's name and year of graduation. If the graduation year is a leap year, a custom exception should be thrown from **CheckLeapYear ()** with an appropriate message that displays the leap year.
	- Custom exception class – 1M
	- Leap year logic – 1M
	- Main, Class and objects – 1 M
	import java.util.Scanner;
	// Custom exception class for leap year
	class LeapYearException extends Exception {
	public LeapYearException(String message) {
	super(message);
	}
	}
	// Student class
	class Student {
	private String name;
	private int graduationYear;
	public Student(String name, int graduationYear) {
	this.name = name;
	this.graduationYear = graduationYear;
	}
	public String getName() {
	return name;
	}
	public int getGraduationYear() {
	return graduationYear;
	}
	// Method to check if the year is a leap year
	public void checkLeapYear() throws LeapYearException
	{
	if ((graduationYear % 4 == 0 && graduationYear % 100 != 0)
	|| (graduationYear % 400 == 0)) {
	throw new LeapYearException("Graduation year " +
	graduationYear + " is a leap year.");
	}
	}
	}
	// Main class to read input and handle exceptions
	public class LeapYearCheck {
		3	CO4	3

	public static void main(String[] args) {
	Scanner scanner = new Scanner(System.in);
	try {
	// Reading student details
	System.out.print("Enter student's name: ");
	String name = scanner.nextLine();
	System.out.print("Enter graduation year: ");
	int graduationYear = scanner.nextInt();
	// Creating Student object
	Student student = new Student(name, graduationYear);
	// Checking if graduation year is a leap year
	student.checkLeapYear();
	System.out.println("Student " + student.getName() + " has a non-leap graduation year: " + student.getGraduationYear());
	} catch (LeapYearException e) {
	// Handling custom leap year exception
	System.out.println(e.getMessage());
	} finally {
	scanner.close();
	}
	} }			

13	Develop a Java program to manage the library system, where each book has the attributes: title, author, and ISBN (all strings). The program should allow the librarian to enter the details of up to five books, storing them in an array of Book objects. The program should include two methods one for displaying book details and the other for searching a book by its partial title, and if a match is found, display the corresponding book(s) details.
	classes, constructors – 1M
	Array of Objects- 1M
	Strings – 1M
	
	Scheme:
	import java.util.Scanner;
	class Book {
	private String title;
	private String author;
	private String isbn;
	// Constructor
	public Book(String title, String author, String isbn) {
	this.title = title;
	this.author = author;
	this.isbn = isbn;
	}
		3	CO1	3

```
    // Getters
    public String getTitle() {
        return title;
    }
    public String getAuthor() {
        return author;
    }

    public String getIsbn() {
        return isbn;
    }

    // Method to display book details
    public void displayBookDetails() {
        System.out.println("Title: " + title + ", Author: " + author + ", ISBN: " +
isbn);
    }
}

public class LibrarySystem {
    private Book[] books;
    private int count;

    public LibrarySystem(int size) {
        books = new Book[size];
        count = 0;
    }

    // Method to add a book
    public void addBook(String title, String author, String isbn) {
        if (count < books.length) {
            books[count++] = new Book(title, author, isbn);
        } else {
            System.out.println("Library is full. Cannot add more books.");
        }
    }
// Method to display all book details
    public void displayAllBooks() {
        for (int i = 0; i < count; i++) {
            books[i].displayBookDetails();
        }
    }
    // Method to search for a book by partial title
    public void searchByPartialTitle(String partialTitle) {
        boolean found = false;
```

	for (int i = 0; i < count; i++) {
	if
	(books[i].getTitle().toLowerCase().contains(partialTitle.toLowerCase())) {
	books[i].displayBookDetails();
	found = true;
	}
	}
	if (!found) {
	System.out.println("No book found with the given partial title.");
	}
	}
	public static void main(String[] args) {
	Scanner scanner = new Scanner(System.in);
	LibrarySystem library = new LibrarySystem(5);
	// Input book details
	for (int i = 0; i < 5; i++) {
	System.out.println("Enter details for book " + (i + 1) + ":");
	System.out.print("Title: ");
	String title = scanner.nextLine();
	System.out.print("Author: ");
	String author = scanner.nextLine();
	System.out.print("ISBN: ");
	String isbn = scanner.nextLine();
	library.addBook(title, author, isbn);
	}
	// Display all book details
	System.out.println("\nDisplaying all book details:");
	library.displayAllBooks();
	// Search for a book by partial title
	System.out.print("\nEnter a partial title to search: ");
	String partialTitle = scanner.nextLine();
	library.searchByPartialTitle(partialTitle);
	scanner.close();
	}
	}			

14	An interface **MyConstants** is defined as follows:
	package mypackage;
	public interface MyConstants {
	int ANSWER = 42;
	}
	Demonstrate two ways to bring the constant ANSWER into view for use in a class **ExamDemo** (residing outside *mypackage*) that contains the **main()** with appropriate justifications**.**
		3	CO2	4

	Answer:
	i) ExamDemo can implement mypackage.MyConstants:
	import mypackage.MyConstants;
	class ExamDemo implements MyConstants{
	public static void main(String[] args){
	System.out.println("The constant value is:"+ANSWER);
	}
	}
	Once a class implements an interface, the data members can be used directly as though it were an instance variable of the class.----------(1+0.5)
	ii) ExamDemo can use static import statement:
	import static mypackage.MyConstants.*;
	class ExamDemo{
	public static void main(String[] args){
	System.out.println("The constant value is:"+ANSWER);
	}
	}
	The static import statement brings all the static members of interface MyConstants into view for use in ExamDemo class.----------(1+0.5)			

15	Write a method checkArmstrong(int) to check if a number is Armstrong (*a number that equals the sum of its digits, each raised to a power*) or not. Create a class ComputeDemo to input lower and upper limits and display all Armstrong numbers between these limits using the checkArmstrong(int) method and find the sum of these Armstrong numbers in the main() method. Note: Consider any n-digit number. Main Class with function call 1M; Logic: 1M; Declaration of static method: 0.5M, Sum calculation:0.5M public class ComputeDemo {	3	CO1	2

	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;
	}
	}			

16	Create a Java program to simulate bank operations using exception handling. First, define three user-defined exceptions: the base class **InvalidTransactionException**, and two subclasses, **InsufficientBalanceException** (triggered when the withdrawal amount exceeds the account balance) and **DailyLimitExceededException** (triggered when the withdrawal exceeds a daily limit of 50,000 units). In the **BankTransaction** class, prompt the user to enter **account_no**, **current_balance**, and **withdrawal_amount**. Use nested **try-catch** blocks where the outer	3	CO4	3

block checks the correctness of the **account_no**, and the inner block checks for both **InsufficientBalanceException** and **DailyLimitExceededException**. Display relevant error messages when exceptions are caught by suitably using the constructor of **Exception** class.
Note: Use the default value "12345678" for the valid account number and set the daily withdrawal limit to Rs. 50,000.

Correct order of inheritance in Super and subclass exceptions (1M)
Nested Try Blocks (1M)
Suitable conditions and throw statements (1M)

```
import java.util.Scanner;

// Base exception class for invalid transactions
class InvalidTransactionException extends Exception {
   public InvalidTransactionException(String message) {
      super(message);
   }
}
// Exception for insufficient balance

class InsufficientBalanceException extends InvalidTransactionException {

   public InsufficientBalanceException(String message) {

      super(message);

   }

}

// Exception for daily limit exceeded

class DailyLimitExceededException extends InvalidTransactionException {

   public DailyLimitExceededException(String message) {

      super(message);    }

}

public class BankTransaction {
   // Predefined constants for account validation and daily limit
   private static final String VALID_ACCOUNT_NUMBER = "12345678";
   private static final double DAILY_LIMIT = 50000.0;
   public static void main(String[] args) {
      Scanner scanner = new Scanner(System.in);
      try {
```

```
        // Prompt user for inputs
        System.out.print("Enter account number: ");
        String accountNo = scanner.nextLine();
        // Outer try block to validate account number
        try {
            if (!accountNo.equals(VALID_ACCOUNT_NUMBER)) {
                throw new InvalidTransactionException("Invalid account
number entered.");
            }
            // Prompt for current balance and withdrawal amount
            System.out.print("Enter current balance: ");
            double currentBalance = scanner.nextDouble();
            System.out.print("Enter withdrawal amount: ");
            double withdrawalAmount = scanner.nextDouble();
            // Inner try block to check balance and daily limit
            try {
                if (withdrawalAmount > DAILY_LIMIT) {
                    throw new DailyLimitExceededException("Withdrawal
exceeds the daily limit of 50,000 units.");
                }
                if (withdrawalAmount > currentBalance) {
                    throw new InsufficientBalanceException("Insufficient
balance for this withdrawal.");
                }
                // If no exception occurs, process the transaction
                currentBalance -= withdrawalAmount;
                System.out.println("Transaction successful! Updated balance:
" + currentBalance);

            } catch (InsufficientBalanceException |
DailyLimitExceededException e) {
                // Handle balance and limit exceptions
                System.out.println("Error: " + e.getMessage());
            }
```

	} catch (InvalidTransactionException e) {
	// Handle invalid account number exception
	System.out.println("Error: " + e.getMessage());
	} } } }			

17	Outline the advantages of Packages in Java. **Each point 0.5 mark** The benefits of packages are – • Classes in packages can be easily reused • Package provides a way to organize related pieces of a program as a unit. • Two different packages can contain classes with same name • Provide a way to 'hide' classes thus preventing other programs or packages from accessing classes that are meant for internal use only. • Thus, packages participate in java's access control mechanism.	2	CO2	2

18	Consider the below Java program that contains several issues in how exceptions are handled during array operations. Identify the errors (line nos) and rewrite the code.
	`1 public class ArrayOperations {`
	`2 public static void main(String[] args) {`
	`3 try {`
	`4 try {`
	`5 int[] numbers = null;`
	`6 numbers[0] = 10;`
	`7 System.out.println(number[2]);`
	`8 } catch (ArrayIndexOutOfBoundsException e) {`
	`9 System.out.println("Array index is out of bounds");`
	`10 throw e;`
	`11 }`
	`12 } catch (ArithmeticException e) {`
	`13 System.out.println("An error occurred: " + e);`
	`14 }`
	`15 }`
	`16 }`
	Identification of errors: 1 M
	Correct codes : 1 M
	public class ArrayOperations {
	public static void main(String[] args) {
	try {
	// Inner try block handles specific exceptions related to array operations
	try {
	// Correcting the null reference issue by initializing the array
		2	CO4	3

	int[] numbers = null; // No error here, assigns a null array
	
	// throws null pointer exception
	numbers[0] = 10;
	
	// Trying to access an element at index 2, which exists
	System.out.println(numbers[2]); // Thows array index out of bounds exception
	
	} catch (ArrayIndexOutOfBoundsException e) {
	// Catching and handling ArrayIndexOutOfBoundsException if it occurs
	System.out.println("Array index is out of bounds");
	rethrow e;
	// No need to rethrow the exception, as it's already handled here
	} catch (NullPointerException e) {
	// Adding NullPointerException handling in case the array is null
	System.out.println("Array is null");
	rethrow e;
	}
	} catch (Exception e) {
	// to catch the rethrown exception the super class to be mentioned in the outer catch
	System.out.println("An error occurred: " + e);
	}
	}
	}			

19	Demonstrate how a nested interface can be implemented. **Interface definition 1 M and class implementation 1 M**	2	CO2	2

	interface If1{
	interface If2{
	void display();
	}
	void method1();
	}
	class A implements If1.If2{
	public void display(){
	System.out.println(“Nested interface method called….”);
	}}
	class NestedInfDemo{
	public static void main(String[] args){
	If1.If2 iref = new A();
	iref.display();
	}}			