USN | | | | | | | | | | |

Internal Assessment Test I – Dec 2023

Sub:	OOPS WITH JAVA					Sub Code:	BCS306A	Branch:	CSE
Date:	19/12/23	Duration:	90 mins	Max Marks:	50	Sem/Sec:	III A,B,C	OBE	

Answer any FIVE FULL Questions

		MARKS	CO	RBT
1. a)	Create a Java program to print the following pattern using nested loops: 1 12 123 1234	[3]	CO1	L2
1 b)	Create a scenario involving multiple classes that showcases the concept of inheritance. Implement a base class and a derived class with overridden methods.	[3]	CO2	L2, L3
1 c)	Write a program that calculates the compound interest using the formula A = P(1 + r/n)^(nt), where: A is the amount after n years, P is the principal amount, r is the annual interest rate (as a decimal), n is the number of times interest is compounded per year, t is the number of years.	[4]	CO1	L3
2 a)	Implement a program that simulates a simple calculator. Take user input for two numbers and an operator (+, -, *, /). Use a switch statement to perform the corresponding operations.	[3]	CO1	L3
2 b)	Write a program that uses the enhanced for loop to iterate over an array of integers. If the loop encounters a negative number, break out of the loop and print the sum of the positive numbers encountered so far.	[4]	CO1	L3
2 c)	Differentiate between literals and variables. Provide examples of different types of literals in Java.	[3]	CO1	L2
3 a)	Define constructors and types of constructors in Java and their purpose. - Explore the use of the 'this' keyword and its significance.	[4]	CO2	L1

3 b)	class Simple{
	int a = 10;
	String s1 = “CMRIT”;
	public static void main(String args[]){
	System.out.println("Hello Java");
	} }
	Based on the above code write the names of identifiers, Literals, types of literals, data types, and access specifier present in this code.
		[3]	CO1	L3

3 c)	Explain the process of argument passing in Java methods. Differentiate between pass-by-value and pass-by-reference.	[3]	CO2	L2
4 a)	Write a program for garbage collection in Java and its impact on memory management.	[4]	CO1	L2
4 b)	Write a program in java to calculate the factorial of a number using recursion concept.	[3]	CO1	L3
4 c)	Discuss the concept of returning objects from methods in Java. Provide an example to illustrate this process.	[3]	CO2	L1
5 a)	Discuss access control in Java classes, including public, private, protected, and default access modifiers	[3]	CO2	L2
5 b)	Implement a Java method that checks if a number is a prime number. Test the method with various inputs.	[4]	CO3	L3
5 c)	Implement a program that prints the first 10 numbers of the Fibonacci series using a for loop.	[3]	CO2	L3
6 a)	Your program involves creating multiple instances of a class representing employees. Explain how you would declare objects, assign object reference variables, and manage these instances.	[3]	CO1	L2

6 b)	Will the code successfully compile? What will be the output.
	public class A {
	int x = 20;
	}
	public class B extends A { int x = 30;}
	public class Test {
	public static void main(String[] args)
	{
	B b = new B();
	System.out.println(b.x);
	
	A a = new A();
	System.out.println(a.x);
	
	A a2 = new B();
	System.out.println(a2.x);
	}
	}
		[3]	CO3	L3

6 c)	Write a java program to define a base class Animal with a method makeSound(), and then it creates two subclasses Dog and Cat that override the makeSound() method. Finally, in the AnimalTest class, objects of Animal, Dog, and Cat are created and their makeSound() methods are called.	[4]	CO3	L3

CI **CCI** **HOD**

CO PO Mapping

Course Outcomes		Modules covered	PO1	PO2	PO3	PO4	PO5	PO6	PO7	PO8	PO9	PO10	PO11	PO12	PSO1	PSO2	PSO3	PSO4
CO1	Describe the characteristics of Graphics Interface and its principles.	**1 & 2**	2	3	-	3	-	3	-	-	-	-	-	-	-	-	3	2
CO2	Analyze, design and evaluate user interface design	**1,2,3,4 & 5**	2	3	3	3	-	3	-	-	-	-	-	-	-	-	3	2
CO3	Explain the components of web systems	**2,3 & 4**	2	3	2	3	-	3	-	-	-	-	-	-	-	-	3	2
CO4	Demonstrate the guidelines of multimedia.	**2,3 & 4**	2	3	2	3	-	3	-	-	-	-	-	-	-	-	3	2
CO5	Understand the prototype and kinds of test	**5**	2	3	2	3	-	3	-	-	-	-	-	-	-	-	3	2

COGNITIVE LEVEL	REVISED BLOOMS TAXONOMY KEYWORDS
L1	List, define, tell, describe, identify, show, label, collect, examine, tabulate, quote, name, who, when, where, etc.
L2	summarize, describe, interpret, contrast, predict, associate, distinguish, estimate, differentiate, discuss, extend
L3	Apply, demonstrate, calculate, complete, illustrate, show, solve, examine, modify, relate, change, classify, experiment, discover.
L4	Analyze, separate, order, explain, connect, classify, arrange, divide, compare, select, explain, infer.
L5	Assess, decide, rank, grade, test, measure, recommend, convince, select, judge, explain, discriminate, support, conclude, compare, summarize.

PROGRAM OUTCOMES (PO), PROGRAM SPECIFIC OUTCOMES (PSO)				CORRELATION LEVELS	
PO1	Engineering knowledge	PO7	Environment and sustainability	0	No Correlation
PO2	Problem analysis	PO8	Ethics	1	Slight/Low
PO3	Design/development of solutions	PO9	Individual and team work	2	Moderate/ Medium

PO4	Conduct investigations of complex problems	PO10	Communication	3	Substantial/ High
PO5	Modern tool usage	PO11	Project management and finance		
PO6	The Engineer and society	PO12	Life-long learning		
PSO1	Develop applications using different stacks of web and programming technologies				
PSO2	Design and develop secure, parallel, distributed, networked, and digital systems				
PSO3	Apply software engineering methods to design, develop, test and manage software systems.				
PSO4	Develop intelligent applications for business and industry				

USN | | | | | | | | | | |

Internal Assessment Test 1 – December 2023

Sub:	OOPS WITH JAVA					Sub Code:	BCS306A	Branch :	CSE	
Date:	19-12-2023	Duration:	90 mins	Max Marks:	50	Sem / Sec:	III(A, B & C)		OBE	

Answer any FIVE FULL Questions

		MARKS	CO	RBT

1 (a)	Create a Java program to print the following pattern using nested loops:
	1
	12
	123
	1234
	SOLUTION:
	public class PatternPrint {
	public static void main(String[] args) {
	int rows = 4; // You can change this value to adjust the number of rows
	// Nested loops to print the pattern
	for (int i = 1; i <= rows; i++) {
	for (int j = 1; j <= i; j++) {
	System.out.print(j);
	}
	System.out.println(); // Move to the next line after each row
	}
	}
	}
		[3]	CO1	L2

1 (b)	Create a scenario involving multiple classes that showcases the concept of inheritance. Implement a base class and a derived class with overridden methods.
	SOLUTION:
	class Animal {
	void sound() {
	System.out.println("Animal makes a sound");
	}
	}
	class Dog extends Animal {
	@Override
		[3]	CO2	L2, L3

	void sound() {
	System.out.println("Dog barks");
	}
	
	void wagTail() {
	System.out.println("Dog wags its tail");
	}
	}
	
	public class InheritanceExample {
	public static void main(String[] args) {
	// Create an instance of the base class
	Animal animal = new Animal();
	animal.sound();
	
	System.out.println();
	
	// Create an instance of the derived class
	Dog dog = new Dog();
	dog.sound(); // This will call the overridden method in Dog class
	dog.wagTail(); // This is a method specific to the Dog class
	}
	}			

1(c)	Write a program that calculates the compound interest using the formula
	A = P(1 + r/n)^(nt), where:
	A is the amount after n years,
	P is the principal amount,
	r is the annual interest rate (as a decimal),
	n is the number of times interest is compounded per year,
	t is the number of years.
	SOLUTION:
	import java.util.Scanner;
	public class CompoundInterestCalculator {
	public static void main(String[] args) {
	Scanner scanner = new Scanner(System.in);
	System.out.print("Enter the principal amount (P): ");
	double principal = scanner.nextDouble();
	System.out.print("Enter the annual interest rate (as a decimal) (r): ");
	double annualRate = scanner.nextDouble();
	System.out.print("Enter the number of times interest is compounded per year (n): ");
	int compoundFrequency = scanner.nextInt();
		[4]	CO1	L3

	System.out.print("Enter the number of years (t): ");
	int years = scanner.nextInt();
	double interestRatePerCompoundingPeriod = annualRate / compoundFrequency;
	int totalCompoundingPeriods = compoundFrequency * years;
	double compoundInterest = principal * Math.pow(1 + interestRatePerCompoundingPeriod, totalCompoundingPeriods) - principal;
	System.out.println("Compound Interest after " + years + " years: " + compoundInterest);
	scanner.close();
	}
	}			

2 (a)	Implement a program that simulates a simple calculator. Take user input for two numbers and an operator (+, -, *, /). Use a switch statement to perform the corresponding operations.
	SOLUTION:
	import java.util.Scanner;
	
	public class SimpleCalculator {
	public static void main(String[] args) {
	Scanner scanner = new Scanner(System.in);
	
	// Input the first number
	System.out.print("Enter the first number: ");
	double num1 = scanner.nextDouble();
	
	// Input the operator
	System.out.print("Enter the operator (+, -, *, /): ");
	char operator = scanner.next().charAt(0);
	
	// Input the second number
	System.out.print("Enter the second number: ");
	double num2 = scanner.nextDouble();
	
	// Perform the calculation based on the operator
	double result = 0;
	switch (operator) {
	case '+':
	result = num1 + num2;
	break;
	case '-':
	result = num1 - num2;
	break;
	case '*':
	result = num1 * num2;
		[3]	CO1	L3

	break;
	case '/':
	if (num2 != 0) {
	result = num1 / num2;
	} else {
	System.out.println("Error: Cannot divide by zero.");
	return;
	}
	break;
	default:
	System.out.println("Error: Invalid operator.");
	return;
	}
	
	// Display the result
	System.out.println("Result: " + result);
	
	scanner.close();
	}
	}			

2(b)	Write a program that uses the enhanced for loop to iterate over an array of integers. If the loop encounters a negative number, break out of the loop and print the sum of the positive numbers encountered so far.
	SOLUTION:
	public class ArraySumWithBreak {
	public static void main(String[] args) {
	// Sample array of integers
	int[] numbers = {1, 5, -3, 8, 2, -7, 10};
	
	// Variable to store the sum of positive numbers
	int sum = 0;
	
	// Enhanced for loop to iterate over the array
	for (int number : numbers) {
	// Check if the number is negative
	if (number < 0) {
	break; // Break out of the loop if a negative number is encountered
	}
	
	// Add the positive number to the sum
	sum += number;
	}
	
	// Print the sum of positive numbers encountered so far
	System.out.println("Sum of positive numbers: " + sum);
	}
	}
		[4]	CO1	L3

2(c)	Differentiate between literals and variables. Provide examples of different types of literals in Java. **SOLUTION:** **Literals**: In programming, a literal is a notation representing a fixed value in the source code. It is a constant value that is used directly in the code without being computed or assigned a variable name. In other words, literals are data given in a variable or constant. **Variables**: Variables, on the other hand, are containers or storage locations identified by a memory address and an associated symbolic name (an identifier). Unlike literals, variables can vary; their values can be changed during the execution of a program. **Types of Literals:** **1. Integer Literal** **2. Float literal** **3. character literal** **4. String literal**	[3]	CO1	L2
3(a).	Define constructors and types of constructors in Java and their purpose. - Explore the use of the 'this' keyword and its significance. **SOLUTION:** **Constructors** in Java: A constructor in Java is a special type of method that is used to initialize objects. It has the same name as the class and doesn't have a return type. When an object is created using the new keyword, a constructor is called automatically to initialize the object. Constructors are used to set initial values for object attributes or perform any setup needed for the object. **Types of Constructors in Java:** **1. Default Constructor:** A constructor with no parameters is called the default constructor. If a class doesn't have any constructor defined, Java provides a default constructor automatically. It initializes the attributes to their default values (e.g., 0 for numeric types, null for objects). **2. Parameterized Constructor:** A constructor with parameters is called a parameterized constructor. It allows you to initialize object attributes with specific values at the time of object creation. **3. Copy Constructor:** A constructor that takes an object of the same class as a parameter is called a copy constructor.It creates a new object by copying the values of another object. The **'this'** Keyword: In Java, the this keyword is a reference variable that refers to the current object. It is used to differentiate between instance variables and local variables when they have the same name	[4]	CO2	L1

3(b)	class Simple{
	int a = 10;
	String s1 = “CMRIT”;
		[3]	CO1	L3

	public static void main(String args[]){ System.out.println("Hello Java"); } } Based on the above code write the names of identifiers, Literals, types of literals, data types, and access specifier present in this code. **SOLUTION:** **Identifiers:** Simple (class name) a (variable name) s1 (variable name) main (method name) args (parameter name) **Literals**: 10 (integer literal assigned to variable a) "CMRIT" (string literal assigned to variable s1) "Hello Java" (string literal passed to System.out.println method) **Types of Literals:** Integer Literal (10) String Literal ("CMRIT" and "Hello Java") **Data Types:** int (data type for variable a) String (data type for variable s1) String[] (data type for parameter args in the main method) **Access Specifier:** public (access specifier for the main method)			
3(c)	Explain the process of argument passing in Java methods. Differentiate between pass-by-value and pass-by-reference. **SOLUTION:** **Pass-by-Value in Java:** In Java, when you pass a primitive data type or an object reference to a method, you are passing a copy of the value. For primitive data types (e.g., int, char), the actual value is passed. For objects, the reference (memory address) to the object is passed, not the object itself. **Pass-by-Reference (Not Applicable in Java):** Java strictly follows the pass-by-value mechanism. The term "pass-by-reference" implies passing the actual reference to the variable, allowing changes to the original variable. This is not how Java works. While it may seem like objects are being passed by reference, what is actually passed is a copy of the reference, not the reference itself. Therefore, Java is more accurately described as "pass-by-value."	[3]	CO2	L2
4(a)	Write a program for garbage collection in Java and its impact on memory management. **SOLUTION:**	[4]	CO1	L2

	Program:
	public class TestGarbage1{
	public void finalize()
	{
	System.out.println("object is garbage collected");
	}
	public static void main(String args[]){
	TestGarbage1 s1=new TestGarbage1();
	TestGarbage1 s2=new TestGarbage1();
	s1=null;
	s2=null;
	System.gc();
	}
	}
	
	Impact:
	Automatic Management: Garbage collection in Java automatically reclaims memory occupied by objects that are no longer in use.
	Prevents Leaks: It helps prevent memory leaks by deallocating memory from unreferenced objects.
	Enhances Productivity: Developers can focus on application logic without manual memory allocation concerns.
	Dynamic Allocation: Java allows dynamic memory allocation, and the garbage collector handles deallocation.
	Performance Impact: While introducing some overhead, modern garbage collectors are designed for minimal impact on application performance.			

4(b)	Write a program in java to calculate the factorial of a number using recursion concept.
	SOLUTION:
	import java.util.Scanner;
	public class FactorialCalculator {
	public static void main(String[] args) {
	Scanner scanner = new Scanner(System.in);
	
	// Input a non-negative integer
	System.out.print("Enter a non-negative integer: ");
	int number = scanner.nextInt();
	
	// Check for a non-negative input
	if (number < 0) {
	System.out.println("Factorial is undefined for negative numbers.");
	} else {
	// Calculate and display the factorial
	long factorial = calculateFactorial(number);
	System.out.println("Factorial of " + number + " is: " + factorial);
	}
		[3]	CO1	L3

	scanner.close();
	}
	
	// Recursive method to calculate factorial
	private static long calculateFactorial(int n) {
	if (n == 0 || n == 1) {
	return 1; // Base case: factorial of 0 and 1 is 1
	} else {
	return n * calculateFactorial(n - 1); // Recursive case
	}
	}
	}			

4(c)	Discuss the concept of returning objects from methods in Java. Provide an example to illustrate this process.
	SOLUTION:
	In Java, methods can return objects just like they can return primitive data types. This allows for flexibility in designing classes and methods, enabling the creation of instances within a method and returning those instances to the caller.
	Example:
	Class Test{
	Test reMetod()
	{
	Test t = new Test();
	return t;
	}
	Public static void main(String args[])
	{
	Test obj = new Test();
	Test obj1 = obj.reMethos();
	}
	}
		[3]	CO2	L1

5 (a)	Discuss access control in Java classes, including public, private, protected, and default access modifiers **SOLUTION:** **Public (public):** A class, method, or variable declared with the public modifier is accessible from any other class or package. It has the broadest visibility. Example: public class MyClass {...} **Private (private):** A class, method, or variable declared with the private modifier is only accessible within the same class.	[3]	CO2	L2

	It provides the highest level of encapsulation. Example: private int myVariable; **Protected (protected):** A class, method, or variable declared with the protected modifier is accessible within the same class, package, and subclasses (even if they are in a different package). It is more restrictive than public but less restrictive than private. Example: protected void myMethod() {...} **Default (Package-Private):** If no access control modifier is specified (i.e., the default), the class, method, or variable is accessible only within the same package. It provides a level of visibility between public and private. Example: class MyClass {...} (without any modifier)			

5 (b)	Implement a Java method that checks if a number is a prime number. Test the method with various inputs.
	SOLUTION:
	`import java.util.Scanner;`
	
	`public class PrimeNumberChecker {`
	
	`public static void main(String[] args) {`
	`Scanner scanner = new Scanner(System.in);`
	
	`// Input a number`
	`System.out.print("Enter a number: ");`
	`int number = scanner.nextInt();`
	
	`// Check if the number is prime and display the result`
	`if (isPrime(number)) {`
	`System.out.println(number + " is a prime number.");`
	`} else {`
	`System.out.println(number + " is not a prime number.");`
	`}`
	
	`scanner.close();`
	`}`
	
	`// Method to check if a number is a prime number`
	`public static boolean isPrime(int number) {`
	`if (number <= 1) {`
	`return false; // 0 and 1 are not prime numbers`
	`}`
	
	`// Check for divisibility from 2 to half of the number`
	`for (int i = 2; i <= number / 2; i++) {`
	`if (number % i == 0) {`
	`return false; // The number is divisible, so it's not a prime number`
	`}`
	`}`
		[4]	CO3	L3

	return true; // The number is prime
	}
	}			

5(c)	Implement a program that prints the first 10 numbers of the Fibonacci series using a for loop.
	SOLUTION:
	public class FibonacciSeries {
	public static void main(String[] args) {
	// Define the number of terms to generate
	int n = 10;
	
	// Print the first 10 numbers of the Fibonacci series
	System.out.println("Fibonacci Series (first 10 numbers):");
	for (int i = 0; i < n; i++) {
	System.out.print(fibonacci(i) + " ");
	}
	}
	
	// Method to calculate the Fibonacci series for a given term
	public static int fibonacci(int term) {
	if (term == 0) {
	return 0;
	} else if (term == 1) {
	return 1;
	} else {
	int a = 0, b = 1, result = 0;
	
	for (int i = 2; i <= term; i++) {
	result = a + b;
	a = b;
	b = result;
	}
	
	return result;
	}
	}
	}
		[3]	CO2	L3

6(a)	Your program involves creating multiple instances of a class representing employees. Explain how you would declare objects, assign object reference variables, and manage these instances.
	SOLUTION:
	public class Employee {
	private String name;
	private int employeeId;
	
	// Constructor
	public Employee(String name, int employeeId) {
	this.name = name;
		[3]	CO1	L2

	this.employeeId = employeeId;
	}
	
	// Getter methods (not shown for brevity)
	
	public static void main(String[] args) {
	// Declare and create objects of the Employee class
	Employee employee1 = new Employee("John Doe", 101);
	Employee employee2 = new Employee("Jane Smith", 102);
	
	// Assign object reference variables
	Employee manager = employee1;
	
	// Access and manage instances
	System.out.println("Employee 1: " + employee1.getName() + ", ID: " + employee1.getEmployeeId());
	System.out.println("Employee 2: " + employee2.getName() + ", ID: " + employee2.getEmployeeId());
	
	System.out.println("Manager: " + manager.getName() + ", ID: " + manager.getEmployeeId());
	
	// Modify an instance
	employee1.setName("John Updated");
	
	// Display updated information
	System.out.println("Updated Employee 1: " + employee1.getName() + ", ID: " + employee1.getEmployeeId());
	}
	}			

6(b)	Will the code successfully compile? What will be the output.
	public class A {
	int x = 20;
	}
	public class B extends A { int x = 30;}
	public class Test {
	public static void main(String[] args)
	{
	B b = new B();
	System.out.println(b.x);
	
	A a = new A();
	System.out.println(a.x);
	
	A a2 = new B();
	System.out.println(a2.x);
	}
	}
	SOLUTION:
	
	Yes
	Output 30
		[3]	CO3	L3

	20 20			

6(c)	Write a java program to define a base class Animal with a method makeSound(), and then it creates two subclasses Dog and Cat that override the makeSound() method. Finally, in the AnimalTest class, objects of Animal, Dog, and Cat are created and their makeSound() methods are called.
	SOLUTION:
	// Base class
	class Animal {
	// Method to make a sound
	public void makeSound() {
	System.out.println("Generic animal sound");
	}
	}
	
	// Subclass Dog
	class Dog extends Animal {
	// Override makeSound() for Dog
	@Override
	public void makeSound() {
	System.out.println("Dog barks");
	}
	}
	
	// Subclass Cat
	class Cat extends Animal {
	// Override makeSound() for Cat
	@Override
	public void makeSound() {
	System.out.println("Cat meows");
	}
	}
	
	// Test class
	public class AnimalTest {
	public static void main(String[] args) {
	// Create objects of Animal, Dog, and Cat
	Animal genericAnimal = new Animal();
	Dog myDog = new Dog();
	Cat myCat = new Cat();
	
	// Call makeSound() for each object
	System.out.println("Sound from generic animal:");
	genericAnimal.makeSound();
	
	System.out.println("\nSound from a dog:");
	myDog.makeSound();
		[4]	CO3	L3

	System.out.println("\nSound from a cat:");
	myCat.makeSound();
	}
	}			

CI **CCI** **HOD**

PO Mapping

CO-PO and CO-PSO Mapping																				
Course Outcomes		Blooms Level	Modules covered	PO1	PO2	PO3	PO4	PO5	PO6	PO7	PO8	PO9	PO10	PO11	PO12	PSO1	PSO2	PSO3	PSO4
CO1	Demonstrate proficiency in writing simple programs involving branching and looping structures.	L1-L3	1	2	3	3	2	3	0	0	0	2	0	0	2	2	0	2	2
CO2	Design a class involving data members and methods for the given scenario.	L1-L3	2	2	3	3	2	3	0	0	0	2	0	0	2	2	0	2	2
CO3	Apply the concepts of inheritance and interfaces in solving real world problems.	L1-L3	3	2	3	3	2	3	0	0	0	2	0	0	2	2	0	2	2
CO4	Use the concept of packages and exception handling in solving complex problem	L3	4	2	3	3	2	3	0	0	0	2	0	0	2	2	0	2	2
CO5	Apply concepts of multithreading, auto boxing and enumerations in program development	L3	5	2	3	3	2	3	0	0	0	2	0	0	2	2	0	2	2

COGNITIVE LEVEL	REVISED BLOOMS TAXONOMY KEYWORDS
L1	List, define, tell, describe, identify, show, label, collect, examine, tabulate, quote, name, who, when, where, etc.
L2	summarize, describe, interpret, contrast, predict, associate, distinguish, estimate, differentiate, discuss, extend

L3	Apply, demonstrate, calculate, complete, illustrate, show, solve, examine, modify, relate, change, classify, experiment, discover.
L4	Analyze, separate, order, explain, connect, classify, arrange, divide, compare, select, explain, infer.
L5	Assess, decide, rank, grade, test, measure, recommend, convince, select, judge, explain, discriminate, support, conclude, compare, summarize.

PROGRAM OUTCOMES (PO), PROGRAM SPECIFIC OUTCOMES (PSO)				CORRELATION LEVELS	
PO1	Engineering knowledge	PO7	Environment and sustainability	0	No Correlation
PO2	Problem analysis	PO8	Ethics	1	Slight/Low
PO3	Design/development of solutions	PO9	Individual and team work	2	Moderate/ Medium
PO4	Conduct investigations of complex problems	PO10	Communication	3	Substantial/ High
PO5	Modern tool usage	PO11	Project management and finance		
PO6	The Engineer and society	PO12	Life-long learning		
PSO1	Develop applications using different stacks of web and programming technologies				
PSO2	Design and develop secure, parallel, distributed, networked, and digital systems				
PSO3	Apply software engineering methods to design, develop, test and manage software systems.				
PSO4	Develop intelligent applications for business and industry				