USN										

Internal Assessment Test 1 – Set 2 – December 2023

Sub:	Object Oriented Programming with Java					Sub Code:	BCS306A	Branch	ISE
Date:	20/12/2023	Duration:	90 min's	Max Marks:	50	Sem/Sec:	III / A, B & C		OBE

Solution

		MARKS	CO	RBT
1a	How is Java different from C programming language?	2		
Ans			CO1	L1

	POP	OOP
Program Organization	Program is divided into small parts called functions	Program is divided into small parts called objects
Importance	Importance is not give to data but to functions	Importance is give to data rather than procedures
Approach	POP follows top down approach	OOP follows bottom up approach
Access Specifier	Does not have any access specifier	Has three access specifiers namely public, private and protected
Data Moving	Data can move freely from function to function in the system	Objects can move and communicate with each other
Maintainability	To add new data and function it is not easy	Provides an easy way to add new data and functions
Data Access	Function uses global data for sharing that can be accessed freely from function to function in the system	Objects use local data and can be accessed in a control manner.

(any 4 correct points carries 2 marks)

1b.	Briefly explain the core characteristics of object oriented programming.	8		
Ans			CO1	L1

Object-oriented programming (OOP) is a programming paradigm that uses objects, which are instances of classes, to organize and structure code. The core characteristics of object-oriented programming include:

Encapsulation: [brief Explanation carries **2 Marks**]

Encapsulation is wrapping of data and function or method into a single unit. It is the mechanism that binds together code and data it manipulates, and keeps both safe from outside interference and misuse. Encapsulation is a protective wrapper that prevents code and data from being arbitrarily accessed by other code defined outside the wrapper. Access to the code and data inside the wrapper is tightly controlled through a well-defined interface. The power of encapsulated code is that everyone knows how to access it and thus can use it regardless of the implementation details and without fear of unexpected side effects.

Data Abstraction: [brief Explanation carries **2 Marks**]

Abstraction means displaying only essential information and hiding the details. Data abstraction refers to providing only essential information about the data to the outside world, hiding the background details or implementation.

3 Inheritance: [brief Explanation carries **2 Marks]**

Inheritance is the process by which one object acquires the properties of another

	object. Inheritance supports the concept of hierarchical classification. For example, a Golden Retriever belongs to the class - dog, dog in turn is part of the class mammal, and mammal is under the larger class animal. Mammal is called the subclass of animals and animals is called the mammal's superclass. 4 **Polymorphism**: [brief Explanation carries **2 Marks**] Polymorphism, as the name suggests, is the phenomena by virtue of which the same entity can exist in two or more forms. In OOPS, functions can be made to exhibit polymorphic behaviour. Functions with different set of formal arguments can have the same name. Polymorphism is of two types: static and dynamic			
2a	What is type casting? Illustrate with an example, the meaning of automatic type casting?	4	CO1	L2

Ans	Type casting**:[1M]**
	It is often necessary to store a value of one type into the variable of another type. In these situations the value that to be stored should be casted to destination type. Assigning a value of one type to a variable of another type is known as Type Casting .Type casting can be done in two ways.
	In Java, type casting is classified into two types,
	1. Widening Casting(Implicit)
	2. Narrowing Casting(Explicitly done)
	
	Automatic type casting**:[1M]**
	When one type of data is assigned to another type of variable, an automatic type conversion will take place if the following two conditions are met:
	• The two types are compatible.
	• The destination type is larger than the source type.
	When these two conditions are met, a widening conversion takes place. For example, the int type is always large enough to hold all valid byte values, so no explicit cast statement is required. For widening conversions, the numeric types, including integer and floating-point types, are compatible with each other. However, there are no automatic conversions from the numeric types to char or boolean. Also, char and boolean are not compatible with each other. As mentioned earlier, Java also performs an automatic type conversion when storing a literal integer constant into variables of type byte, short, long, or char.
	
	Example**:[2M]**
	class Promote {
	public static void main(String args[]) {
	byte b = 42;
	char c = 'a';
	short s = 1024;
	int i = 50000;
	float f = 5.67f;
	double d = .1234;			

	double result = (f * b) + (i / c) - (d * s);
	System.out.println((f * b) + " + " + (i / c) + " - " + (d * s));
	System.out.println("result = " + result);
	}
	}			

2b	Briefly explain the different features of Java with suitable examples.	6	CO1	L2
Ans	**Simple:** Java was designed to be easy for the professional programmer to learn and use effectively. Assuming that you have some programming experience, you will not find Java hard to master. If you already understand the basic concepts of object-oriented programming, learning Java will be even easier. Best of all, if you are an experienced C++ programmer, moving to Java will require very little effort. Because Java inherits the C/C++ syntax and many of the object-oriented features of C++, most programmers have little trouble learning Java. **Object-Oriented:** Although influenced by its predecessors, Java was not designed to be source-code compatible with any other language. This allowed the Java team the freedom to design with a blank slate. One outcome of this was a clean, usable, pragmatic approach to objects. Borrowing liberally from many seminal object-software environments of the last few decades, Java manages to strike a balance between the purist's "everything is an object" paradigm and the pragmatist's "stay out of my way" model. The object model in Java is simple and easy to extend, while primitive types, such as integers, are kept as high-performance non objects. **Robust:** The multiplatform environment of the Web places extraordinary demands on a program, because the program must execute reliably in a variety of systems. Thus, the ability to create robust programs was given a high priority in the design of Java. To gain reliability, Java restricts you in a few key areas to force you to find your mistakes early in program development. At the same time, Java frees you from having to worry about many of the most common causes of programming errors. Because Java is a strictly typed language, it checks your code at compile time. However, it also checks your code at run time. Many hard-to-track-down bugs that often turn up in hard-to-reproduce run-time situations are simply impossible to create in Java. Knowing that what you have written will behave in a predictable way under diverse conditions is a key feature of Java. **Multithreaded:** Java was designed to meet the real-world requirement of creating interactive, networked programs. To accomplish this, Java supports multithreaded programming, which allows you to write programs that do many things simultaneously. The Java run-time system comes with an elegant yet sophisticated solution for multiprocess synchronization that enables you to construct smoothly running interactive systems. Java's easy-to-use approach to multithreading allows you to think about the specific behavior of your program, not the multitasking subsystem. **Distributed:** Java is designed for the distributed environment of the Internet because it handles TCP/IP protocols. In fact, accessing a resource using a URL is not much different			

from accessing a file. Java also supports Remote Method Invocation (RMI). This feature enables a program to invoke methods across a network.

Dynamic:
Java programs carry with them substantial amounts of run-time type information that is used to verify and resolve accesses to objects at run time. This makes it possible to dynamically link code in a safe and expedient manner. This is crucial to the robustness of the Java environment, in which small fragments of bytecode may be dynamically updated on a running system.

[any 6 points carries 6M]

3a	Write a Java program to calculate the average among the elements {8, 6, 2, 7} using *for each* looping.	7	CO1	L1

Ans

```
public class AverageCalculator {    //[creating class carries 1M]
   public static void main(String[] args) {
      // Array of elements
      int[] numbers = {8, 6, 2, 7};//[declaration of variable carries 2M]

      // Calculate the sum using a for-each loop
      int sum = 0;
      for (int number : numbers) {
         sum += number;
      }//[using for each carries 2M]

      // Calculate the average
      double average = (double) sum / numbers.length;

      // Display the result
      System.out.println("Elements: " + java.util.Arrays.toString(numbers));
      System.out.println("Sum: " + sum);
      System.out.println("Average: " + average);
     //[printing appropriate output carries 2M]
   }
}
```

3b	How is *for each* different from *for* loop?	3	CO1	L1

Ans

The enhanced for loop (for-each loop) in Java is designed to simplify the process of iterating over collections or arrays. Here are three key differences between the for-each loop and the traditional for loop:

Normal for-loop	**for-each(Enhanced for loop)**
This for-loop is present from JDK1	This for loop is present from JDK5
In a normal for-loop, we can increase the counter as per our wish by using i=i+x(where x is any constant x=1,2,3…)	But enhanced for loop will execute in a sequential manner i.e counter will always increase by one.
Using this for loop we can iterate on any container object.	We can only iterate on that container by using this loop to implement the iterable interface.
In this for-loop, we can iterate in both decrement or increment order.	But in this for-loop, we can iterate only in increment order.

	In this for-loop, we can replace elements at any specific index.	But in this for-loop, we don't have access to the index, so we cannot replace elements at any specific index.			
	By using normal for-loop we can print array elements either in the original order or in reverse order.	But in the for-each loop, we can print array element only in the original order, not in reverse order			

	Example: Printing element in a 1D array
	int[ ] x={1,2,3};
	for(int i=0;i<x.length;i++){
	System.out.println(x[i]);
	}
		Example: Printing element in a 1D array using for-each loop
		int[ ] x={1,2,3};
		for(int a : x){
		System.out.println(a);
		}			

	[any 3 points carries 3M]				
4	How to declare and accept values for two dimensional arrays in Java? Explain with a suitable example.		10	CO1	L2

Ans	In Java, multidimensional arrays are actually arrays of arrays. These, as you might expect, look and act like regular multidimensional arrays. However, as you will see, there are a couple of subtle differences. To declare a multidimensional array variable, specify each additional index using another set of square brackets.
	
	For example, the following declares a two-dimensional array variable called twoD:
	int twoD[][] = new int[4][5];
	This allocates a 4 by 5 array and assigns it to twoD. Internally, this matrix is implemented as an array of arrays of int.
	
	The following program numbers each element in the array from left to right, top to bottom, and then displays these values:
	
	// Demonstrate a two-dimensional array.
	class TwoDArray {
	public static void main(String args[]) {
	int twoD[][]= new int[4][5];
	int i, j, k = 0;
	for(i=0; i<4; i++)
	for(j=0; j<5; j++) {
	twoD[i][j] = k;
	k++;
	}
	for(i=0; i<4; i++) {
	for(j=0; j<5; j++)
	System.out.print(twoD[i][j] + " ");
	System.out.println();
	}
	}
	}				

5a	Are the below statements valid? I)for (; ;) ii) for(int a:arr[]) iii) while() { loop: a=b+c/d; } if(condition==TRUE) loop;		3	CO1	L3

Ans	i) Valid ii) Invalid iii) Invalid			
5b	List the different iteration statements used in Java and briefly explain them with suitable examples	7		

Ans	Java's iteration statements are for, while, and do-while. These statements create what we commonly call loops. As you probably know, a loop repeatedly executes the same set of instructions until a termination condition is met. As you will see, Java has a loop to fit any programming need.
	[listing the statements carries 1M]
	
	While:[explanation 1M+Example 1M]
	The while loop is Java's most fundamental loop statement. It repeats a statement or block while its controlling expression is true. Here is its general form:
	`while(condition) {`
	`// body of loop`
	`}`
	The condition can be any Boolean expression. The body of the loop will be executed as long as the conditional expression is true. When condition becomes false, control passes to the next line of code immediately following the loop. The curly braces are unnecessary if only a single statement is being repeated.
	
	Here is a while loop that counts down from 10, printing exactly ten lines of "tick":
	`// Demonstrate the while loop.`
	`class While {`
	`public static void main(String args[]) {`
	`int n = 10;`
	`while(n > 0) {`
	`System.out.println("tick " + n);`
	`n--;`
	`}`
	`}`
	`}`
	
	do-while:[2M]
	there are times when you would like to test the termination expression at the end of the loop rather than at the beginning. Fortunately, Java supplies a loop that does just that: the do-while. The do-while loop always executes its body at least once, because its conditional expression is at the bottom of the loop. Its general form is
	`do {`
	`// body of loop`
	`} while (condition);`
	Each iteration of the do-while loop first executes the body of the loop and then evaluates the conditional expression. If this expression is true, the loop will repeat. Otherwise, the loop terminates. As with all of Java's loops, condition must be a Boolean expression.
	`// Demonstrate the do-while loop.`
	`class DoWhile {`
	`public static void main(String args[]) {`
	`int n = 10;`
	`do {`
	`System.out.println("tick " + n);`
	`n--;`
	`} while(n > 0);`
	`}`			

	}
	
	for[2M]
	there are two forms of the for loop. The first is the traditional form
	that has been in use since the original version of Java. The second is the newer "for-each" form.
	Both types of for loops are discussed here, beginning with the traditional form.
	Here is the general form of the traditional for statement:
	for(initialization; condition; iteration) {
	// body
	}
	If only one statement is being repeated, there is no need for the curly braces.			

6	A class called EMPLOYEE, which models the employeesof an organization and identified with *id*, *name* and *salary*. The method *incrementSalary(percentage)* increases the salary of the employee by the computed percentage. The calculated percentage to be sent depends on the grade of the employee received as input from the user. Grade A-3%, B-5%, C-7% and D-10%.Develop an EMPLOYEE class with suitable *main()* method using *Java* programming language.	10	CO2	L3

Ans	import java.util.Scanner;
	class EMPLOYEE {
	private int id;
	private String name;
	private double salary;
	
	public EMPLOYEE(int id, String name, double salary) {
	this.id = id;
	this.name = name;
	this.salary = salary;
	}
	
	public void incrementSalary(char grade) {
	double percentage = 0.0;
	switch (grade) {
	case 'A':
	percentage = 3.0;
	break;
	case 'B':
	percentage = 5.0;
	break;
	case 'C':
	percentage = 7.0;
	break;
	case 'D':
	percentage = 10.0;
	break;
	default:
	System.out.println("Invalid grade. Salary remains unchanged.");
	return;
	}
	
	double incrementAmount = salary * (percentage / 100);
	salary += incrementAmount;			

```
        System.out.println("Salary incremented by " + percentage + "%. New salary: $"
+ salary);
    }

    public void displayDetails() {
        System.out.println("Employee ID: " + id);
        System.out.println("Employee Name: " + name);
        System.out.println("Employee Salary: $" + salary);
    }
}

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

        System.out.print("Enter Employee ID: ");
        int id = scanner.nextInt();
        scanner.nextLine(); // consume the newline character

        System.out.print("Enter Employee Name: ");
        String name = scanner.nextLine();

        System.out.print("Enter Employee Salary: $");
        double salary = scanner.nextDouble();

        System.out.print("Enter Employee Grade (A/B/C/D): ");
        char grade = scanner.next().charAt(0);

        EMPLOYEE employee = new EMPLOYEE(id, name, salary);
        employee.displayDetails();
        employee.incrementSalary(grade);

        scanner.close();
    }}
```

Output:

```
Enter Employee ID: 1234
Enter Employee Name: CMRIT
Enter Employee Salary: $70000
Enter Employee Grade (A/B/C/D): B
Employee ID: 1234
Employee Name: CMRIT
Employee Salary: $70000.0
Salary incremented by 5.0%. New salary: $73500.0
```