Internal Assessment Test 2 – August 2022

Scheme & Model Solution

Sub: Object Oriented Concepts		Sub Code: 18CS45	Semester: IV		Sections: A,B,C
			MARKS	CO	RBT
Question	1a	With the example write a short note on i) this ii) final	5	CO1	L2
Scheme		this – points (2.5M) final (2.5M)	2.5 + 2.5		

Solution		**this keyword**
		• this keyword can be used to refer current class instance variable.
		• If there is ambiguity between the instance variable and parameter, this keyword resolves the problem of ambiguity
		Understanding the problem without this keyword
		Let's understand the problem if we ;;don't use this keyword by the example given below:
		`class student {`
		`int id; String name;`
		`student(int id,String name) {`
		`id = id; name = name;`
		`}`
		`void display() {`
		`System.out.println(id+" "+name);`
		`} }`
		`Class MyPgm {`
		`public static void main(String args[]) {`
		`student s1 = new student(111,"Anoop");`
		Output 0 null
		0 null
		In the above example, parameter (formal arguments) and instance variables are same that is why we are using this keyword to distinguish between local variable and instance variable
		Solution of the above problem by this keyword			

```
class Student {
        int id;  String name;
        student(int id,String name)  {
                this.id = id;
                this.name = name;
                }
         void display()
        {
                System.out.println(id+" "+name);
        }}

   Class MyPgm {
        public static void main(String args[]) {
                Student s1 = new Student(111,"Anoop");
                Student s2 = new Student(222,"Aryan");
                s1.display();
                s2.display();
        } }

Output111 Anoop
        222 Aryan
```

ii) The final keyword in java is used to restrict the user. The final keyword can be used in many context. Final can be:

1. variable
2. method
3. class

1) final variable: If you make any variable as final, you cannot change the value of final variable(It will be constant).

2) final method: If you make any method as final, you cannot override it.

3) final class:If you make any class as final, you cannot extend it.

Question	1b	Write a java program to print the sum, difference and product of two complex numbers by creating a class named 'Complex' with separate methods for each operation whose real and imaginary parts are entered by user.	5	CO2	L3
Scheme		Program (5M) Addition -1M Substruction – 1M Multiplication -2M Main function – 1M	5		

Solution

```
import java.util.Scanner;
public class Complex {
	double real, img;
	Complex(double r, double i){
		this.real = r;
		this.img = i;
  }
	public static Complex sum(Complex c1,
Complex c2)  {
		Complex temp = new Complex(0, 0);
		temp.real = c1.real + c2.real;
		temp.img = c1.img + c2.img;
		return temp;
	}
	public static Complex diff(Complex c1,
Complex c2)  {
		Complex temp = new Complex(0, 0);
		temp.real = c1.real - c2.real;
		temp.img = c1.img - c2.img;
		return temp;
	}
	public static Complex
Multiplication(Complex c1, Complex c2) {
		Complex temp = new Complex(0, 0);
		temp.real = (c1.real * c2.real) -
(c1.img * c2.img);
		temp.img = (c1.img * c2.real) +
(c1.real * c2.img);
		return temp;
	}
	public static void main(String[] args) {
		Scanner sc = new Scanner(System.in);
		System.out.println("Enter two
complex number: ");
		double a = sc.nextDouble(); double
b= sc.nextDouble();
		double c= sc.nextDouble(); double d=
sc.nextDouble();
		Complex c1 = new Complex(a,b);
		Complex c2 = new Complex(c,d);
		Complex temp = sum(c1, c2);
		Complex temp_diff = diff(c1, c2);
		Complex temp_mul =
Multiplication(c1, c2);
		System.out.printf("Sum is: "+
temp.real+" + "+ temp.img +"i");
		System.out.printf("Difference is: "+
temp_diff.real+" + "+ temp_diff.img +"i");
		System.out.printf("Difference is: "+
temp_mul.real+" + "+ temp_mul.img +"i");
	}

}
```

Question	2a	Demonstrate any two utilities of "super" with programming example	5	CO1	L2
Scheme		To call superclass constructors – 2.5 Use -1M + Example – 1.5M To access a member of the superclass: – 2.5 Use -1M + Example – 1.5M	2+3		

Solution		**Use -1**
		class Box {
		double width; double height; double depth;
		}
		Box (double w, double h, double d) {
		width = w; height = h; depth = d;
		}
		double volume() {
		return height * depth * width; } }
		
		class BoxWeight extends Box {
		double weight;
		BoxWeight (double w, double h, double d, double m){
		super(w,h,d);
		weight = m;
		}
		
		Use -2
		class B extends A {
		int i;
		B(int a, int b) {
		super.i = a;
		i = b;
		}
		void show() {
		System.out.println("i in superclass: " + super.i);
		System.out.println("i in subclass:" + i);
		} }
		class UseSuper {
		public static void main(String args[]) {
		B subOb = new B(1,2);
		subOb.show();
		}
		}			

Question	2b	Create a class named 'Member' having the following members: Data members: Name, Age, Phone number, Address, Basic Salary (By default to 1000) It also has a method named 'printSalary' which prints the salary of the members. Two classes 'Employee' and 'Manager' inherits the 'Member' class. Input all the data members to an employee and a manager. Print the net-salary for Employee and Manager where there is 10% and 15% increment of basic salary for Employee and Manager respectively	5	CO2	L3
Scheme		Program (5M) Member class – 1M Employee class – 1.5 M Manager class – 1.5 M Main class – 1M	5		

Solution		**Member.java**
		public class Member {
		String name,phn, address; int age;double sal;
		public Member(String n,String p,String a, int ag) {
		name =n; phn =p; address = a; age = ag; sal = 1000;
		}
		void printSalaty() {
		System.out.println("The salary is : "+sal);
		} }
		
		Employee.java
		public class Employee extends Member {
		public Employee(String n,String p,String a, int ag) {
		super(n,p,a,ag);
		}
		void printSalaty() {
		double netsal = sal + (sal*10)/100;
		System.out.println("The salary is : "+netsal);
		} }
		Manager.java
		public class Manager extends Member {
		public Manager(String n,String p,String a, int ag) {
		super(n,p,a,ag);
		}
		void printSalaty() {
		double netsal = sal + (sal*15)/100;
		System.out.println("The salary is : "+netsal);
		} }
		
		Main class
		public class Demo {
		public static void main(String[] args) {
		Employee emp = new Employee("Shyamasree", "9999999", "BAngalore", 35);
		Manager mng = new Manager("Sanchari", "8888888", "Mumbai", 38);
		emp.printSalaty();
		mng.printSalaty();
		} }			

Question	3a	Justify the statement : " **Method overriding form a basis for Dynamic Method Dispatch** " with suitable programming example.	5	CO1	L3
Scheme		Justification / Explanation – 2M Program - 3M	5		

Solution

Dynamic method dispatch is the mechanism in which a call to an overridden method is resolved at run time, rather than compile time.

Java uses the fact "a superclass reference variable can refer to a subclass object, to resolve calls to overridden methods at run time.

When an overridden method is called by a reference, java determines which version of that method to execute based on the type of object it refers to.

It is the type of the object being referred to not the type of reference variable that determines which version of an overridden method will be executed.

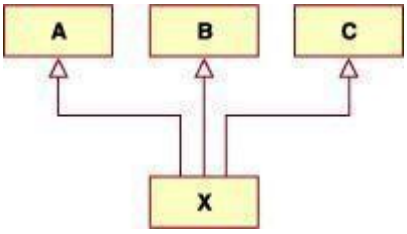
Therefore, if a superclass contains a method that is overridden by a subclass, then when different types of objects are referred to through a superclass reference variable, different versions of the method are executed.

Pictorial representation of dynamic method dispatch

Example program:

```
class A  {
   void callme ()  {
      System.out.println("Inside A's callme method");
   } }
class B extends A   {
   void callme  ()  {
      System.out.println("Inside B's callme method");
   } }
class C extends A {
   void callme()  {
      System.out.println("Inside C's callme method");
   }}
class Dispatch  {
   public static void main(String args[])  {
      A a = new A();B b = new B();C c = new C();
      A r;        // obtain a reference of type A
      r = a;     // r refers to an A object
      r.callme();       // Calls A's version of callme()
       r = b;     // r refers to a B object
      r.callme();     // Calls B's version of callme()
       r = c;   // r refers to a C object
      r.callme();     // Calls C's version of callme()
      }  }
```

Question	3b	Can Java provides multiple inheritance? Justify your answer with suitable example	5	CO1	L2
Scheme		Explanation / Justification- 4 points (2M) Program (3M)	2+3		

Solution		The mechanism of inheriting the features of more than one base class into a single class is known as multiple inheritance. Java does not support multiple inheritance but the multiple inheritance can be achieved by using the interface.
		 
		Here you can derive a class from any number of base classes. Deriving a class from more than one direct base class is called multiple inheritance.
		Java does not support multiple Inheritance
		In Java Multiple Inheritance can be achieved through use of Interfaces by implementing more than one interfaces in a class.
		Multiple Inheritance is a feature of object oriented concept, where a class can inherit properties of more than one parent class.
		The problem occurs when there exist methods with the same signature in both the super classes and subclass. On calling the method, the compiler cannot determine which class method to be called and even on calling which class method gets the priority
		Therefore, in order to avoid such complications **Java does not support multiple inheritance of classes.** But, a class can implement two or more interfaces
		A class can implement more than one interface, which can contain default methods that have the same name. The Java compiler provides some rules to determine which default method a particular class uses.
		Example program to demonstrate multiple inheritance using interface
		// Program to demonstrate multiple inheritance using interface
		// Define the interface I1
		interface I1 {
		void showI1() ;
		}
		// Define the interface I2
		interface I2 {
		void showI2();
		}
		// Define MInheritance that implements both I1 and I2
		class MInheritance implements I1, I2 {
		// Implement I1's interface
		public void showI1() {
		System.out.println("Inside showI1");
		}
		// Implement I2's interface
		public void showI2() {
		System.out.println("Inside showI2");
		}
		}
		class TestMI {
		public static void main(String args[]) {			

```
    MInheritance MI = new MInheritance();
    MI.showI1();
    MI.showI2();
  }
}
```

Output:

```
$ javac TestMI.java
$ java TestMI
Inside showI1
Inside showI2
```

Question	4a	Define chained Exception. Illustrate how chained exception can provide a root cause for the generated exception, with an suitable example	5	CO1	L2
Scheme		Definition – 1M Example Program 4M	1+4		

Solution		The chained exception feature allows you to associate another exception with an exception.
		This second exception describes the cause of the first exception.
		Exception chaining (also known as "nesting exception") is a technique for handling the exception, which occur one after another i.e. most of the time
		class ChainExcDemo {
		static void demoproc () {
		NullPointerException e = new NullPointerException ("top layer");
		e.initCause(new ArithmeticException("cause"));
		throw e;
		}
		public static void main(String args[]) {
		try {
		demoproc();
		}
		catch (NullPointerException e) {
		System.out.println("Caught: " + e);
		System.out.println("Original cause : " + e.getCause());
		} } }			

Question	4b	Consider the following two Exceptions : "/ by 0" and "Array out of bound " Exceptions and demonstrate how LIFO approach is followed during the execution of nested try statements	5	CO2	L3
Scheme		Program -4M Demonstration / Explanation – 1M	5		

Solution		class NestTry {
		public static void main(String args[]) {
		try {
		int a = args.length;
		/* If no command-line args are present,the following statement will generate a divide-by-zero exception. */
		int b = 42 / a;
		System.out.println("a = " + a);
		try { // nested try block
		/* If one command-line arg is used, then a divide-by-zero exception will be generated by the following code. */
		if(a==1) a = a/(a-a); // division by zero
		/* If two command-line args are used,then generate an out-of-bounds exception. */
		if(a==2) {
		int c[] = { 1 }; c[42] = 99; // generate an out-of-bounds exception }
		}
		catch(ArrayIndexOutOfBoundsException e) {
		System.out.println("Array index out-of-bounds: " + e);
		} }
		catch(ArithmeticException e) {
		System.out.println("Divide by 0: " + e);
		} } }
		
		The program works as follows. When you execute the program with no command-line arguments, a divide-by-zero exception is generated by the outer try block. Execution of the program with one command-line argument generates a divide-by-zero exception from within the nested try block. Since the inner block does not catch this exception, it is passed on to the outer try block, where it is handled. If you execute the program with two command-line arguments, an array boundary exception is generated from within the inner try block. Here are sample runs that illustrate each case:
		
		$ javac NestTry.java
		$ java NestTry
		Divide by 0: java.lang.ArithmeticException: / by zero
		
		$ java NestTry cmrit
		a = 1
		Divide by 0: java.lang.ArithmeticException: / by zero			

Question	5a	Formulate a table to explain how packages provides a fine-grained access control to the classes and sub classes.	5	CO1	L2
Scheme		Table formation – 2M Explanation – 3M	2 + 3		
Solution					

TABLE 9-1 Class Member Access

	Private	No Modifier	Protected	Public
Same class	Yes	Yes	Yes	Yes
Same package subclass	No	Yes	Yes	Yes
Same package non-subclass	No	Yes	Yes	Yes
Different package subclass	No	No	Yes	Yes
Different package non-subclass	No	No	No	Yes

Packages add another dimension to access control. Java provides many levels of protection to allow fine-grained control over the visibility of variables and methods within classes, subclasses, and packages.

Classes and packages are both means of encapsulating and containing the name space and scope of variables and methods. Packages act as containers for classes and other subordinate packages. Classes act as containers for data and code. The class is Java's smallest unit of abstraction. Because of the interplay between classes and packages, Java addresses four categories of visibility for class members:

• Subclasses in the same package
• Non-subclasses in the same package
• Subclasses in different packages
• Classes that are neither in the same package nor subclasses

The three access specifiers, private, public, and protected, provide a variety of ways to produce the many levels of access required by these categories. The following table sums up the interactions.

1) **private access modifier**

The private access modifier is accessible only within class.

2)**default access modifier**

If you don't use any modifier, it is treated as default by default. The default modifier is accessible only within package.

3)**protected access modifier**

The protected access modifier is accessible within package and outside the package but through inheritance only. The protected access modifier can be applied on the data member, method and constructor. It can't be applied on the class.

4) **public access modifier**

The public access modifier is accessible everywhere. It has the widest scope among all other modifiers.

Question	5b	Write a Package MCA which has one class Student. Accept student detail through parameterized constructor. Write display () method to display details. Create another class (main class) which will use package and calculate total marks and percentage.	5	CO2	L3
Scheme		Program (5M)	5		

Solution		**Student.java**
		package mca;
		public class Student
		{
		public int r_no;
		public String name;
		public int a,b,c;
		int sum=0;
		public Student(int roll, String nm, int m1,int m2,int m3) {
		r_no = roll;
		name = nm;
		a = m1;
		b = m2;
		c = m3;
		sum = a+b+c;
		}
		public void display() {
		System.out.println("Roll_no : "+r_no);
		System.out.println("Name : "+name);
		System.out.println("-----MARKS-------");
		System.out.println("Sub 1 : "+a);
		System.out.println("Sub 2 : "+b);
		System.out.println("Sub 3 : "+c);
		System.out.println("Total : "+sum);
		System.out.println("percentage: "+sum/3);
		System.out.println("------------------");
		}
		}
		
		Studentmain.java
		import mca.Student;
		import java.util.*;
		import java.lang.*;
		import java.io.*;
		class StudentMain {
		public static void main(String[] args) {
		String nm; int roll;
		Scanner sc = new Scanner(System.in);
		System.out.print("Enter Roll no:= ");
		roll = sc.nextInt();
		System.out.print("Enter Name:= ");
		nm = sc.next();
		int m1,m2,m3;
		System.out.print("Enter 3 sub mark:= ");
		m1 = sc.nextInt();
		m2 = sc.nextInt();
		m3 = sc.nextInt();
		Student s = new Student(roll,nm,m1,m2,m3);
		s.display();
		}
		}			

Question	6a	Define the syntax for interface definition. Explain how interface can be inherited with suitable example.	5	CO1	L2
Scheme		syntax – 1M Example Program (4M)	1 + 4		

Solution		access_specifier interface name {
		return-type method-name1 (parameter-list);
		return-type method-name2 (parameter-list);
		type final-varname1 = value;
		type final-varname2 = value;
		//...
		return-type method-nameN(parameter-list);
		type final-varnameN = value;
		}
		Interface can be Extended
		• one interface can inherit another by use of the keyword extends.
		• When a class implements an interface that inherits another interface, it must provide implementations for all methods defined within the interface inheritance chain.
		Eg:
		// One interface can extend another.
		
		Interface A {
		void meth1();
		void meth2();
		}
		// B now includes meth1() and meth2() --- it adds meth3()
		interface B extends A {
		void meth3();
		}
		// This class must implement all of A and B
		class MyClass implements B {
		public void meth1() {
		System.out.println(“Method1”);
		}
		public void meth2() {
		System.out.println(“Method2”);
		}
		public void meth3 () {
		System.out.println(“Method3”);
		}
		}
		
		Class IFExtend {
		public static void main(String args[]) {
		MyClass ob = new MyClass();
		
		ob.meth1();
		ob.meth2();
		ob.meth3();
		}			

Question	6b	State the conceptual and structural differences of Abstract class and interface	5	CO1	L2
Scheme		Minimum 8 Differences – 5M	5		

Solution

oops interface vs abstract class

Interface	Abstract class
Interface support multiple inheritance	Abstract class does not support multiple inheritance
Interface does'n Contains Data Member	Abstract class contains Data Member
Interface does'n contains Cunstructors	Abstract class contains Cunstructors
An interface Contains only incomplete member (signature of member)	An abstract class Contains both incomplete (abstract) and complete member
An interface cannot have access modifiers by default everything is assumed as public	An abstract class can contain access modifiers for the subs, functions, properties
Member of interface can not be Static	Only Complete Member of abstract class can be Static

Abstract class	Interface
1) Abstract class can **have abstract and non-abstract** methods.	Interface can have **only abstract** methods. Since Java 8, it can have **default and static methods** also.
2) Abstract class **doesn't support multiple inheritance**.	Interface **supports multiple inheritance**.
3) Abstract class **can have final, non-final, static and non-static variables**.	Interface has **only static and final variables**.
4) Abstract class **can provide the implementation of interface**.	Interface **can't provide the implementation of abstract class**.
5) The **abstract keyword** is used to declare abstract class.	The **interface keyword** is used to declare interface.
6) An **abstract class** can extend another Java class and implement multiple Java interfaces.	An **interface** can extend another Java interface only.
7) An **abstract class** can be extended using keyword "extends".	An **interface** can be implemented using keyword "implements".
8) A Java **abstract class** can have class members like private, protected, etc.	Members of a Java interface are public by default.

9)**Example:**
public abstract class Shape{
public abstract void draw();
}
	Example:
	public interface Drawable{
	void draw();
	}
