SPRING MID SEMESTER EXAMINATION-2024

School of Computer Engineering
Kalinga Institute of Industrial Technology, Deemed to be University
Object Oriented Programming using Java
[CS20004/CC20004]

Solution Scheme

Time: 1 1/2 Hours **Full Mark: 20**

Answer Any four questions including question No.1 which is compulsory.
The figures in the margin indicate full marks. Candidates are required to give theiranswers in their own words as far as practicable and all parts of a question should beanswered at one place only.

1. Answer all the questions. [1 Mark X 5]

a) List out the object oriented concepts in java. Define class and object with suitable example.

Ans:

Object oriented concepts in java

- Class and Object
- Message Passing
- Abstraction
- Encapsulation
- Inheritance
- Polymorphism

A **class** in Java is a set of objects which shares common characteristics/ behavior and common properties/ attributes. It is a user-defined blueprint or prototype from which objects are created. An object in Java is a basic unit of Object-Oriented Programming and represents real-life entities. Objects are the instances of a class that are created to use the attributes and methods of a class. For example, **Student is a class while a particular student named Ravi is an object of Student class.**

[0.5 mark for concepts + 0.5 mark for class and object]

b) Find the output.

```
class Leftshift_operator
    {
        public static void main(String args[])
            {
                byte x = 64;
                int i;
                byte y;
                i = x << 2;
                y = (byte) (x << 2);
                System.out.print(i + " " + y);
            }
```

}

Output: 256 0 [1 mark for correct output]

c) Write the three uses of 'final' keyword in java.

Ans: Create Constants, prevent inheritance, and prevent methods from being inheritance are the three main uses of final keyword in java. **[1 mark for three uses]**

- The final variable cannot be reinitialized with another value. (eg. final int i=5;)
- A final method cannot be overridden by another method. (eg. final void display(){})
- A final class cannot be extended or inherited by another child class. (eg. final class A{})

d) Can we use this() and super() both together in a constructor ? Justify your answer.

Ans: NO. Constructor must always be the first statement. super() calls the base class constructor & this() calls the current class constructor. we can not have two statements as first statement, so we either need to call super() or this() but not the both. **[correct answer(0.5 mark) + explaination (0.5 mark)]**

e) Find the output with reason.

```
class A{
 int a=90;
}
class B extends A{
 int a=150;

 public static void main(String args[]){

A obj=new B();

  System.out.println(obj.a);
}
}
```

Output: 90

In the above code, both the classes have a data member i. We are accessing the data member by the reference variable of Parent class(A) which refers to the subclass(B) object. Since we are accessing the data member(i) which is not overridden, hence it will access the data member of the Parent class always.

Rule: Runtime polymorphism can't be achieved by data members. [0.5 mark for output + 0.5 mark for reason]

2. a) Explain the features of Object Oriented Programming with suitable example in java i.e. mentioned below?

i. Robust Language [2.5 Marks]

ii. Secured

iii. Compiled and interpreted language
iv. Portable language
v. Architecture Neutral

Ans: **[0.5 mark for each feature]**

Robust: Java makes an effort to eliminate error prone codes by emphasizing mainly on compile time error checking and runtime checking. But the main areas which Java improved were Memory Management and mishandled Exceptions by introducing automatic Garbage Collector and Exception Handling.

Secured: When it comes to security, Java is always the first choice. With java secure features it enable us to develop virus free, temper free system. Java program always runs in Java runtime environment with almost null interaction with system OS, hence it is more secure.

Compiled and interpreted language: Java can be considered both a compiled and an interpreted language because its source code is first compiled into a binary byte-code. This byte-code runs on the Java Virtual Machine (JVM), which is usually a software-based interpreter.

Portable: Java Byte code can be carried to any platform. No implementation dependent features. Everything related to storage is predefined, example: size of primitive data types

Architectural Neutral: Compiler generates bytecodes, which have nothing to do with a particular computer architecture, hence a Java program is easy to intrepret on any machine.

b) Write a program in java to receive five integers from user where three are from command line arguments and two are using scanner class and count the number of prime numbers in them. [2.5Marks]

Ans: **[1 mark for inputs + 1 mark for prime number logic + 0.5 mark for other]**

```
import java.util.*;
class PrimeCLA
{
public static void main(String arg[])
{
Scanner sc=new Scanner( System.in);
int a[]=new int[5];
a[0]=Integer.parseInt(arg[0]);
a[1]=Integer.parseInt(arg[1]);
a[2]=Integer.parseInt(arg[2]);
a[3]=sc.nextInt();
a[4]=sc.nextInt();
int f=0;
int count=0;
```

```
for(int j=0;j<5;j++)
{
for(int i=2;i<=a[j]/2; i++)
{
if(a[j]%i==0)
{

f=1;
break;
}
}
if(f==0)
{

count++;
}
}
System.out.println("Total no of Prime numbers:"+count);
}
}
```

3. a) Briefly explain the different types of inheritance in java with proper syntax. [2.5 Marks]

Ans: **[0.5 mark for each inheritance with syntax]**

In Java, inheritance is a mechanism that allows one class to acquire the properties and behaviors of another class. There are several types of inheritance in Java:

1. **Single Inheritance:** A class can extend only one class.
   Example: class A { /* ... */ }
   class B extends A { /* ... */ }
2. **Multiple Inheritance (through Interfaces):** Java supports multiple inheritance through interfaces, where a class can implement multiple interfaces.
   Example: interface A { /* ... */ }
   interface B { /* ... */ }
   class C implements A, B { /* ... */ }
3. **Multilevel Inheritance:** A class can be derived from another class, and then another class can be derived from it.
   Example: class A { /* ... */ }

class B extends A { /* ... */ }
class C extends B { /* ... */ }

4. **Hierarchical Inheritance:** Multiple classes can be derived from a single base class.
   Example: class A { /* ... */ }
   class B extends A { /* ... */ }
   class C extends A { /* ... */ }
5. **Hybrid Inheritance:** Hybrid inheritance is a combination of multiple types of inheritance within a single program.
   Example: class A { /* ... */ }
   interface B { /* ... */ }
   class C extends A implements B { /* ... */ }

b) Write a program in java to create an abstract class Shape with two abstract methods getArea() , getPerimeter() and a non-abstract printInfo() method. Create two subclasses Circle and Rectangle, each implementing these abstract methods. The application Demo class demonstrates the usage of these classes by creating objects of Circle and Rectangle and invoking their methods. [2.5 Marks]
Ans:

/ Abstract class **[1mark for abstract class + 1.5 mark for other subclasses]**

```
abstract class Shape {
   // Abstract method to get area
   public abstract double getArea();

   // Abstract method to get perimeter
   public abstract double getPerimeter();

   // Concrete method
   public void printInfo() {
      System.out.println("This is a shape.");
   }
}

// Subclass Circle
class Circle extends Shape {
   private double radius;

   // Constructor
   public Circle(double radius) {
      this.radius = radius;
```

```
    }

    // Implementing abstract method to get area
    @Override
    public double getArea() {
        return Math.PI * radius * radius;
    }

    // Implementing abstract method to get perimeter
    @Override
    public double getPerimeter() {
        return 2 * Math.PI * radius;
    }
}

// Subclass Rectangle
class Rectangle extends Shape {
    private double width;
    private double height;

    // Constructor
    public Rectangle(double width, double height) {
        this.width = width;
        this.height = height;
    }

    // Implementing abstract method to get area
    @Override
    public double getArea() {
        return width * height;
    }

    // Implementing abstract method to get perimeter
```

```
    @Override
    public double getPerimeter() {
        return 2 * (width + height);
    }
}


public class Main {
    public static void main(String[] args) {
        Circle circle = new Circle(5);
        System.out.println("Circle Area: " + circle.getArea());
        System.out.println("Circle Perimeter: " + circle.getPerimeter());
        circle.printInfo();


        Rectangle rectangle = new Rectangle(4, 6);
        System.out.println("\nRectangle Area: " + rectangle.getArea());
        System.out.println("Rectangle Perimeter: " + rectangle.getPerimeter());
        rectangle.printInfo();
    }
```

4. a) Write a program in java having three classes Apple, Banana and Cherry. Class Banana and Cherry are inherited from class Apple and each class has its own member function show() . Using Dynamic Method Dispatch concept, display all the show() method of each class. [2 Marks]

[1 mark for DMD concept + 1 mark for other]

Ans:

```
class Apple
{
 void show()
{
 System.out.println("Apple show method");
}

}
class Banana extends Apple
{
void show()
{
 System.out.println("Banana show method");
}
}
class Cherry extends Apple
{
void show()
{
 System.out.println("Cherry show method");
}
}
```

```
class Demo
{
public static void main(String args[])
{
 Apple ob1=new Apple();
Apple ob2=new Banana();
Apple ob3=new Cherry();
ob1.show();
ob2.show();
ob3.show();
}
}
```

b) Write a class Complex in Java with two data members -real, img and overloaded constructors. It contains two methods - Swap() and Sum(). Swap method interchanges the values of real and img of an object and Sum method adds two complx numbers and returns a new complex object. Write the complete program to check the functionality of both the methods. [3 Marks]

[1 mark for Swap() + 1 mark for Sum() + 1 mark for other]

Ans:

```
import java.util.*;
public class Complex
{
    float real,imag;
    Complex()
    {
        this.real=10.0f;
        this.imag=10.0f;
    }
    Complex(float a, float b)
    {
        this.real=a;
        this.imag=b;
    }
```

```
public static Complex Swap(Complex C1)
{
    Complex temp=new Complex(0.0f,0.0f);
    temp.real=C1.real;
    C1.real=C1.imag;
    C1.imag=temp.real;
    return temp;
}
public static void main(String args[])
{
    Complex C1=new Complex();
    Complex C2=new Complex(1.5f,2.5f);
    Complex add=Sum(C1,C2);
    System.out.println("Sum="+add.real+" "+a
    Complex D1=Swap(C1);
```

5. a) Explain the different types of inner class with example in java. [2.5 Marks]

Ans: **[0.5 mark for types of inner class + 0.5 mark for each innerclass explanation]**

Java inner class or nested class is a class that is declared inside the class or interface. We use inner classes to logically group classes and interfaces in one place to be more readable and maintainable. Additionally, it can access all the members of the outer class, including private data members and methods.

Syntax of Inner class

```
class Java_Outer_class{
 //code
 class Java_Inner_class{
  //code
 }
}
```

There are 4 types of inner class.

1. Static inner class
2. Non static inner class
3. Local inner class
4. Anonymous inner class

Static inner class

Nested classes that are declared static are called static nested classes.A static class is a class that is created inside a class, is called a static nested class in Java. It cannot access non-static data members and methods. It can be accessed by outer class name. It can access static data members of the outer class, including private.

Example:

```
class TestOuter1{
  static int data=30;
  static class Inner{
   void msg(){System.out.println("data is "+data);}
  }
  public static void main(String args[]){
  TestOuter1.Inner obj=new TestOuter1.Inner();
```

```
        obj.msg();
        }
      }
```

Non static inner class

Non-static nested classes are called inner classes. A nested class is a member of its enclosing class. Non-static nested classes (inner classes) have access to other members of the enclosing class, even if they are declared private.

Example:

```
public class OuterClass {
   private int outerInstanceField;
   public OuterClass(int outerInstanceField) {
      this.outerInstanceField = outerInstanceField;
   }
   // Non-static (inner) nested class
   public class InnerClass {
      private int innerInstanceField;
      public InnerClass(int innerInstanceField) {
         this.innerInstanceField = innerInstanceField;
      }
      public void display() {
         // Accessing the instance field from the outer class
         System.out.println("Outer instance field: " + outerInstanceField);
         // Accessing the instance field from the inner class
         System.out.println("Inner instance field: " + innerInstanceField);
      }
   }
   public static void main(String[] args) {
      // Create an instance of the outer class
      OuterClass outerObject = new OuterClass(10);
      // Create an instance of the inner class using the outer class instance
      InnerClass innerObject = outerObject.new InnerClass(20);
      // The display() method of the inner class is being called
      innerObject.display();
   }
}
```

Local inner class

Local Inner Classes are the inner classes that are defined inside a block. Generally, this block is a method body. Sometimes this block can be a for loop or an if clause. Local Inner classes are not a member of any enclosing classes. They belong to the block they are defined within, due to which local inner classes cannot have any access modifiers associated with them. However, they can be marked as final or abstract. This class has access to the fields of the class enclosing it. Local inner class must be instantiated in the block they are defined in.

Example:

```
public class Outer
{
        private void getValue()
        {

                int sum = 20;

                // Local inner Class inside method
                class Inner
                {
```

```
                public int divisor;
                public int remainder;

                public Inner()
                {
                        divisor = 4;
                        remainder = sum%divisor;
                }
                private int getDivisor()
                {
                        return divisor;
                }
                private int getRemainder()
                {
                        return sum%divisor;
                }
                private int getQuotient()
                {
                        System.out.println("Inside inner class");
                        return sum / divisor;
                }
        }

        Inner inner = new Inner();
        System.out.println("Divisor = "+ inner.getDivisor());
        System.out.println("Remainder = " + inner.getRemainder());
        System.out.println("Quotient = " + inner.getQuotient());
    }

    public static void main(String[] args)
    {
        Outer outer = new Outer();
        outer.getValue();
    }
}
```

Anonymous inner class

Java anonymous inner class is an inner class without a name and for which only a single object is created. An anonymous inner class can be useful when making an instance of an object with certain "extras" such as overloading methods of a class or interface, without having to actually subclass a class. In simple words, a class that has no name is known as an anonymous inner class in Java. It should be used if you have to override a method of class or interface. Java Anonymous inner class can be created in two ways: Class (may be abstract or concrete) and Interface.

```
abstract class Person{
  abstract void eat();
}
class TestAnonymousInner{
public static void main(String args[]){
 Person p=new Person(){
 void eat(){System.out.println("nice fruits");}
 };
 p.eat();
 }
}
```

b) Create a package biodata which consists of two interfaces namely schooling, college and a class Student that implements these interfaces. Access the Student class in another package pack1 in a class Department and test it.

[2.5 Marks]

[1 mark for creation of package and 1.5 marks for accessing class inside package]

```
package Biodata;
interface Schooling
{
  public void getdata();
}
interface College
{
 public void display();
}
class Student implements Schooling,College
{
   public void getdata()
{
   System.out.println("Schooling data");
}

 public void display()
{
   System.out.println("College data");
}
}
```

```
package pack1;
import Biodata.*;
class Department
{
 public static void main(String args[])
{
  Student s=new Student();
  s.getdata();
  s.display();
}
}
```

*** Best of Luck ***