**DIT: OOP Assignment**

**Part A:**

**Instructions for part A: answer all the Questions in this section.**

1. **Using a well labeled diagram, explain the steps of creating a system using OOP principles. [4 Marks]**  Identify Objects:

Identify the entities or objects in the problem domain that need to be represented in the system. These could be real-world entities or abstract concepts.

Define Classes:

For each identified object, create a corresponding class. A class is a blueprint or template for creating objects. It defines the properties (attributes) and behaviors (methods) of the objects.

Identify Relationships:

Determine how objects are related to each other. Establish associations, aggregations, or compositions between classes to represent the connections and interactions between objects.

Encapsulation:

Encapsulate the properties and methods within each class. Hide the internal details of the class and provide a well-defined interface for interacting with the object. Use access modifiers to control the visibility of class members.

Inheritance:

Identify commonalities among classes and use inheritance to create a hierarchy of classes. Inheritance allows a class to inherit properties and behaviors from another class, promoting code reuse and establishing an "is-a" relationship.

Polymorphism:

Use polymorphism to allow objects of different classes to be treated as objects of a common base class. This enables flexibility and extensibility in the system. Polymorphism includes method overloading and method overriding.

Abstraction:

Abstract complex systems by creating abstract classes or interfaces that define common characteristics without specifying the implementation details. Abstraction simplifies the understanding of the system.

Modularity:

Break down the system into modular components or modules, where each module corresponds to a class or a group of related classes. This promotes maintainability and ease of understanding.

Test and Iterate:

Test each module and the system as a whole. Identify and fix any issues. Iterate through the development process to refine the design and implementation based on feedback and changing requirements.

Documentation:

Document the design, structure, and functionality of the system. Provide clear and comprehensive documentation for developers and future maintainers.

Implementation:

Implement the system by writing the actual code based on the OOP design. Follow best practices, coding standards, and design patterns to ensure code quality and maintainability.

Maintenance:

After deployment, maintain the system by addressing bugs, adding new features, and adapting to changing requirements. Follow a systematic approach to ensure the long-term sustainability of the system.

1. **What is the Object Modeling Techniques (OMT). [1 Marks]** -Object Modeling Techniques (OMT) is a method for visualizing and documenting the analysis and design of a software system.
2. **Compare object-oriented analysis and design (OOAD) and object analysis and design (OOP). [2 Marks] -**-OOAD is a broader term that encompasses both the analysis and design phases of the software development life cycle. OOP specifically refers to the phase where the software is implemented using object-oriented programming languages.
3. **Discuss Main goals of UML. [2 Marks] -**Standardization; Provide a standardized notation for visualizing, specifying, constructing, and documenting the artifacts of software systems. -Visualization; Enable stakeholders, including analysts, designers, developers, and users, to visualize and understand the structure and behavior of a system.
4. **DESCRIBE three advantages of using object oriented to develop an information system. [3Marks] -**Modularity and reusability; System is modularized into classes and objects. Each class encapsulates data and methods related to a specific entity or concept. This modularity facilitates code organization and maintenance. -Encapsulation; This is when the data (attributes) and the methods that operate on the data are encapsulated within a class. Access to the data is controlled through the class's methods. -Inheritance; Inheritance allows a class (subclass or derived class) to inherit properties and behaviors from another class (superclass or base class). This promotes a hierarchical structure and code reuse.
5. **Briefly explain the following terms as used in object-oriented programming. Write a sample java code to illustrate the implementation of each concept. [12 Marks]**
   1. **Constructor;** Is a special method in a class that is automatically invoked when an object of that class is created.
   2. **Object;** Is an instance of a class. It is a runtime entity that represents a real-world entity and encapsulates data (attributes) and behavior (methods).
   3. **Destructor;** Is a special member function of a class that is called when an object of that class goes out of scope or is explicitly deleted.
   4. **polymorphism;** Allows objects of different types to be treated as objects of a common type. This can be achieved through method overloading or method overriding.
   5. **class;** A class is a blueprint or a template for creating objects. It defines the structure (attributes and methods) that the objects will have.
   6. **Inheritance;** Inheritance is a mechanism where a new class (subclass or derived class) inherits properties and behaviors from an existing class (superclass or base class).

**vi. *EXPLAIN* the three types of associations (relationships) between objects in object oriented. [6 Marks]**  Association: Association is a general relationship where objects are connected, and one object is aware of the other

Aggregation: Aggregation is a more specialized form of association where one object represents a "whole" and another object represents a "part." The part object can exist independently of the whole.

Composition: Composition is a stronger form of aggregation where the part objects are part of the whole, and their lifecycle is tightly bound. If the whole is destroyed, all its parts are destroyed as well.

**Vii. What do you mean by class diagram? Where it is used and also discuss the steps to draw the class diagram with any one example. [6 Marks] -**A class diagram is a type of diagram in the Unified Modeling Language (UML) that represents the structure and relationships of classes within a system. It provides a static view of the system, showcasing the classes, their attributes, methods, and the associations between them.Step1; Identify Classes

Identify the key classes in your system. These are the entities that have both attributes and behaviors.

Step 2: Identify Relationships

Determine the relationships between the classes. Common types include association, aggregation, and composition.

Step 3: Identify Attributes and Methods

For each class, identify its attributes (properties) and methods (functions or behaviors).

Step 4: Add Multiplicity

Specify the multiplicity of associations, indicating how many instances of one class are related to an instance of another class.

Step 5: Add Visibility

Indicate the visibility of attributes and methods (public, private, protected) using symbols like +, -, and #.

Step 6: Draw the Diagram

Use UML notation to draw the class diagram, representing classes as rectangles, attributes as labeled lines, and associations as lines connecting the classes.

1. **Given that you are creating area and perimeter calculator using C++, to computer area and perimeter of various shaped like Circles, Rectangle, Triangle and Square, use well written code to explain and implement the calculator using the following OOP concepts.**
   1. **Inheritance (Single inheritance, Multiple inheritance and Hierarchical inheritance) [10 Marks]** #include <iostream>

#include <cmath>

// Base class for shapes

class Shape {

public:

virtual double calculateArea() const = 0;

virtual double calculatePerimeter() const = 0;

};

// Derived class for Circle

class Circle : public Shape {

private:

double radius;

public:

Circle(double r) : radius(r) {}

double calculateArea() const override {

return 3.14 \* radius \* radius;

}

double calculatePerimeter() const override {

return 2 \* 3.14 \* radius;

}

};

// Derived class for Rectangle

class Rectangle : public Shape {

private:

double length;

double width;

public:

Rectangle(double l, double w) : length(l), width(w) {}

double calculateArea() const override {

return length \* width;

}

double calculatePerimeter() const override {

return 2 \* (length + width);

}

};

// Derived class for Triangle

class Triangle : public Shape {

private:

double side1;

double side2;

double side3;

public:

Triangle(double s1, double s2, double s3) : side1(s1), side2(s2), side3(s3) {}

double calculateArea() const override {

// Heron's formula for area of a triangle

double s = (side1 + side2 + side3) / 2;

return sqrt(s \* (s - side1) \* (s - side2) \* (s - side3));

}

double calculatePerimeter() const override {

return side1 + side2 + side3;

}

};

// Derived class for Square (Hierarchical Inheritance)

class Square : public Rectangle {

public:

Square(double side) : Rectangle(side, side) {}

};

int main() {

Circle circle(5.0);

Rectangle rectangle(4.0, 6.0);

Triangle triangle(3.0, 4.0, 5.0);

Square square(5.0);

std::cout << "Circle - Area: " << circle.calculateArea() << ", Perimeter: " << circle.calculatePerimeter() << std::endl;

std::cout << "Rectangle - Area: " << rectangle.calculateArea() << ", Perimeter: " << rectangle.calculatePerimeter() << std::endl;

std::cout << "Triangle - Area: " << triangle.calculateArea() << ", Perimeter: " << triangle.calculatePerimeter() << std::endl;

std::cout << "Square - Area: " << square.calculateArea() << ", Perimeter: " << square.calculatePerimeter() << std::endl;

return 0;

* 1. **Friend functions [5 Marks]**
  2. **Method overloading and method overriding [10 Marks] -**Method overloading is demonstrated in the classes(Refer to the code in **Vii(a)** above) where there are multiple versions of the same function (calculateArea and calculatePerimeter). Method overriding is achieved using the override keyword in the derived classes.
  3. **Late binding and early binding [8 Marks]** -Late binding is achieved through the use of virtual functions in the base class (Shape). Early binding (compile-time binding) occurs when the base class pointer refers to the derived class object, and the compiler determines the function at compile-time.(Refer to the code in **Vii(a)** above)
  4. **Abstract class and pure functions [6 Marks]** -The base class Shape is an abstract class with pure virtual functions (calculateArea and calculatePerimeter). Objects of the abstract class cannot be instantiated, and derived classes must implement these functions.(Refer to the code in **Vii(a)** above)

1. **Using a program written in C++, differentiate between the following. [6 Marks]**
   1. **Function overloading and operator overloading Function overloading**

#include <iostream>

class OverloadExample {

public:

// Function Overloading

int add(int a, int b) {

return a + b;

}

double add(double a, double b) {

return a + b;

}

};

int main() {

OverloadExample obj;

std::cout << "Sum (int): " << obj.add(3, 4) << std::endl;

std::cout << "Sum (double): " << obj.add(3.5, 4.5) << std::endl;

return 0;

}

**Operator overloading**

#include <iostream>

class Complex {

private:

double real;

double imag;

public:

// Operator Overloading

Complex operator+(const Complex& other) {

Complex result;

result.real = real + other.real;

result.imag = imag + other.imag;

return result;

}

void display() {

std::cout << "Real: " << real << ", Imaginary: " << imag << std::endl;

}

};

int main() {

Complex c1, c2, result;

// Operator Overloading

result = c1 + c2;

result.display();

return 0;

}

**b) Pass by Reference and Pass By Value;**

Pass By Value

#include <iostream>

void modifyValue(int x) {

x = 10;

}

int main() {

int num = 5;

modifyValue(num);

std::cout << "Value after modification: " << num << std::endl;

return 0;

}

Pass by Value: Involves passing a copy of the actual parameter to the function. Changes made inside the function do not affect the original value.

Pass By Reference;

#include <iostream>

void modifyReference(int &x) {

x = 10;

}

int main() {

int num = 5;

modifyReference(num);

std::cout << "Value after modification: " << num << std::endl;

return 0;

}

Pass by Reference: Involves passing the memory address (reference) of the actual parameter to the function. Changes made inside the function directly affect the original value.

**c) Parameters and Arguments:**

Parameters; These are variables declared in a function's signature. They act as placeholders for values that will be passed when the function is called.

#include <iostream>

void printSum(int a, int b) {

std::cout << "Sum: " << a + b << std::endl;

}

int main() {

int num1 = 3, num2 = 4;

printSum(num1, num2);

return 0;

}

Arguments; These are the actual values passed to a function when it is called. They correspond to the parameters in the function's definition.

#include <iostream>

int add(int a, int b) {

return a + b;

}

int main() {

int result = add(3, 4);

std::cout << "Sum: " << result << std::endl;

return 0;

}

***NOTE: To score high marks, you are required to explain each question in detail. Do good research and cite all the sources of your information. DO NOTE CITE WIKIPEDIA.***

**Create a new class called *CalculateG.*Copy and paste the following initial version of the code. Note variables declaration and the types.**

**class** *CalculateG* **{  
int** main**(){**

(*datatype*) gravity =-9.81; // Earth's gravity in m/s^2 (*datatype*) fallingTime = 30;

(*datatype*)initialVelocity = 0.0; (*datatype*) finalVelocity = ;

(*datatype*) initialPosition = 0.0; (*datatype*) finalPosition = ;

// Add the formulas for position and velocity

Cout<<"The object's position after " << fallingTime << " seconds is "

+ finalPosition + << m."<<endl;

// Add output line for velocity (similar to position)

} }

Modify the example program to compute the position and velocity of an object after falling for 30 seconds, outputting the position in meters. The formula in Math notation is:

𝑥(𝑡)=0.5∗𝑎𝑡2 +𝑣𝑖𝑡+𝑥𝑖 𝑣(𝑡)=𝑎𝑡+𝑣𝑖

Run the completed code in Eclipse (Run → Run As → Java Application). 5. Extend *datatype* class with the following code:

**public class** *CalculateG* {

**public double** multi(**......**){ // method for multiplication

}

// add 2 more methods for powering to square and summation (similar to multiplication)

**public void** outline(**......**){  
// method for printing out a result

}  
**int** main() {

// compute the position and velocity of an object with defined methods and print out the

result

} }

6. Create methods for multiplication, powering to square, summation and printing out a result in *CalculateG* class.

public class CalculateG {

public static void main(String[] args) {

double gravity = -9.81; // Earth's gravity in m/s^2

double fallingTime = 30.0;

double initialVelocity = 0.0;

double finalVelocity;

double initialPosition = 0.0;

double finalPosition;

// Add the formulas for position and velocity

finalPosition = 0.5 \* gravity \* Math.pow(fallingTime, 2) + initialVelocity \* fallingTime + initialPosition;

finalVelocity = gravity \* fallingTime + initialVelocity;

// Output the result

System.out.println("The object's position after " + fallingTime + " seconds is " + finalPosition + " m.");

System.out.println("The object's velocity after " + fallingTime + " seconds is " + finalVelocity + " m/s.");

}

// Extend the class with additional methods

public double multi(double a, double b) {

return a \* b;

}

public double power(double base, int exponent) {

return Math.pow(base, exponent);

}

public double summation(double a, double b) {

return a + b;

}

// Method for printing out a result

public void outline(double result) {

System.out.println("Result: " + result);

}

}

**Part B:**

**Instructions for part B: Do question 1 and any other one question from this section.**

1. Each new term in the Fibonacci sequence is generated by adding the previous two terms. By starting with 1 and 2, the first 10 terms will be:  
1, 2, 3, 5, 8, 13, 21, 34, 55, 89, ...

By considering the terms in the Fibonacci sequence whose values do not exceed four million, write a C++ method to find the sum of all the even- valued terms.

#include <iostream>

int main() {

const int limit = 4000000;

int term1 = 1, term2 = 2, nextTerm, sum = 0;

while (term1 <= limit) {

// Check if the current term is even

if (term1 % 2 == 0) {

sum += term1;

}

// Generate the next term in the Fibonacci sequence

nextTerm = term1 + term2;

term1 = term2;

term2 = nextTerm;

}

std::cout << "Sum of even-valued terms in the Fibonacci sequence (up to 4 million): " << sum << std::endl;

return 0;

}

**Question Two: [15 marks]**

2. A palindrome number is a number that remain the same when read from behind or front ( a number that is equal to reverse of number) for example, 353 is palindrome because reverse of 353 is 353 (you see the number remains the same). But a number like 591 is not palindrome because reverse of 591 is 195 which is not equal to 591. Write C++ program to check if a number entered by the user is palindrome or not. You should provide the user with a GUI interface to enter the number and display the results on the same interface.

The interface:

**Check if a number is palindrome**

345

Enter the number

Not palindrome

Output 🡪

**Question three: [15 marks]**

Write a C++ program that takes 15 values of type integer as inputs from user, store the values in an array.

1. Print the values stored in the array on screen.
2. Ask user to enter a number, check if that number (entered by user) is present in array or not. If it is present print, “the number found at index (index of the number) ” and the text “number not found in this array”
3. Create another array, copy all the elements from the existing array to the new array but in reverse order. Now print the elements of the new array on the screen
4. Get the sum and product of all elements of your array. Print product and the sum each on its own line.

#include <iostream>

int main() {

const int size = 15;

int inputArray[size];

int searchNumber, sum = 0, product = 1;

// a) Input values into the array

std::cout << "Enter " << size << " integer values:" << std::endl;

for (int i = 0; i < size; ++i) {

std::cout << "Enter value " << i + 1 << ": ";

std::cin >> inputArray[i];

sum += inputArray[i];

product \*= inputArray[i];

}

// b) Print values stored in the array

std::cout << "\nValues stored in the array:" << std::endl;

for (int i = 0; i < size; ++i) {

std::cout << inputArray[i] << " ";

}

// b) Ask user to enter a number and check if it's present in the array

std::cout << "\nEnter a number to search in the array: ";

std::cin >> searchNumber;

bool numberFound = false;

for (int i = 0; i < size; ++i) {

if (inputArray[i] == searchNumber) {

std::cout << "The number found at index " << i << std::endl;

numberFound = true;

break;

}

}

if (!numberFound) {

std::cout << "Number not found in this array" << std::endl;

}

// c) Create another array, copy elements in reverse order, and print the new array

int reversedArray[size];

for (int i = 0, j = size - 1; i < size; ++i, --j) {

reversedArray[j] = inputArray[i];

}

std::cout << "\nElements of the new array (in reverse order):" << std::endl;

for (int i = 0; i < size; ++i) {

std::cout << reversedArray[i] << " ";

}

// d) Print the sum and product of array elements

std::cout << "\n\nSum of array elements: " << sum << std::endl;

std::cout << "Product of array elements: " << product << std::endl;

return 0;

}