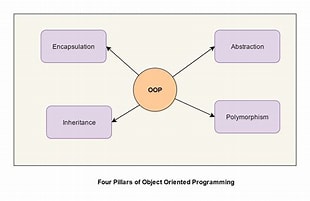
Brian Kiptoo Sumukwo  
SCT121-0607/2021

**Using a well labeled diagram, explain the steps of creating a system using OOP principles.**

i. 

1. Identifying the objects

The first step in OOP is to identify all the objects the system needs to handle. An object can be a data structure, a variable, a function or a method.

1. Organize the objects

Once the objects are identified, they need to be organized into classes. A class is a blueprint for creating objects. It defines what attributes and methods will be common to objects of that class.

1. Define the Attributed and Methods

Attributes are the properties that define the state of an object .Methods define the behavior of an object.

1. Establish Relationships between Objects

Objects can interact with each other through methods. The relationship between objects can be of different types like association, aggregation, inheritance , and composition.

1. Implement Encapsulation

Encapsulation is the principle of hiding the internal details of how an object works. It’s about making the fields in a class private and providing access to the fields via public methods.

1. Implement inheritance

Inheritance is a mechanism where one class acquires the properties (fields and methods) of another. With the use of inheritance , information is made manageable in a hierarchial order.

1. Implement polymorphism

Polymorphism allows methods to do different things based on the object that it is acting upon.

8. Create the objects

Once the classes are defined, you can create objects from those classes. This is called instantiation.

II **What is the Object Modeling Techniques (OMT).**

Object modelling technique is a methodology for designing and modelling an application’s structure, its data, and how it interacts with the user.

III **Compare object-oriented analysis and design (OOAD) and object analysis and design (OOP).**

OOAD is a technical approach used In the analysis and design of an application or system through the application of the object-oriented paradigm and concepts including visual modelling. This approach improves system quality and productivity of system analysis and design by making it more usable, resilient to change, and scalable.

OAD is a term often used in a broader context, not necessarily limited to the object – oriented paradigm. It involves a set of methodologies that also includes object – oriented methodologies, but it could potentially include other methodologies that use the concept of “objects”.

IV **Discuss Main goals of UML.**

* Provide a standard Notation that can be used by all software developers for modelling systems.
* Visualize the system architecture by providing set of diagrams to represent different views of the system, making it easier to understand the system’s architecture.
* Specify system details of the system design, including the interactions between objects, the sequence of operations, and data flow
* Construct the system – some UML tools can generate code in various languages from UML diagrams
* Document the system – The diagrams and specifications can serve as a valuable reference for the development team, and can also be used to communicate with stakeholders.

V **DESCRIBE three advantages of using object oriented to develop an information system.**

* Modularity for easier troubleshooting – When a problem happens in OOP – designed program, you can isolate the issue to a specific object or class, fix it there, and not worry about it affecting other parts of your code. This makes troubleshooting and debugging much easier.
* Reuse of code through inheritance – OOP allows classes to inherit commonly used state and behavior from other classes. In this way , you can create a general class first the define subclasses to inherit methods and fields from the general class. This promotes code reuse and is a significant time saver
* Flexibility through polymorphism – Polymorphism allows you to use an entity in multiple forms, and it’s one of the fundamental aspects of OOP. It let’s you redefine the way something works, by changing how it is done or by reusing what you’ve already done in a slightly different way.

VI **Briefly explain the following terms as used in object-oriented programming. Write a sample java code to illustrate the implementation of each concept.**

1. Constructor

a constructor in java is block of code similar to a method that’s called when an instance of an object is created .It has the same name as the class and has no return type

public class MyClass {

public MyClass() {

System.out.println("Constructor is called");

}

}

1. Object

An object is the instance of a class. It has state ( it stores values in fields) and behavior ( methods)

public class MyClass {

int x = 5;

}

public class Main {

public static void main(String[] args) {

MyClass myObj = new MyClass();

System.out.println(myObj.x);

}

}

1. Destructor
2. Polymorphism

Polymorphism in java is a concept by which we can perform a single action in different ways. We can perform in java by method overloading and method overriding.

class Animal {

void sound(){

System.out.println("Animal is making a sound");

}

}

class Dog extends Animal {

void sound(){

System.out.println("Dog is barking");

}

}

public class Main {

public static void main(String args[]) {

Animal myDog = new Dog();

myDog.sound();

}

}

1. Class

A class is a blueprint for creating objects. A class contains fields (variables) and methods to describe the behavior of an object

public class MyClass {

int x = 5;

}

1. Inheritance

Inheritance is a mechanism in which one class acquires the property of another class.

class Animal {

public void eat() {

System.out.println("Eating...");

}

}

class Dog extends Animal {

public void bark() {

System.out.println("Barking...");

}

}

public class Main {

public static void main(String args[]) {

Dog d = new Dog();

d.eat();

d.bark();

}

}

VII ***EXPLAIN* the three types of associations (relationships) between objects in object oriented.**

Association

This is a simple relationship between two or more objects.Each object has its own lifecycle and there is no owner. For example, a teacher and a student objects have an association relationship, as a teacher can be associated with multiple students and vice versa.



Aggregation

This is a special form of association where objects have their own lifecycle, but there is ownership. This is also known as “has-a” relationship. For example, a class object can have multiple student objects, indicating that a class has students.

Composition

This is a strong form of aggregation where the child object does not have its own lifecycle and any action on the parent will affect the child. This is also known as “part –of” relationship. For example ,a house object is composed of room objects, and if the house is destroyed, the rooms are also destroyed.

The following java example illustrates the relationship

// Association

class Teacher {

Student[] students;

}

class Student {

}

// Aggregation

class Class {

Student[] students; // Class has students

}

// Composition

class House {

Room[] rooms; // House is composed of rooms

}

class Room {

}

VIII **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.**

A class diagram is a type of static structure diagram in the Unified Modelling Language that describes the structure of a system by showing the system’s classes, their attributes, and the relationship among the classes. It’s widely used in software engineering for visualizing, specifying, and documenting software systems.

Steps to draw a class diagram

Identify classes

This involves identifying the primary objects within the system

Identifying relationships

This involves finding how the classes interact with each other, such as inheritance, aggregation and association.

Add Attributes and Methods

This involves defining the properties and behaviors of a class.

Example : consider a simple system of a library

Classes: book,member,librarian

Relationship: a member can borrow a book, a librarian can issue a book

Attributes and Methods

Book: attributes could be title, author, ISBN: methods could be getDetails(), issueBook()

IX

**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.**

#include <iostream>

#include <cmath>

// Abstract class

class Shape {

public:

// Pure virtual functions

virtual double area() = 0;

virtual double perimeter() = 0;

};

// Single Inheritance

class Circle : public Shape {

private:

double radius;

public:

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

// Method overriding

double area() override {

return M\_PI \* pow(radius, 2);

}

double perimeter() override {

return 2 \* M\_PI \* radius;

}

};

// Hierarchical Inheritance

class Rectangle : public Shape {

private:

double width, height;

public:

Rectangle(double w, double h) : width(w), height(h) {}

double area() override {

return width \* height;

}

double perimeter() override {

return 2 \* (width + height);

}

};

class Square : public Rectangle {

public:

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

};

// Friend function

void printAreaAndPerimeter(Shape& shape) {

std::cout << "Area: " << shape.area() << ", Perimeter: " << shape.perimeter() << std::endl;

}

int main() {

Circle circle(5.0);

Rectangle rectangle(4.0, 5.0);

Square square(4.0);

printAreaAndPerimeter(circle); // Late binding

printAreaAndPerimeter(rectangle); // Late binding

printAreaAndPerimeter(square); // Late binding

return 0;

}

Shape is an abstract class with pure functions area() and perimeter()

Circle, Rectange and Square are concrete classes that inherit from shape and override its pure virtual functions

printAreaAndPerimeter() is a friend function that can access the private and protected members of shape

The calls to area() and perimeter() in printAreaAndPerimeter() are examples of late binding, because the actual function that gets called is determined at runtime based on the type of the object.

1. X **Using a program written in C++, differentiate between the following. [6 Marks]**
   1. **Function overloading and operator overloading**
   2. **Pass by value and pass by reference**
   3. **Parameters and arguments**
2. Function overloading and operator overloading

Function overloading is when multiple functions have the same name but different parameters. Operator overloading is when an operator is redefined to provide a special meaning beyond its standard operational meaning

// Function Overloading

void print(int i) {

std::cout << "Printing int: " << i << std::endl;

}

void print(double f) {

std::cout << "Printing float: " << f << std::endl;

}

// Operator Overloading

class Complex {

private:

int real, imag;

public:

Complex(int r = 0, int i =0) {real = r; imag = i;}

Complex operator + (Complex const &obj) {

Complex res;

res.real = real + obj.real;

res.imag = imag + obj.imag;

return res;

}

};

b. Pass by value and pass by reference

pass by value means that a copy of the data is made and the copy is passed into the function. Pass by reference means the reference of an argument is passed into the function, not the actual value

// Pass by Value

void func(int num) {

num = 10;

}

// Pass by Reference

void funcRef(int& num) {

num = 10;

}

C. Parameters and arguments

Parameters are the variables that are part of the method/function declaration

Arguments are the actual values that are passed in when the function is invoked

void myFunction(int myParam) { // myParam is a parameter

// ...

}

int main() {

myFunction(5); // 5 is an argument

return 0;

}

XI

public class CalculateG {

public double multi(double a, double b) {

return a \* b;

}

public double powerToSquare(double a) {

return a \* a;

}

public double sum(double a, double b) {

return a + b;

}

public void outline(String message, double result) {

System.out.println(message + result);

}

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 initialPosition = 0.0;

CalculateG calc = new CalculateG();

double finalPosition = calc.sum(calc.multi(0.5 \* gravity, calc.powerToSquare(fallingTime)), calc.multi(initialVelocity, fallingTime));

finalPosition = calc.sum(finalPosition, initialPosition);

double finalVelocity = calc.sum(calc.multi(gravity, fallingTime), initialVelocity);

calc.outline("The object's position after " + fallingTime + " seconds is ", finalPosition);

calc.outline("The object's velocity after " + fallingTime + " seconds is ", finalVelocity);

}

}

PART B

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>

long long sumEvenFibonacci(long long limit) {

long long sum = 0;

long long a = 1; // First term

long long b = 2; // Second term

while (b <= limit) {

if (b % 2 == 0) {

sum += b;

}

long long nextTerm = a + b;

a = b;

b = nextTerm;

}

return sum;

}

int main() {

long long limit = 4000000;

std::cout << "Sum of even-valued Fibonacci terms not exceeding " << limit << " is " << sumEvenFibonacci(limit) << std::endl;

return 0;

}

3.

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

#include <iostream>

int main() {

int array[15];

std::cout << "Enter 15 integers:\n";

for(int i = 0; i < 15; i++) {

std::cout << "Enter integer " << (i+1) << ": ";

std::cin >> array[i];

}

std::cout << "You entered:\n";

for(int i = 0; i < 15; i++) {

std::cout << "Integer " << (i+1) << ": " << array[i] << "\n";

}

return 0;

}