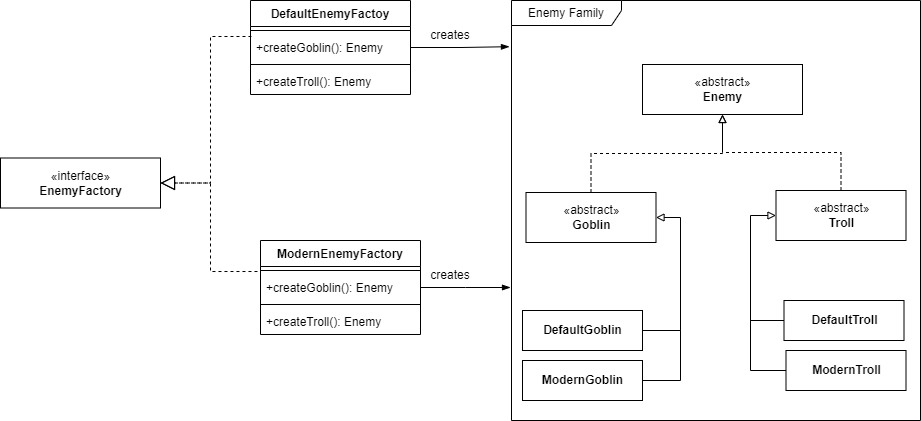
**DIT: OOP Assignment**

**Name:Arbogasti Evaline Atieno**

**Regno:SCT121-0990/2022**

**Part A:**

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



* **Diagram Component**: *Establish Relationships*
* **Explanation**: Determine how the objects/classes are related to each other. This includes associations like one-to-one, one-to-many, or many-to-many relationships. For instance, a **Library** has a relationship with multiple **Books** and **Members**.

Step 4: Encapsulation

* **Diagram Component**: *Encapsulation*
* **Explanation**: Encapsulation involves bundling the data (attributes) and methods (behaviors) within a class, allowing access to the data only through the defined methods. This helps in hiding the internal complexity and ensures data integrity and security.

Step 5: Abstraction

* **Diagram Component**: *Abstraction*
* **Explanation**: Abstract away unnecessary details and focus on essential attributes and behaviors. Define interfaces to interact with objects while hiding implementation details. For instance, an interface for a **Book** might have methods like **checkAvailability**, **getDetails**, etc.

Step 6: Inheritance

* **Diagram Component**: *Inheritance*
* **Explanation**: Use inheritance to create a hierarchy among classes. This allows for the creation of new classes (derived/child classes) based on existing classes (base/parent classes). The derived classes inherit attributes and behaviors from the base class and can have their own unique properties or methods.

Step 7: Polymorphism

* **Diagram Component**: *Polymorphism*
* **Explanation**: Implement polymorphism, which allows objects to be treated as instances of their parent class. This enables flexibility in the program, where a method can perform different actions based on the specific instance of the object.

1. **What are the Object Modeling Techniques (OMT)? [1 Marks]**

Object Modeling Technique (OMT) is a method for modeling and designing software systems using object-oriented concepts.

1. **Compare object-oriented analysis and design (OOAD) and object analysis and design (OOP). [2 Marks]**

OOAD primarily concentrates on the entire software development lifecycle, covering both analysis and design phases while OOP is a narrower term that usually refers specifically to the design phase within the object-oriented development process.

1. **Discuss Mian goals of UML. [2 Marks]**

The main goals of the Unified Modeling Language (UML) revolve around providing a standardized and comprehensive approach to visualizing, specifying, constructing, and documenting the artifacts involved in software systems and processes.

1. **DESCRIBE three advantages of using object oriented to develop an information system. [3Marks]**

**Modularity**: OOP allows breaking down complex systems into smaller, more manageable modules or objects. Each object encapsulates data and functionalities related to a specific entity or behavior.

**Encapsulation**: OOP's encapsulation hides the internal workings of an object from the outside world, exposing only necessary interfaces. It combines data and methods within a class, controlling access to the data.

**Inheritance and Polymorphism**: OOP supports inheritance, allowing new classes to inherit properties and behaviors from existing classes, promoting code reuse and extensibility. Polymorphism enables objects to be treated as instances of their parent classes.

1. **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**
   2. **object**
   3. **Destructor**
   4. **polymorphism**
   5. **class**
   6. **Inheritance**

**a. Constructor:**

**Explanation**: Constructors are special methods in a class that are automatically called when an instance (object) of the class is created. They initialize the object's state by assigning initial values to the object's attributes.

Example:

public class Car {

private String brand;

private String model;

// Constructor - initializes brand and model attributes

public Car(String brand, String model) {

this. brand = brand;

this. model = model;

}

public void displayInfo() {

System.out.println("Brand: " + brand + ", Model: " + model);

}

public static void main(String[] args) {

// Creating an instance (object) of the Car class using the constructor

Car myCar = new Car("Toyota", "Corolla");

myCar.displayInfo();

}

}

**b. Object:**

* **Explanation**: An object is an instance of a class. It's a real-world entity or concept that has attributes and behaviors defined by its class.

Example:

public class Person {

private String name;

private int age;

public Person(String name, int age) {

this.name = name;

this.age = age;

}

public void displayInfo() {

System.out.println("Name: " + name + ", Age: " + age);

}

public static void main(String[] args) {

// Creating objects (instances) of the Person class

Person person1 = new Person("Alice", 25);

Person person2 = new Person("Bob", 30);

// Accessing object behaviors

person1.displayInfo();

person2.displayInfo();

}

}

**c. Destructor:**

* **Explanation**: In Java, there isn't a direct equivalent of a destructor like in some other programming languages. Java uses automatic garbage collection to reclaim memory occupied by objects that are no longer referenced.

**d. Polymorphism:**

Explanation: Polymorphism refers to the ability of objects to take on multiple forms. In Java, it can be achieved through method overriding or method overloading.

class Animal {

public void makeSound() {

System.out.println("Some sound");

}

}

class Dog extends Animal {

@Override

public void makeSound() {

System.out.println("Woof!");

}

}

class Cat extends Animal {

@Override

public void makeSound() {

System.out.println("Meow!");

}

}

public class PolymorphismExample {

public static void main(String[] args) {

Animal myDog = new Dog();

Animal myCat = new Cat();

myDog.makeSound(); // Outputs: Woof!

myCat.makeSound(); // Outputs: Meow!

}

}

**e. Class:**

* **Explanation**: A class is a blueprint/template that defines the properties (attributes/fields) and behaviors (methods/functions) that an object of that class will have.

public class Rectangle {

private int length;

private int width;

public Rectangle(int length, int width) {

this.length = length;

this.width = width;

}

public int calculateArea() {

return length \* width;

}

public static void main(String[] args) {

// Creating an object of the Rectangle class

Rectangle rect = new Rectangle(5, 10);

int area = rect.calculateArea();

System.out.println("Area of Rectangle: " + area);

}

}

**Inheritance:**

* **Explanation**: Inheritance allows a class (subclass/child class) to inherit attributes and methods from another class (superclass/parent class). It promotes code reuse and supports the "is-a" relationship.

class Vehicle {

void move() {

System.out.println("Vehicle is moving...");

}

}

class Car extends Vehicle {

void accelerate() {

System.out.println("Car is accelerating...");

}

}

public class InheritanceExample {

public static void main(String[] args) {

Car myCar = new Car();

myCar.move(); // Inherited method

myCar.accelerate(); // Method specific to Car class

}

}

1. **Explain the three types of associations (relationships) between objects in object oriented. [6 Marks]**

* **Association**: It is a relationship between two objects where both objects have their lifecycle and there is no ownership between the objects. For example, a teacher and a student can be associated with each other, but there is no ownership between them.
* **Aggregation**: It is a specialized form of association where all objects have their lifecycle. However, there is ownership, and child objects can belong to only one parent object at a time.
* **Composition**: It is again a specialized form of aggregation and is also called a “death” relationship. It is a strong type of aggregation where child objects cannot exist without the parent object.

1. **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.**

In software engineering, a class diagram in the is a type of static structure diagram that describes the structure of a system by showing the system's classes, their attributes, operations (or methods), and the relationships among objects.

Here are the steps to draw a class diagram:

1. Identify the objects in the problem domain, and create classes for each of them. For example, in a library management system, we can identify objects such as books, authors, publishers, and members.
2. Add attributes for those classes. For instance, the book class can have attributes such as title, author, publisher, and ISBN.
3. Add operations for those classes. For example, the book class can have operations such as borrowBook(), returnBook(), and reserveBook().
4. Determine how each of the classes or objects are related to one another. For instance, the book class can be related to the author class through an association relationship.
5. **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]**
   2. **Friend functions [5 Marks]**
   3. **Method overloading and method overriding [10 Marks]**
   4. **Late binding and early binding [8 Marks]**

**Abstract class and pure functions**

Here is a sample implementation of an area and perimeter calculator using C++ and OOP concepts:

#include <iostream>

#include <cmath>

using namespace std;

// Abstract class Shape

class Shape {

public:

virtual double area() = 0;

virtual double perimeter() = 0;

};

// Class Circle derived from Shape

class Circle : public Shape {

private:

double radius;

public:

Circle(double r) {

radius = r;

}

double area() {

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

}

double perimeter() {

return 2 \* M\_PI \* radius;

}

};

// Class Rectangle derived from Shape

class Rectangle : public Shape {

private:

double length, width;

public:

Rectangle(double l, double w) {

length = l;

width = w;

}

double area() {

return length \* width;

}

double perimeter() {

return 2 \* (length + width);

}

};

// Class Triangle derived from Shape

class Triangle : public Shape {

private:

double a, b, c;

public:

Triangle(double a, double b, double c) {

this->a = a;

this->b = b;

this->c = c;

}

double area() {

double s = (a + b + c) / 2;

return sqrt(s \* (s - a) \* (s - b) \* (s - c));

}

double perimeter() {

return a + b + c;

}

};

int main() {

// Circle

Circle c(5);

cout << "Circle: " << endl;

cout << "Area: " << c.area() << endl;

cout << "Perimeter: " << c.perimeter() << endl;

// Rectangle

Rectangle r(3, 4);

cout << "Rectangle: " << endl;

cout << "Area: " << r.area() << endl;

cout << "Perimeter: " << r.perimeter() << endl;

// Triangle

Triangle t(3, 4, 5);

cout << "Triangle: " << endl;

cout << "Area: " << t.area() << endl;

cout << "Perimeter: " << t.perimeter() << endl;

return 0;

}

1. **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**

Here’s a C++ program that differentiates between function overloading and operator overloading, pass by value and pass by reference, and parameters and arguments.

#include <iostream>

using namespace std;

// Function overloading

int add(int x, int y) {

return x + y;

}

double add(double x, double y) {

return x + y;

}

// 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;

}

};

// Pass by value

void swap(int x, int y) {

int temp = x;

x = y;

y = temp;

}

// Pass by reference

void swap(int &x, int &y) {

int temp = x;

x = y;

y = temp;

}

int main() {

// Function overloading

cout << add(1, 2) << endl;

cout << add(1.0, 2.0) << endl;

// Operator overloading

Complex c1(10, 5), c2(2, 4);

Complex c3 = c1 + c2;

cout << c3.real << " + i" << c3.imag << endl;

// Pass by value

int a = 10, b = 20;

swap(a, b);

cout << "a = " << a << ", b = " << b << endl;

// Pass by reference

int c = 10, d = 20;

swap(c, d);

cout << "c = " << c << ", d = " << d << endl;

// Parameters and arguments

int x = 10, y = 20;

cout << add(x, y) << endl;

return 0;

}

1. **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.

Here’s the modified version of the code that computes the position and velocity of an object after falling for 30 seconds, outputting the position in meters:

class CalculateG {

public static void main(String[] args) {

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

double fallingTime = 30;

double initialVelocity = 0.0;

double finalVelocity = gravity \* fallingTime + initialVelocity;

double initialPosition = 0.0;

double finalPosition = 0.5 \* gravity \* fallingTime \* fallingTime + initialVelocity \* fallingTime + initialPosition;

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.");

}

}

To extend the CalculateG class with the methods for multiplication, powering to square, summation, and printing out a result, you can use the following code:

public class CalculateG {

public double multi(double a, double b) {

return a \* b;

}

public double square(double a) {

return a \* a;

}

public double sum(double a, double b) {

return a + b;

}

public void outline(double result) {

System.out.println("The result is " + result);

}

public static void main(String[] args) {

CalculateG calc = new CalculateG();

double a = 2.0;

double b = 3.0;

double result1 = calc.multi(a, b);

double result2 = calc.square(a);

double result3 = calc.sum(a, b);

calc.outline(result1);

calc.outline(result2);

calc.outline(result3);

}

}

**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.**

12 #include <iostream>

#include <vector>

using namespace std;

vector<int> generate\_fibonacci\_sequence(int max\_value) {

vector<int> sequence;

int a = 1, b = 2, c = 0;

sequence.push\_back(a);

sequence.push\_back(b);

while (c <= max\_value) {

c = a + b;

a = b;

b = c;

if (c % 2 == 0) {

sequence.push\_back(c);

}

}

return sequence;

}

Once we have the Fibonacci sequence, we can sum up all the even-valued terms in the sequence.

int sum\_even\_fibonacci\_terms(vector<int> sequence) {

int sum = 0;

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

if (sequence[i] % 2 == 0) {

sum += sequence[i];

}

}

return sum;

}

We can call these functions to get the sum of all the even-valued terms in the Fibonacci sequence whose values do not exceed four million:

int max\_value = 4000000;

vector<int> sequence = generate\_fibonacci\_sequence(max\_value);

int sum = sum\_even\_fibonacci\_terms(sequence);

cout << "The sum of all the even-valued terms in the Fibonacci sequence whose values do not exceed four million is " << sum << "." << endl;

We can use a more efficient algorithm that generates only the even-valued terms in the Fibonacci sequence. Here is a C++ code snippet that implements this algorithm:

int sum\_even\_fibonacci\_terms(int max\_value) {

int sum = 0;

int a = 1, b = 2, c = 0;

while (b <= max\_value) {

sum += b;

c = a + 2 \* b;

a = b;

b = c;

}

return sum;

}

We can call this function to get the sum of all the even-valued terms in the Fibonacci sequence whose values do not exceed four million:

int max\_value = 4000000;

int sum = sum\_even\_fibonacci\_terms(max\_value);

cout << "The sum of all the even-valued terms in the Fibonacci sequence whose values do not exceed four million is " << sum << "." << endl;

13 **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>

#include <algorithm>

using namespace std;

int main() {

int arr[15], arr2[15];

int i, num, sum = 0, prod = 1;

// Take input from user

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

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

cin >> arr[i];

}

// Print the values stored in the array

cout << "Values stored in the array: ";

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

cout << arr[i] << " ";

}

cout << endl;

// Ask user to enter a number and check if it is present in the array

cout << "Enter a number to search: ";

cin >> num;

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

if (arr[i] == num) {

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

break;

}

}

if (i == 15) {

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

}

// Create another array and copy all the elements from the existing array but in reverse order

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

arr2[i] = arr[14 - i];

}

// Print the elements of the new array

cout << "Elements of the new array: ";

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

cout << arr2[i] << " ";

}

cout << endl;

// Get the sum and product of all elements of the array

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

sum += arr[i];

prod \*= arr[i];

}

// Print the sum and product of all elements of the array

cout << "Sum of all elements of the array: " << sum << endl;

cout << "Product of all elements of the array: " << prod << endl;

return 0;

}