Ivy maina

Sct0940-2022

Dit 2.1

DIT: OOP Assignment Part

A:

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

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

ii. **What is the Object Modeling Techniques (OMT). [1 Marks]**

Object Modeling Techniques (OMT) is a method for modeling and designing object-oriented systems.

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

Object-Oriented Analysis and Design (OOAD) is a comprehensive approach to software development that combines object-oriented principles with analysis and design processes.

OOAD is a higher-level process that encompasses both analysis and design phases, while OOP specifically refers to the implementation phase where code is written using object-oriented principles.

Discuss Mian goals of UML. [2 Marks]

The main goals of the Unified Modeling Language (UML) are to provide a standardized and widely accepted notation that facilitates communication and understanding within the software development community.

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

Modularity and Reusability

Improved Maintenance and Adaptability

Encapsulation for Data Security

v. **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]** a. **Constructor**

```
A constructor is a special method in a class that is use      d for   initializing
objects. It is called automatically when an object is created.
public class Car {
    // Fields
    String model;
    int year;

    // Constructor
    public Car(String carModel, int carYear) {
        model = carModel;
        year = carYear;
        System.out.println("A new car object is created!");
    }

    // Other methods can follow...
}
```

b. **object**

An object is an instance of a class. It represents a real-world entity and has both state (attributes) and behavior (methods).

```
public class Main {     public static
void main(String[] args) {         //
Creating an object of the Car class
        Car myCar = new Car("Toyota Camry", 2022);

        // Accessing object properties
        System.out.println("Model: " + myCar.model);
        System.out.println("Year: " + myCar.year);
    }
```

c. **Destructor**

In Java, there is no explicit destructor like in some other languages. Java uses garbage collection to automatically reclaim memory occupied by objects that are no longer in use

d. **polymorphism**

Polymorphism allows objects of different types to be treated as objects of a common type. It includes method overloading (compile-time polymorphism) and method overriding (runtime polymorphism).

```
class Animal {
    void makeSound() {
        System.out.println("Some generic sound");
    }
}

Class dog exdends animal{
@Override  void make sound(){
System out print\n(woof woof!”)
}
                    }
```

```
    public static void main(String[] args) {
        Animal myAnimal = new Dog(); // Polymorphic behavior
        myAnimal.makeSound(); // Calls Dog's overridden method
    }
}
```

e. **class**

```
A class is a blueprint or template for creating objects. It defines the attributes and behaviors that the objects of the class will have.
```

f. **Inheritance**

```
Inheritance is a mechanism that allows a class (subclass/derived class) to inherit properties and behaviors from another class (superclass/base class).
class Vehicle {
    void start() {
        System.out.println("Vehicle starting...");
    }
}

class Car extends Vehicle {
    void drive() {
        System.out.println("Car driving...");
    }
}

public class InheritanceExample {
    public static void main(String[] args) {
        Car myCar = new Car();
        myCar.start(); // Inherited method
        myCar.drive(); // Method from the Car class
    }
}
```

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

You

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

In object-oriented programming, the three main types of associations between objects are:

i. *Association:*
   - *Definition:* A basic relationship where one object is related to another.
   - *Nature:* It can be a one-way or a two-way connection.
   - *Example:* A `Teacher` class being associated with a `Student` class.

ii. *Aggregation:*
   - *Definition:* A specialized form of association, representing a "whole-part" relationship.
   - *Nature:* It implies a more significant relationship where one class encompasses another.
   - *Example:* A `University` class having an aggregation relationship with a `Department` class.

iii. *Composition:*
   - *Definition:* A stronger form of aggregation, indicating a "whole-part" relationship with exclusive ownership.
   - *Nature:* When the container object is destroyed, the contained object is also destroyed.
   - *Example:* A `Car` class having a composition relationship with an `Engine` class, where the engine is an integral part of the car.

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 UML (Unified Modeling Language) diagram that represents the structure and relationships of classes in a system. It provides a static view of the system, showcasing classes, attributes, operations, and the associations among them.

iv. **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.**

Inheritance (Single inheritance, Multiple inheritance and Hierarchical inheritance) [10 Marks] • Friend functions [5 Marks]
Method overloading and method overriding [10 Marks]

•

Late binding and early binding **[8 Marks]**

```
Abstract class and pure functions
#include <iostream>
#include <cmath>

// Base class Shape • class Shape {
public:
virtual float area() const = 0;
virtual float perimeter() const = 0;
};

// Derived class Circle • class Circle : public Shape {
private:
float radius;

public:
Circle(float r) : radius(r) {}

float area() const override {
return 3.14 * radius * radius;
}

float perimeter() const override {
return 2 * 3.14 * radius;
}
};

// Derived class Rectangle • class Rectangle : public Shape {
private:
float length, width;

public:
Rectangle(float l, float w) : length(l), width(w) {}

float area() const override {
return length * width;
}

float perimeter() const override {
    return 2 * (length + width);
```

```
    }
    };

    // Derived class Triangle • class Triangle : public Shape {
    private:
    float side1, side2, side3;

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

    float area() const override {
    float s = (side1 + side2 + side3) / 2;
    return sqrt(s * (s - side1) * (s - side2) * (s - side3));
    }

    float perimeter() const override {
    return side1 + side2 + side3;
    }
    };

    // Derived class Square • class Square : public Rectangle {
    public:
    Square(float side) : Rectangle(side, side) {}
    };

    // Friend function to compare areas of two shapes
    bool compareAreas(const Shape& shape1, const Shape& shape2) {
    return shape1.area() > shape2.area();
    }

    int main() {
    Circle circle(5);
    Rectangle rectangle(4, 6);
    Triangle triangle(3, 4, 5);
    Square square(5);

    std::cout << "Circle - Area: " << circle.area() << ", Perimeter: " <<
    circle.perimeter() << std::endl;
  std::cout << "Rectangle - Area: " << rectangle.area() << ", Perimeter: " <<
rectangle.perimeter() << std::endl;
```

```
std::cout << "Triangle - Area: " << triangle.area() << ", Perimeter: " <<
triangle.perimeter() << std::endl;
std::cout << "Square - Area: " << square.area() << ", Perimeter: "
<<square.perimeter() << std::endl;

// Friend function usage to compare areas
if (compareAreas(circle, rectangle)) {
std::cout << "Circle has a larger area than Rectangle." << std::endl;
} else {
std::cout << "Rectangle has a larger area than Circle." << std::endl;
}

return 0;
}
```

[6 Marks]

v. **Using a program written in C++, differentiate between the following. [6 Marks]**

- Function overloading and operator overloading
- Pass by value and pass by reference
- Parameters and arguments

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:

$x(t)=0.5*at^2+v_it+x_i\ v(t)=at+v_i$

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.

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, ...

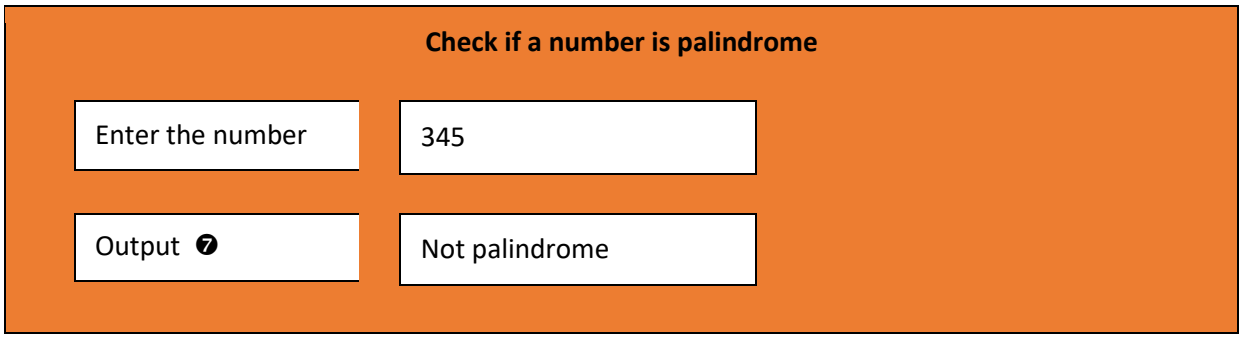
By considering the terms in the Fibonacci sequence whose values do not exceed four million, write a C++ method to find the sum of all the even- valued terms.

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:



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.

a) Print the values stored in the array on screen.
b) 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"
c) 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
d) 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 originalArray[size];

// Part (a): Input values into the array and print them
std::cout << "Enter 15 integer values:\n";
for (int i = 0; i < size; ++i) {
std::cout << "Value " << i + 1 << ": ";
std::cin >> originalArray[i];
}
```

```
q)
            std::cout << "\nValues stored in the array:\n";
            for (int i = 0; i < size; ++i) {
            std::cout << originalArray[i] << " ";
            }
v)
          // Part (b): Check if a number is present in the array
          int searchNumber;
          std::cout << "\nEnter a number to search in the array: ";
          std::cin >> searchNumber;
          bb)     bool found = false; cc)     int foundIndex = -1; dd)
  for (int i = 0; i < size; ++i) { ff)        if
(originalArray[i] == searchNumber) {
           found = true;
         foundIndex = i; ii)
break;        }
    }
                    if (found) {
                    std::cout << "The number found at index " << foundIndex <<
                    ".\n"; oo)     } else {
 std::cout << "Number not found in this array.\n"; }
// Part (c): Create a new array in reverse order tt)
int reversedArray[size]; uu)     for (int i = 0; i < size;
++i) {
                    reversedArray[i] = originalArray[size - 1 - i];
                    }

   std::cout << "\nElements of the new array (in reverse order):\n";
   for (int i = 0; i < size; ++i) {

 std::cout << reversedArray[i] << " "; bbb)     }

                    // Part (d): Calculate sum and product
int sum = 0;
                    int product = 1;
  for (int i = 0; i < size; ++i) { iii)
sum += originalArray[i]; product *=
originalArray[i];
kkk)              }
lll)
std::cout << "\nSum of the elements: " << sum << "\n"; nnn)
std::cout << "Product of the elements: " << product << "\n";
```

```
return 0; }
```