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]**

1. **Identify Objects:**
 - Identify the key entities or objects in the system. Objects represent real-world entities or concepts that you want to model in your system. These could include nouns like "Customer," "Order," or "Product."
2. **Define Classes:**
 - For each identified object, define a corresponding class. A class is a blueprint or template for creating objects. It encapsulates the data (attributes) and behavior (methods) related to the object. Use proper naming conventions for classes.
3. **Identify Relationships:**
 - Determine the relationships between classes. Relationships define how classes interact with each other. Identify associations, aggregations, or compositions between classes to represent the connections in your system.
4. **Model the Behavior:**
- Identify the behavior of each class by defining methods. Methods represent the operations or actions that can be performed by the objects of a class. Consider encapsulation, abstraction, and information hiding principles while defining methods.
5**. Encapsulation:**
 - Encapsulate the data within classes. Use access modifiers to control the visibility of attributes and methods. Encapsulation helps in hiding the internal details of a class and exposing only what is necessary.
6. **Inheritance:**
 - Identify commonalities between classes and apply inheritance where appropriate. Inheritance allows a class to inherit properties and behaviors from another class, promoting code reuse and establishing an "is-a" relationship.
7. **Polymorphism:**
 - Use polymorphism to allow objects of different classes to be treated as objects of a common base class. This enables methods to be called on objects of different types through a common interface, facilitating flexibility and extensibility.
8. **Abstraction:**
- Abstract away unnecessary details and focus on essential features. Define abstract classes or interfaces to represent common characteristics among classes.
9. **Create Instances (Objects):**
 - Instantiate objects based on the defined classes. Objects are instances of classes, and they represent specific occurrences of the entities modeled in your system.
10. **Implement and Test:**

- Implement the classes and their methods. Test the functionality of your system by creating instances of objects, invoking methods, and ensuring that the system behaves as expected.

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

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

- OMT provides a set of graphical notations and processes for visualizing, specifying, constructing, and documenting the artifacts of a software system.

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

	Object-Oriented Analysis and Design (OOAD)	**Object-Oriented Programming (OOP)**
Focus	OOAD is a holistic approach that focuses on analyzing and designing a system using object-oriented principles before implementation. - It involves understanding the problem domain, identifying objects, defining their attributes and behaviors, and creating a blueprint for the entire system before coding begins.	OOP, on the other hand, is the actual implementation phase of the object-oriented development process. It involves translating the design produced during OOAD into executable code.
Process	OOAD typically follows a systematic process that includes requirements gathering, analysis, system design, and object design. - It emphasizes creating models to represent the static and dynamic aspects of the system using techniques such as use case diagrams, class diagrams, sequence diagrams, and state diagrams.	OOP follows the principles of **encapsulation, inheritance, and polymorphism** to structure code in a way that reflects the designed object model. - It involves writing and organizing code using classes, defining methods, and implementing the relationships and behaviors specified during the design phase.

Goal	The primary goal of OOAD is to create a robust and flexible design that can be translated into an efficient and maintainable implementation.	The primary goal of OOP is to create a working software system based on the design specifications developed during OOAD. - It emphasizes code reuse, modularity, and maintainability.

Comparison:

- OOAD is a higher-level process that precedes OOP and involves activities such as requirements analysis, system modeling, and architectural design.
- OOP is the actual coding phase where the designs from OOAD are implemented using an object-oriented programming language.
- OOAD focuses on abstraction, modeling, and design concepts, while OOP focuses on the actual coding and implementation of those designs.

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

Unified Modeling Language (UML) has several primary goals. The major goals of UML are:

Standardization:

UML aims to provide a standardized notation that can be used by software developers, analysts, and other stakeholders across different organizations and industries.

- **This standardization helps ensure that system models are understood in the same way and facilitates communication among team members.**

Consistency in Modeling:

In this case, the UML is used as a tool for modeling software systems in a consistent manner.

- It has well defined diagrams and notations representing various aspects of a system such as **structure, behavior, and interactions**. The aim here is to ensure consistency which leads to making meaning from the models.

Flexibility:

UML is designed to be flexible and adaptable to different modeling needs.

- It provides a variety of diagram types, including **class diagrams, sequence diagrams, use case diagrams**, etc. allowing developers to choose the most suitable representations for their specific requirements.

Tool-Independence:

UML is intended to be tool-independent, this means that UML diagrams can be created and understood using various modeling tools that support the UML standard. This promotes compatibility and allows teams to use different tools based on their preferences or requirements.

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

Reusability: OOP languages allow developers to create classes that can be used as templates for creating new objects. This means that code can be reused across different parts of the system, which saves time and reduces the likelihood of errors.

Scalability: OOP languages are designed to be scalable, which means that they can be used to develop systems of any size. This makes them ideal for developing large, complex information systems that need to be able to handle a lot of data and users.

Flexibility: OOP provides flexibility in designing and evolving systems. The use of polymorphism allows objects to be treated as instances of their base class, providing a common interface for various objects. This flexibility enables you to replace or extend components without affecting the overall system, making it easier to adapt to changing requirements.

Modularity: OOP languages allow developers to break down a program into smaller, more manageable pieces called objects. This modularity makes it easier to troubleshoot problems because each object is self-contained and can be tested independently. It also allows multiple developers to work on different objects simultaneously without interfering with each other's work.

vi. **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 that is called when an object of a class is created. The purpose of the constructor is to initialize the object`s data members and prepare It for use.

- Constructors are used to set the initial values of an object's properties and to perform any other initialization tasks that are required.

b. **Object -** Any entity that has state and behavior is known as an object. For example, a chair, pen, table, keyboard, bike, etc. It can be physical or logical.

c. **Destructor**- Destructor is an **instance member function** that is invoked automatically whenever an object is going to be destroyed.
A destructor is also a special member function like a constructor.
It is used to destroy the class objects created by the constructor.

- A destructor is the last function that is going to be called before an object is destroyed.

d. **Polymorphism** – Polymorphism is the ability to take more than one form, therefore **polymorphism in (OOP) is the ability of different objects to respond to the same message or method invocation in different ways.**

e. **Class -** In object-oriented programming (OOP), **a class is a blueprint or template for creating objects that share common properties and behaviors.** A class defines a set of attributes (data members) and methods (member functions) that are common to all objects of that class.

f. **Inheritance - inheritance is a mechanism that allows a class to inherit properties and behaviors from another class.**

- Allows a class to inherit properties and behaviors from another class. The class that is being inherited from is called the parent class or superclass, while the class that inherits from it is called the child class or subclass.
- Inheritance **promotes code reuse and establishes relationships between classes**. It allows developer to create new classes based on the existing one, which can save time and reduce the likelihood of errors.

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

Association: Association is a generic term for a relationship between two or more classes. It represents a bi-directional connection, **implying that objects of one class are somehow related to objects of another class**. Associations can be one-to-one, one-to-many, or many-to-many.

Aggregation: Aggregation is a specific form of **association where one class represents a whole, and another class represents a part of that whole**. The part (or component) class is not dependent on the whole, and it can exist independently. Aggregation is often represented by a diamond-shaped arrow pointing to the whole class.

Composition: Composition is a stronger form of aggregation where the part class is strongly tied to the whole class. In composition, the part class cannot exist independently of the whole class. If the whole is destroyed, the parts are also destroyed. Composition is often represented by a filled diamond-shaped arrow pointing to the whole class.

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 of a system by depicting the classes, their attributes, methods, and the relationships between classes.

- **It describes the structure of an object-oriented system by showing the classes in that system and the relationships between the classes.**
- **It provides a static view of a system, illustrating the various entities and their interactions in terms of classes and their associations.**

Steps to draw the class diagram:

1. **Identify classes:** Identify the objects in the problem domain, and create classes for each of them. (e.g. Teacher, Student, Course for an enrollment system) **Classes represents objects in the system and should be named based on their purpose or functionality.**
2. **Add attributes** : once a class is created add attributes for those classes (e.g. name, address, telephone for the Student class) Attributes are the data members of a class and should be listed under the class name.
3. **Add operations/ Methods** for those classes (e.g. addStudent(student) for the Course class) Methods are the member functions of a class and should be listed under the class name.
4. **Add relationships:** Connect the classes with appropriate relationships (e.g. Relate Teacher and Course with an association) Relationships can be inheritance, association, aggregation, or composition
5. **Refine the diagram:** After adding the classes, attributes, methods, and relationships, the final step is to refine the diagram. This involves reviewing the diagram to ensure that it accurately represents the system and making any necessary changes.

The class diagram example below shows the classes involved in a sales order system:

1. Classes in the diagram below:
   - OrderStatus

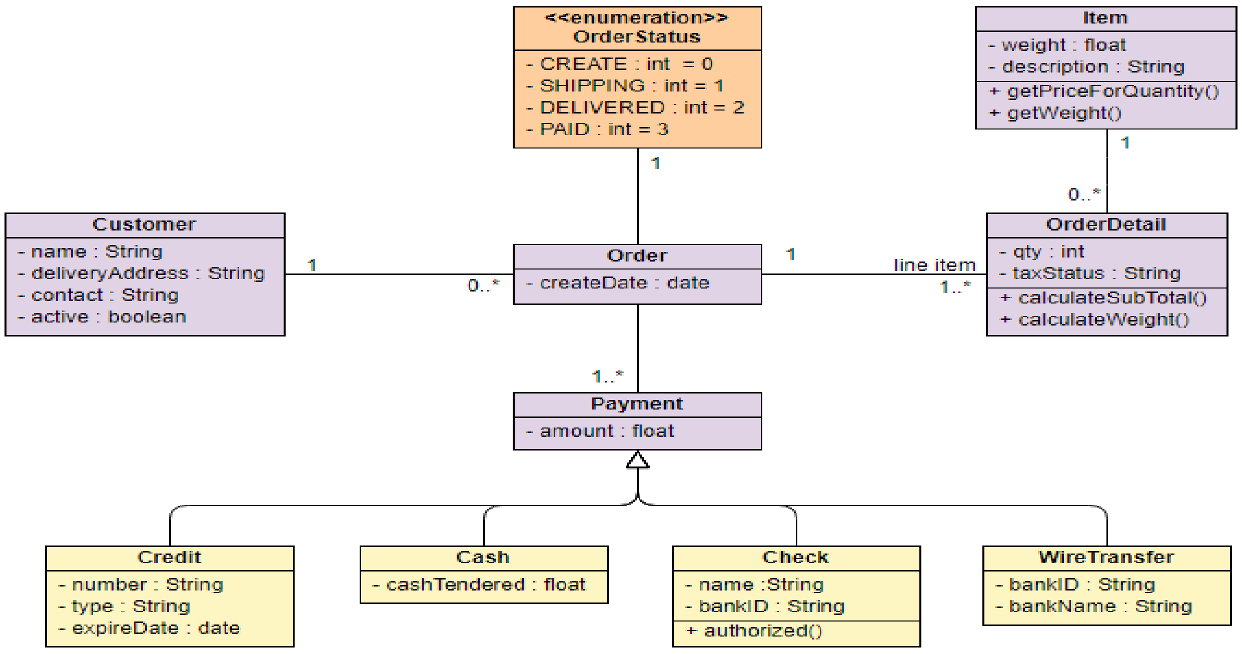
- Items
- Customer
- Orders
- orderDetails
- payments
- credit
- cash
- check and wire transfer

2. Attributes in the below class diagram:

   For example in class order status there are the following attributes:

   a. Create
   b. Shipping
   c. Delivered
   d. Paid

3.



vii. **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.**

   a. **Inheritance (Single inheritance, Multiple inheritance and Hierarchical inheritance) [10 Marks]**
   b. **Friend functions [5 Marks]**

c. Method overloading and method overriding [10 Marks]
d. Late binding and early binding [8 Marks]
e. Abstract class and pure functions [6 Marks]

```
#include <iostream>
#include <cmath>

using namespace std;

// Abstract class Shape
class Shape {
public:
   virtual double area() = 0;
   virtual double perimeter() = 0;
};

// Derived class Circle
class Circle : public Shape {
private:
   double radius;
public:
   Circle(double r) : radius(r) {}
   double area() override { return M_PI * radius * radius; }
   double perimeter() override { return 2 * M_PI * radius; }
};

// Derived class Rectangle
class Rectangle : public Shape {
private:
   double length, width;
public:
   Rectangle(double l, double w) : length(l), width(w) {}
   double area() override { return length * width; }
   double perimeter() override { return 2 * (length + width); }
};

// Derived class Triangle
class Triangle : public Shape {
private:
   double a, b, c;
public:
   Triangle(double side1, double side2, double side3) : a(side1), b(side2),
c(side3) {}
```

```
    double area() override {
        double s = (a + b + c) / 2;
        return sqrt(s * (s - a) * (s - b) * (s - c));
    }
    double perimeter() override { return a + b + c; }
};

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

int main() {
    // Circle example
    Circle c(5);
    cout << "Circle area: " << c.area() << endl;
    cout << "Circle perimeter: " << c.perimeter() << endl;

    // Rectangle example
    Rectangle r(4, 6);
    cout << "Rectangle area: " << r.area() << endl;
    cout << "Rectangle perimeter: " << r.perimeter() << endl;

    // Triangle example
    Triangle t(3, 4, 5);
    cout << "Triangle area: " << t.area() << endl;
    cout << "Triangle perimeter: " << t.perimeter() << endl;

    // Square example
    Square s(7);
    cout << "Square area: " << s.area() << endl;
    cout << "Square perimeter: " << s.perimeter() << endl;

    return 0;
}
```

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

a. Function overloading and operator overloading

1. Function overloading

```
#include <iostream>
Using namespace std;

class Overload {
```

```
public:
    // Function Overloading
    int add(int a, int b) {
        return a + b;
    }

    double add(double a, double b) {
        return a + b;
    }
};

int main() {
    Overload obj;

    cout << obj.add(3, 4) << endl;       // Calls int add(int a, int b)
    cout << obj.add(3.5, 4.5) << endl;   // Calls double add(double a, double b)

    return 0;
}
```

2. **Operator overloading:**

```
#include <iostream>
Using namespace std;

class OperatorOverload {
public:
    int value;

    // Operator Overloading
    OperatorOverload        operator+(const OperatorOverload& other) {
        OperatorOverload result;
        result.value = this->value + other.value;
        return result;
    }
};

int main() {
    OperatorOverload obj1, obj2;
    obj1.value = 3;
    obj2.value = 4;

    OperatorOverload  result  =  obj1  +  obj2;  //  Calls  operator+(const
OperatorOverload& other)
```

```
    cout << result.value << endl;

    return 0;
}
```

b. Pass by value and pass by reference

1. Pass by value

```
#include <iostream>
Using namespace std;

void incrementByValue(int x) {
    x++;
}

int main() {
    int value = 5;
    incrementByValue(value);

cout << "Value after pass by value: " << value << endl;  // Output: 5

    return 0;
}
```

2. Pass by reference

```
#include <iostream>
Using namespace std;

void incrementByReference(int &x) {
    x++;
}

int main() {
    int value = 5;
    incrementByReference(value);

    cout << "Value after pass by reference: " << value << endl;  // Output: 6

    return 0;
}
```

c. Parameters and arguments

```
#include <iostream>
```

```
// Parameters
void printSum(int a, int b) {
std::cout << "Sum: " << a + b << std::endl;
}

int main() {
int num1 = 3, num2 = 4;

// Arguments
printSum(num1, num2);  // Here, num1 and num2 are arguments passed
to the function.
return 0;
}
```

a and b in the function printSum are parameters. When the function is called with num1 and num2, these values are referred to as arguments. Parameters are the variables in the function definition, while arguments are the values passed to the function when it is called.

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:

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

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.

```
#include <iostream>

using namespace std;

class CalculateG {

private:

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

    double fallingTime = 30;

    double initialVelocity = 0.0;
    double finalVelocity = 0.0;

    double initialPosition = 0.0;

    double finalPosition = 0.0;

public:

    void calculate() {

        // Add the formulas for position and velocity

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

        finalVelocity = gravity * fallingTime + initialVelocity;

        cout << "The object's position after " << fallingTime << " seconds is " << finalPosition
<< " m." << endl;

        cout << "The object's velocity after " << fallingTime << " seconds is " << finalVelocity
<< " m/s." << endl;

    }

    double multiply(double a, double b) {

        return a * b;

    }

    double power(double a) {

        return a * a;

    }

    double sum(double a, double b) {

        return a + b;

    }

    void outline(string text) {
        cout << text << endl;

    }

};

int main() {

    CalculateG obj;

    obj.calculate();

    obj.outline("Multiplication: " + to_string(obj.multiply(2, 3)));

    obj.outline("Power: " + to_string(obj.power(4)));

    obj.outline("Sum: " + to_string(obj.sum(2, 3)));

    return 0;

}
```

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, ...

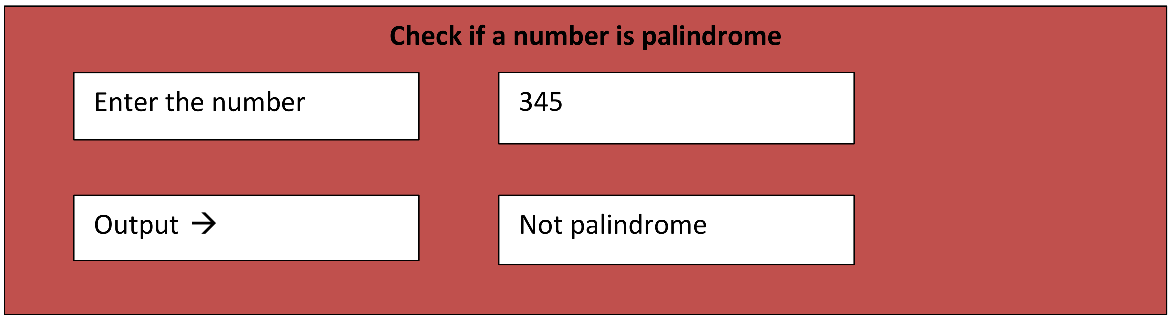
By considering the terms in the Fibonacci sequence whose values do not exceed four million, write a C++ method to find the sum of all the even- valued terms.

```
#include <iostream>

using namespace std;

int main() {

    int sum = 0;

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

    while (b <= 4000000) {
        if (b % 2 == 0) {
            sum += b;
        }
        c = a + b;
        a = b;
        b = c;
    }
    cout << "The sum of all even-valued terms in the Fibonacci sequence whose values do not exceed four million is " << sum << "." << 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:



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>

using namespace std;

int main() {
    const int size = 15;
    int arr[size];

    // Input values into the array
    cout << "Enter 15 integer values:" << endl;
    for (int i = 0; i < size; ++i) {
        cout << "Enter value " << (i + 1) << ": ";
        cin >> arr[i];
    }

    // a) Print values stored in the array
    cout << "\nValues stored in the array:" << endl;
    for (int i = 0; i < size; ++i) {
        cout << arr[i] << " ";
    }

    // b) Check if a number is present in the array
    int searchNumber;
    cout << "\nEnter a number to search in the array: ";
    cin >> searchNumber;

    bool numberFound = false;
    int index;

    for (int i = 0; i < size; ++i) {
        if (arr[i] == searchNumber) {
            numberFound = true;
            index = i;
            break;
        }
    }

    if (numberFound) {
        cout << "The number found at index " << index << endl;
    } else {
        cout << "Number not found in this array" << endl;
    }

    // c) Create another array in reverse order
    int reverseArr[size];
    for (int i = 0; i < size; ++i) {
        reverseArr[i] = arr[size - 1 - i];
    }

    // Print elements of the new array in reverse order
    cout << "\nElements of the new array (in reverse order):" << endl;
    for (int i = 0; i < size; ++i) {
        cout << reverseArr[i] << " ";
    }

    // d) Calculate sum and product of elements in the array
    int sum = 0;
    long long product = 1;

    for (int i = 0; i < size; ++i) {
        sum += arr[i];
        product *= arr[i];
    }

    // Print sum and product
    cout << "\n\nSum of all elements: " << sum << endl;
    cout << "Product of all elements: " << product << endl;

    return 0;
}
```