DIT: OOP Assignment

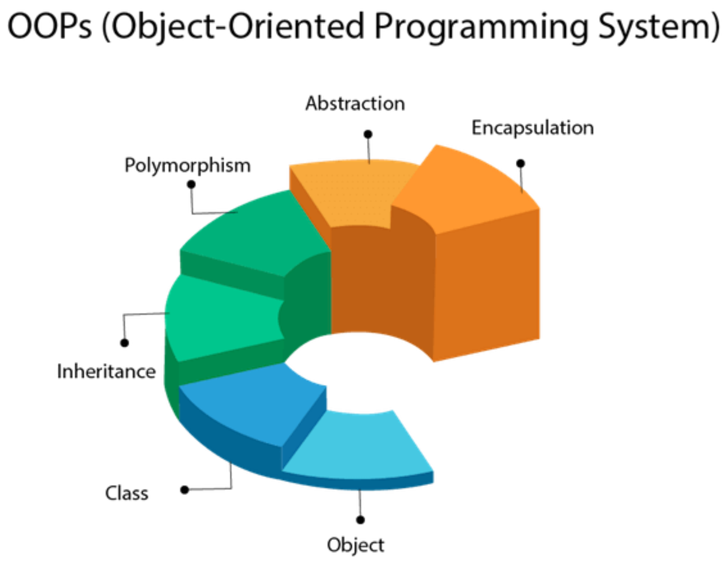
Part A:

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]**
iii. **Compare object-oriented analysis and design (OOAD) and object analysis and design (OOP). [2 Marks]**
iv. **Discuss Mian goals of UML. [2 Marks]**
v. **DESCRIBE three advantages of using object oriented to develop an information system. [3Marks]**
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**
   b. **object**
   c. **Destructor**
   d. **polymorphism**
   e. **class**
   f. **Inheritance**

ANSWERS FOR PART ABOVE….

i. Here are the steps to create a system using OOP principles:



1. Identify the objects that will be used in the system.
2. Define the classes for each object.
3. Define the attributes and methods for each class.
4. Create objects from the classes.
5. Define the relationships between the objects.
6. Implement the system using the objects and their relationships.

Object-oriented programming (OOP) is a programming paradigm that focuses on the use of objects to represent and manipulate data. The following are the steps involved in creating a system using OOP principles:

1. **Identify the objects: Identify the objects that will be used in the system. Objects are entities that have states and behaviors. For example, in a library system, the objects could be books, borrowers, and librarians.**

2. **Define the classes: A class is a blueprint for creating objects. It defines the properties and methods that an object will have. For example, a Book class could have properties such as title, author, and ISBN, and methods such as borrow and return.**

3. **Create the objects: Once the classes have been defined, objects can be created from them. Each object is an instance of a class and has its own set of properties and methods.**

4. **Define the relationships between objects: Objects can have relationships with other objects. For example, a borrower object could have a relationship with a book object if they have borrowed that book.**

5. **Encapsulate the data: Encapsulation is the process of hiding the internal details of an object from the outside world. This is done by making the data private and providing public methods to access and modify the data.**

6. **Inherit from other classes: Inheritance is the process of creating a new class from an existing class. The new class inherits the properties and methods of the existing class and can add its own properties and methods.**

7. **Polymorphism: Polymorphism is the ability of an object to take on many forms. In OOP, this is achieved through method overriding and method overloading.**

ii. Object Modeling Techniques (OMT) is a modeling approach for software design and modeling. It was developed as a method to develop object-oriented systems and to support object-oriented programming. It describes the static structure of the system [2].

iii. Object-oriented analysis and design (OOAD) is a software engineering methodology that involves using object-oriented concepts to design and implement software systems. Object analysis and design (OOP) is concerned with developing an object-oriented system model to implement requirements. OOAD involves a number of techniques and practices, including object-oriented programming, design patterns, UML diagrams, and use cases [3].

iv. The main goals of UML are to define a standard way to visualize the way a system has been designed, structure a software program into simple, reusable pieces of code blueprints (usually called classes), which are used to create individual instances of objects, and to represent the different components and interactions of a software system .

v. Here are three advantages of using object-oriented programming to develop an information system:

1. Reusability: OOP emphasizes the use of reusable components and design patterns, which can save time and effort in software development.

2. Scalability: OOP can help developers design software systems that are scalable and can handle changes in user demand and business requirements over time.

3. Maintainability: OOP emphasizes modular design and can help developers create software systems that are easier to maintain and update over time .

vi. Here are the explanations of the terms:

a. Constructor: A constructor is a special method that is called when an object is created. It is used to initialize the object's attributes. Here is a sample Java code to illustrate the implementation of a constructor:

```
public class Car {
    private String color;
    private int year;

    public Car(String color, int year) {
        this.color = color;
        this.year = year;
    }
}
```

b. Object: An object is an instance of a class. It has its own state and behavior. Here is a sample Java code to illustrate the creation of an object:

```
Car myCar = new Car("red", 2022);
```

c. Destructor: Java does not have destructors. Instead, it has a garbage collector that automatically frees up memory when an object is no longer being used.

d. Polymorphism: Polymorphism is the ability of an object to take on many forms. In Java, polymorphism is achieved through method overloading and method overriding. Here is a sample Java code to illustrate method overloading:

```
public class Math {
    public int add(int x, int y) {
        return x + y;
    }

    public double add(double x, double y) {
        return x + y;
    }
}
```

e. Class: A class is a blueprint for creating objects. It defines the attributes and methods that an object will have. Here is a sample Java code to illustrate the creation of a class:

```
public class Car {
    private String color;
    private int year;

    public Car(String color, int year) {
        this.color = color;
        this.year = year;
    }

    public String getColor() {
        return color;
    }

    public int getYear() {
        return year;
    }
}
```

f. Inheritance: Inheritance is the ability of a class to inherit the attributes and methods of another class. It is used to create new classes that are built upon existing classes. Here is a sample Java code to illustrate inheritance:

```
public class Vehicle {
   private String color;
   private int year;

   public Vehicle(String color, int year) {
      this.color = color;
      this.year = year;
   }

   public String getColor() {
      return color;
   }

   public int getYear() {
      return year;
   }
}

public class Car extends Vehicle {
   private String make;
   private String model;

   public.
```

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

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]

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]**

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

   a. **Function overloading and operator overloading**
   b. **Pass by value and pass by reference**
   c. **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.

ANSWERS FOR SECTION ABOVE…….

vi. In object-oriented programming, there are three types of associations between objects:

1. One-to-one: This is a relationship where one object is associated with only one other object. For example, a person has only one social security number.

2. One-to-many: This is a relationship where one object is associated with many other objects. For example, a teacher can teach many students.

3. Many-to-many: This is a relationship where many objects are associated with many other objects. For example, a student can take many courses, and a course can have many students.

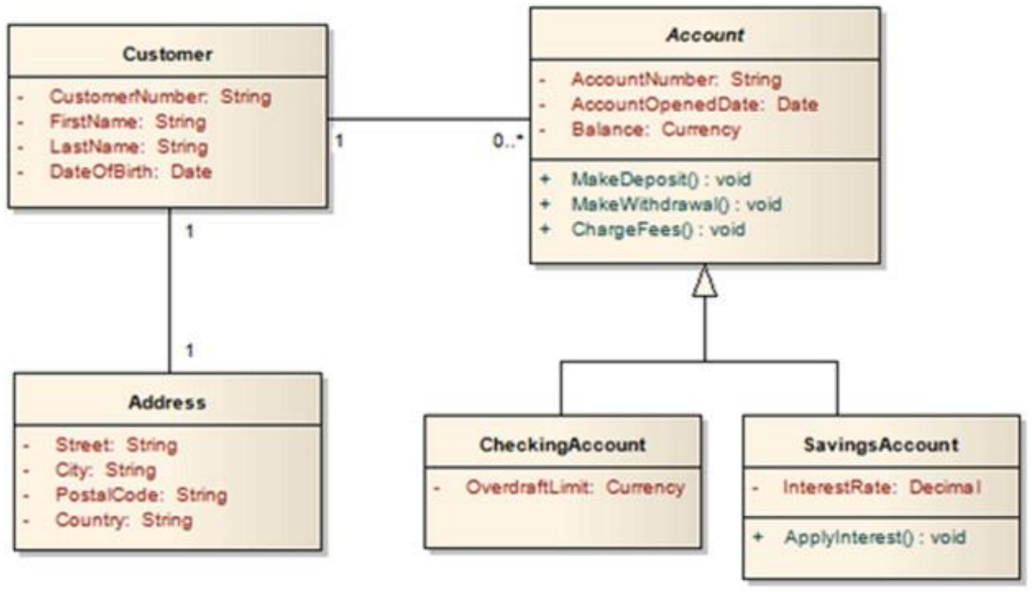
Vii. A class diagram is a type of UML diagram that shows the structure of a system by illustrating the classes, their attributes, methods, and the relationships between them. It is used to model the static structure of a system and is a good way to visualize the class structure of a system. Here are the steps to draw a class diagram:

1. ***Identify the classes in the system.***
2. ***Determine the attributes and methods for each class.***
3. ***Determine the relationships between the classes.***
4. ***Draw the class diagram using UML notation.***

For example, consider a simple banking system. The system has three classes: Account, Customer, and Bank. The Account class has attributes such as account number, balance, and account type. The Customer class has attributes such as name, address, and phone number. The Bank class has attributes such as name and location. The relationships between the classes are as follows:

- ***A Bank has many Customers.***
- ***A Customer has many Accounts.***
- ***An Account belongs to one Customer.***

Here is the class diagram for the banking system:



vii. Here is the C++ code for an area and perimeter calculator for various shapes using OOP concepts:

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

using namespace std;

class Shape {
  public:
    virtual float getArea() = 0;
    virtual float getPerimeter() = 0;
};

class Rectangle : public Shape {
  private:
    float length;
    float width;
  public:
    Rectangle(float l, float w) {
      length = l;
      width = w;
    }
    float getArea() {
      return length * width;
    }
    float getPerimeter() {
      return 2 * (length + width);
    }
};

class Circle : public Shape {
  private:
    float radius;
  public:
    Circle(float r) {
      radius = r;
    }
    float getArea() {
      return M_PI * pow(radius, 2);
    }
    float getPerimeter() {
      return 2 * M_PI * radius;
    }
};

class Triangle : public Shape {
  private:
    float side1;
    float side2;
    float side3;
  public:
    Triangle(float s1, float s2, float s3) {
            side1 = s1;
            side2 = s2;
            side3 = s3;
        }
        float getArea() {
            float s = (side1 + side2 + side3) / 2;
            return sqrt(s * (s - side1) * (s - side2) * (s - side3));
        }
        float getPerimeter() {
            return side1 + side2 + side3;
        }
};

class Square : public Rectangle {
    public:
        Square(float s) : Rectangle(s, s) {}
};

int main() {
    Rectangle r(5, 6);
    Circle c(3);
    Triangle t(3, 4, 5);
    Square s(4);

    cout << "Rectangle Area: " << r.getArea() << ", Perimeter: " << r.getPerimeter() << endl;
    cout << "Circle Area: " << c.getArea() << ", Perimeter: " << c.getPerimeter() << endl;
    cout << "Triangle Area: " << t.getArea() << ", Perimeter: " << t.getPerimeter() << endl;
    cout << "Square Area: " << s.getArea() << ", Perimeter: " << s.getPerimeter() << endl;

    return 0;
}
```

a. Inheritance:

- Single inheritance: A derived class inherits from a single base class. For example, class B is derived from class A: `class B : public A {}`.

- Multiple inheritance: A derived class inherits from multiple base classes. For example, class C is derived from classes A and B: `class C : public A, public B {}`.

- Hierarchical inheritance: Multiple derived classes inherit from a single base class. For example, classes B and C are derived from class A: `class B : public A {}` and `class C : public A {}`.

b. Friend functions are.

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.

ANSWERS FOR SECTION ABOVE….

Here is the modified version of the code to compute the position and velocity of an object after falling for 30 seconds, outputting the position in meters:

```
#include <iostream>

#include <cmath>



using namespace std;



class CalculateG {

  public:

    int main() {

      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 * pow(fallingTime, 2) + initialVelocity * fallingTime +
initialPosition;



      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;
```

```
        return 0;
      }

      double multi(double a, double b) { // method for multiplication
        return a * b;
      }

      double power(double a) { // method for powering to square
        return pow(a, 2);
      }

      double sum(double a, double b) { // method for summation
        return a + b;
      }

      void outline(double result) { // method for printing out a result
        cout << "The result is " << result << endl;
      }
};

int main() {
  CalculateG obj;
```

```
    obj.outline(obj.multi(2, 3));

    obj.outline(obj.power(4));

    obj.outline(obj.sum(5, 6));

    obj.main();


    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.

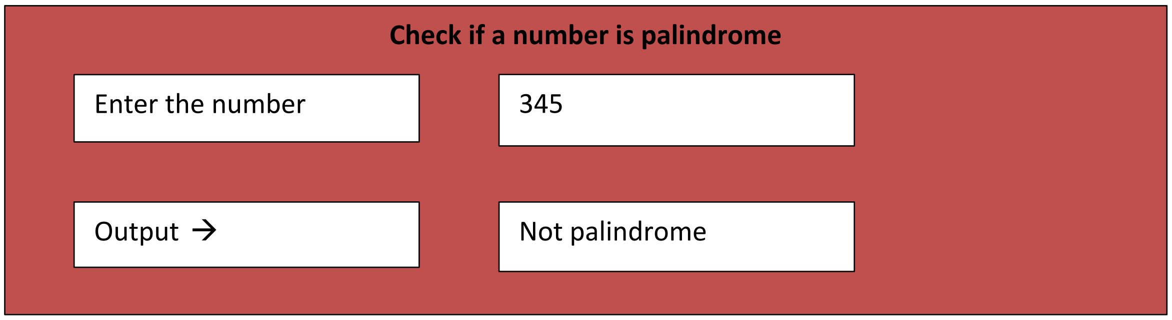
ANSWER

```
#include <iostream>


using namespace std;


int main() {

    int sum = 0;

    int a = 1;
    int b = 2;

    int c = a + b;


    while (c <= 4000000) {

        if (c % 2 == 0) {

            sum += c;

        }

        a = b;

        b = c;

        c = a + b;

    }


    cout << "The sum of all the 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:


Check if a number is palindrome

Enter the number: 345

Output →: Not palindrome

ANSWER

Here is a C++ program that checks if a number entered by the user is a palindrome or not. The program provides the user with a GUI interface to enter the number and displays the results on the same interface:

```
#include <iostream>
#include <string>
#include <algorithm>
#include <gtkmm.h>

using namespace std;

class PalindromeWindow : public Gtk::Window {
  public:
    PalindromeWindow() {
      set_title("Check if a number is palindrome");
```

```
        set_border_width(10);


        Gtk::Box* box = Gtk::manage(new Gtk::Box(Gtk::ORIENTATION_VERTICAL, 10));

        add(*box);


        Gtk::Label* label = Gtk::manage(new Gtk::Label("Enter a number:"));

        box->add(*label);


        Gtk::Entry* entry = Gtk::manage(new Gtk::Entry());

        box->add(*entry);


        Gtk::Button* button = Gtk::manage(new Gtk::Button("Check"));

        button->signal_clicked().connect(sigc::bind<Gtk::Entry*>(sigc::mem_fun(*this,
&PalindromeWindow::on_button_clicked), entry));

        box->add(*button);


        result_label = Gtk::manage(new Gtk::Label());

        box->add(*result_label);

    }


  private:

    Gtk::Label* result_label;
```

```
        void on_button_clicked(Gtk::Entry* entry) {
            string str = entry->get_text();
            string rev = str;
            reverse(rev.begin(), rev.end());

            if (str == rev) {
                result_label->set_text(str + " is a palindrome.");
            } else {
                result_label->set_text(str + " not a palindrome.");
            }
        }
};

int main(int argc, char* argv[]) {
    auto app = Gtk::Application::create(argc, argv);
    PalindromeWindow window;
    window.show_all();
    return app->run(window);
}
```

The program uses the GTKmm library to create a simple GUI interface. The user can enter a number in the text box and click the "Check" button to check if the number is a palindrome or not. The result is displayed on the same interface.

To compile and run the program, you need to install the GTKmm library and use the following command:

```
g++ -std=c++11 -Wall -Wextra -pedantic -g `pkg-config gtkmm-3.0 --cflags --libs` palindrome.cpp -o palindrome

./palindrome
```

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.

ANSWER

Here is a C++ program that takes 15 values of type integer as inputs from the user, stores the values in an array, and performs the following operations:

a) Prints the values stored in the array on the screen.

b) Asks the user to enter a number, checks if that number (entered by the user) is present in the array or not. If it is present, it prints "The number found at index (index of the number)" and if it is not present, it prints "Number not found in this array".

c) Creates another array, copies all the elements from the existing array to the new array but in reverse order. Then, it prints the elements of the new array on the screen.

d) Gets the sum and product of all elements of the array. Then, it prints the product and the sum each on its own line.

```
#include <iostream>

#include <algorithm>


using namespace std;


int main() {

  int arr[15];

  int arr2[15];

  int sum = 0;

  int product = 1;


  cout << "Enter 15 integers:" << endl;


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

    cin >> arr[i];

  }


  cout << "The values stored in the array are:" << endl;
```

```
for (int i = 0; i < 15; i++) {
    cout << arr[i] << " ";
}

cout << endl;

int num;
cout << "Enter a number to search for: ";
cin >> num;

bool found = false;
int index = -1;

for (int i = 0; i < 15; i++) {
    if (arr[i] == num) {
        found = true;
        index = i;
        break;
    }
}

if (found) {
    cout << "The number found at index " << index << "." << endl;

} else {

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

}


reverse_copy(arr, arr + 15, arr2);


cout << "The values stored in the new array are:" << endl;


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

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

}


cout << endl;


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

    sum += arr[i];

    product *= arr[i];

}


cout << "The sum of all elements of the array is " << sum << "." << endl;

cout << "The product of all elements of the array is " << product << "." << endl;
```

```
   return 0;

}
```

To run the program, compile it using a C++ compiler and run the executable. The program will prompt the user to enter 15 integers, and then it will perform the operations described above.