OOP in Java

```
// The 'Hello' class is the main class where the Java program begins.
class Hello {
// The 'main' method is the entry point of the program, where execution starts.
public static void main(String[] args) {
// This line prints the message "Hello Class" to the console.
System.out.println("Hello Class");
}
}
```

```
##include <stdio.h>
float calculateArea(float length, float width)
{
return length * width;
}
int main() {
float length = 5.0;
float width = 3.0;
float area = calculateArea(length, width);
printf("Area of the rectangle: %f\n", area);
return 0;
}
```

- **Procedure-Oriented Programming (POP)**
  - POP is also known as structured programming.
  - It breaks down a program into functions or procedures.
  - Procedures manipulate data often stored in global variables.
  - Data and functions are separate in POP.
  - Examples: C, Pascal.

```
class Rectangle {
private float length;
private float width;
public Rectangle(float length, float width) {
this.length = length;
this.width = width;
}
public float calculateArea() {
return length * width;
}
public static void main(String[] args) {
Rectangle rectangle = new Rectangle(5.0f, 3.0f);
float area = rectangle.calculateArea();
System.out.println("Area of the rectangle: " + area);
}
}
```

- **Object-Oriented Programming (OOP)**
  - OOP revolves around objects, instances of classes.
  - Data and functions are encapsulated within objects.
  - Objects communicate through well-defined interfaces.
  - Key concepts: Inheritance, Encapsulation, Polymorphism.
  - Examples: Java, C++, Python, Ruby.

Procedure Oriented versus Object Oriented Programming

- Procedure Oriented Programming is a programming paradigm that relies on procedures or routines.
- In POP, the program is divided into small, manageable parts called procedures or functions.
- These procedures can share data through global variables, and the focus is on procedures that perform operations on data.

- Object-Oriented Programming is a paradigm that revolves around objects, which are instances of classes.
- A class is a blueprint that defines the properties (attributes) and behaviors (methods) of objects.
- OOP promotes the concept of encapsulation, inheritance, and polymorphism.

Procedure Oriented versus Object Oriented Programming

```
#include <stdio.h>
// Function to calculate the sum of two numbers
int add(int a, int b) {
    return a + b;
}
// Function to calculate the difference of two numbers
int subtract(int a, int b) {
    return a - b;
}
int main() {
    int num1 = 10, num2 = 5;
    // Calling functions
    int sum_result = add(num1, num2);
    int diff_result = subtract(num1, num2);

    // Displaying results
    printf("Sum: %d\n", sum_result);
    printf("Difference: %d\n", diff_result);
    return 0;
}
```

```
// Define a class named Calculator
class Calculator {
    // Attributes
    private int result;
    // Methods
    public void add(int a, int b) {
        result = a + b;
    }
    public void subtract(int a, int b) {
        result = a - b;
    }
    public int getResult() {
        return result;
    }
}
public class Main {
    public static void main(String[] args) {
        // Creating an instance of the Calculator class
        Calculator calculator = new Calculator();
        // Calling methods on the object
        calculator.add(10, 5);
        System.out.println("Sum: " + calculator.getResult());
        calculator.subtract(10, 5);
        System.out.println("Difference: " + calculator.getResult());
    }
}
```

OOP principles

- Object-Oriented Programming (OOP) is built on four main principles:
  - encapsulation,
  - inheritance,
  - polymorphism, and
  - abstraction.
- These principles provide a way to structure and design code in a modular and efficient manner

OOP principles

- **Encapsulation**:
- Definition:
  - Encapsulation is the bundling of data (attributes) and the methods (functions) that operate on the data into a single unit, known as a class.
- Purpose:
  - It helps in hiding the internal details of an object and restricts access to its inner workings.
  - Users interact with the object through an interface provided by the class.

```
class Car {
    // Encapsulated data
    private String model;
    // Encapsulated method
    public void setModel(String newModel) {
        model = newModel;
    }
    public String getModel() {
        return model;
    }
}
public class Main {
    public static void main(String[] args) {
        // Using encapsulation
        Car myCar = new Car();
        myCar.setModel("Toyota");
        System.out.println("Car Model: " + myCar.getModel(
    }
}
```

OOP principles

- **Inheritance**:
- Definition:
  - Inheritance is a mechanism that allows a new class (subclass or derived class) to inherit properties and behaviors of an existing class (superclass or base class).
- Purpose:
  - It promotes code reuse and establishes a relationship between classes, where the subclass can reuse the features of the superclass and can also extend or override them.

```
class Animal {
    public void eat() {
        System.out.println("Animal is eating");
    }
}
class Dog extends Animal {
    public void bark() {
        System.out.println("Dog is barking");
    }
}
public class Main {
    public static void main(String[] args) {
        // Using inheritance
        Dog myDog = new Dog();
        myDog.eat();  // Inherited from Animal
        myDog.bark(); // Specific to Dog
    }
}
```

OOP principles

- **Polymorphism**:
- Definition:
  - Polymorphism allows objects of different types to be treated as objects of a common type.
  - It comes in two forms: compile-time (method overloading) and runtime (method overriding).
- Purpose:
  - It enables flexibility in programming by allowing objects to be used interchangeably, enhancing code readability, and supporting dynamic behavior.

```
class Shape {
    public void draw() {
        System.out.println("Drawing a shape");
    }
}
class Circle extends Shape {
    @Override
    public void draw() {
        System.out.println("Drawing a circle");
    }
}
class Square extends Shape {
    @Override
    public void draw() {
        System.out.println("Drawing a square");
    }
}
public class Main {
    public static void main(String[] args) {
        // Using polymorphism
        Shape myShape = new Circle();
        myShape.draw(); // Calls draw method in Circle
        myShape = new Square();
        myShape.draw(); // Calls draw method in Square
    }
}
```

OOP principles

- **Abstraction**:
- Definition:
  - Abstraction involves simplifying complex systems by modeling classes based on the essential properties and behaviors relevant to the problem domain, while ignoring irrelevant details.
- Purpose:
  - Abstraction allows programmers to focus on high-level concepts and ignore low-level details, making code more understandable and maintainable.

```
abstract class Shape {
    // Abstract method (no implementation)
    public abstract void draw();
}
class Circle extends Shape {
    @Override
    public void draw() {
        System.out.println("Drawing a circle");
    }
}
class Square extends Shape {
    @Override
    public void draw() {
        System.out.println("Drawing a square");
    }
}
public class Main {
    public static void main(String[] args) {
        // Using abstraction
        Shape myShape = new Circle();
        myShape.draw(); // Calls draw method in Circle

        myShape = new Square();
        myShape.draw(); // Calls draw method in Square
    }
}
```

Advantages and Disadvantages of OOP

- **Advantages**:
- Modularity: Encapsulates objects for easier understanding and maintenance.
- Reusability: Classes and objects can be reused in different programs.
- Flexibility: Allows easy addition of new classes without affecting existing code.
- Easier Maintenance: Changes to one part of the code don't impact others.
- Improved Productivity: Higher-level abstraction leads to increased productivity.
- Real-world Modeling: Maps software solutions to real-world problems effectively.
- Encapsulation: Hides internal details, improving security and reducing complexity.
- Inheritance: Promotes code reuse and establishes relationships between classes.
- Polymorphism: Enables flexibility and simplifies code.

- **Disadvantages**:
- Learning Curve: Concepts can be challenging, especially for beginners.
- Performance Overhead: Runtime dynamic dispatch may introduce performance overhead.
- Complexity: Can lead to complex and verbose code if not properly designed.
- Not Always Suitable: May not be the best paradigm for all types of applications.
- Overhead of Abstraction: Excessive abstraction can be cumbersome for developers.
- Overemphasis on Design: Focus on design may lead to delays in implementation.
- Potential for Misuse: Flexibility can result in misuse if principles are not followed correctly.

Assignments

- Define Object-Oriented Programming (OOP) and explain its core principles.
- Provide a simple example of how OOP principles are implemented in java programming language.
- Contrast Procedure Oriented Programming (POP) with Object-Oriented Programming (OOP). Highlight at least two key differences.
- In what scenarios would you choose OOP over POP for software development, and why?
- List three advantages and three disadvantages of Object-Oriented Programming.
- Discuss how the advantages of OOP, such as modularity and reusability, can contribute to the maintainability of large software projects.