Introduction to Object Oriented Programming

ITT Computer Science
4th Year
First Trimester

Prerequisites

- Fundamental programming structures (for, if) and types (int,float,string,boolean)
- Boole’s Algebra
- Ability to translate a generic problem into flow charts
- Ability to translate flowchart to code (C or Python)

Knowledge

- Difference between the most important programming paradigms
- Fundamental properties of OOP (Polymorphism, Inheritance…)
- Overriding and Overloading
- Difference between class and object
- Which scenarios are ideal to be modeled as classes
- Divide et impera principle

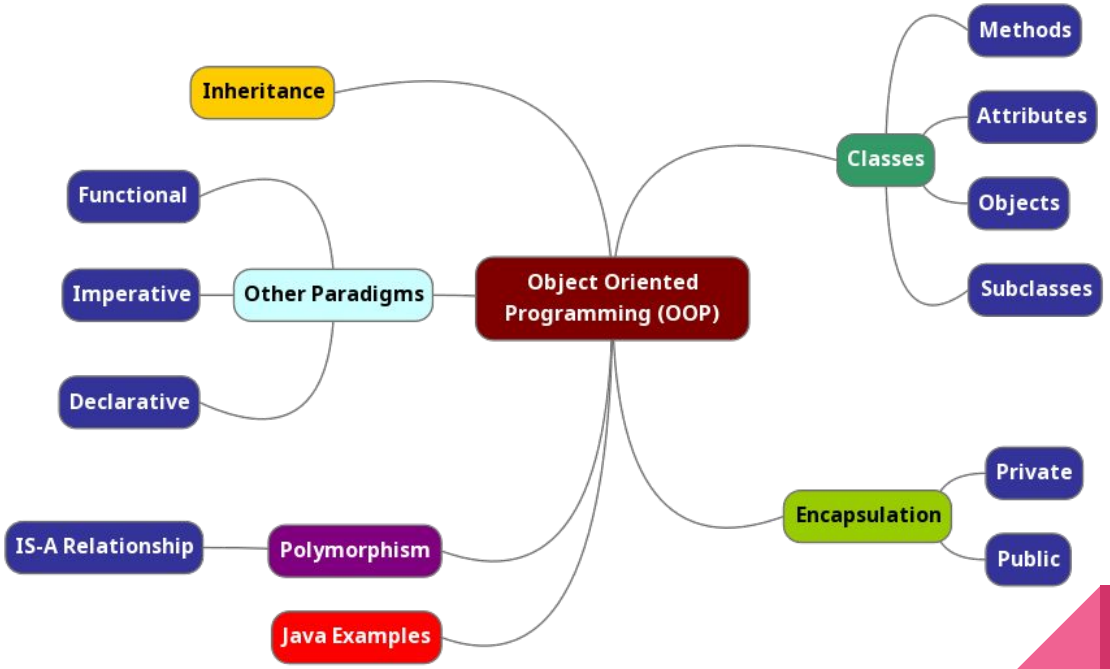
Skills

- Define interconnection between classes
- Ability to make code reusable
- Ability to write elegant code

Competences

- Understand which paradigm suits better for a certain task
- Being able to divide a problem in models (classes)
- Being able to design classes efficiently
- Choosing the appropriate OOP features to address specific challenges

Mind Map


Inheritance
Functional
Imperative
Other Paradigms
Declarative
IS-A Relationship
Polymorphism
Java Examples
Object Oriented Programming (OOP)
Classes
Methods
Attributes
Objects
Subclasses
Encapsulation
Private
Public

OOP Question Time!

Video Games

Many video games were made using OOP!

Introduction

What are programming paradigms?

They are **approaches** to programming that **dictate how** developers **structure** their code **to solve** problems

For example there are: **functional** programming (Scala, Java Steams), **imperative** (C,C++,Java), **declarative** (SQL), **object oriented** (Java, C++) and many more

Some languages are **multi-paradigm**!

Why OOP?

Sometimes modeling a problem with just functions or native types (int, float, bool) it’s not practical

Giving a more complex structure with a meaning can help

Trying to aggregate types and functions for a bigger **entity**

Basic Terminology

Field/Attribute: A characteristic of the class that represents the data or attributes associated with the class, usually it's a simple type like *int*, but it can be more complex (like a *custom object* itself)

Method: It's a function of a class that describes its behaviour or actions

Class: It's the "abstract" definition of a complex type with its own fields and methods

Object: An instance of a class, representing a specific, tangible occurrence of the abstract class

Class vs Object

Class: Car

Fields:

- String model;
- Color color;
- int passengers;
- double fuel;

Methods:

- Add/Remove passenger
- Fill tank

Object: a car

Fields:

- String model= ‘X’;
- Color color = Color.Black;
- int passengers = 1;
- double fuel = 50;

Methods: same as the class

How to create an Object

In order to create an object we often use the keyword ***new***

It relies on the use of a special method, called **Constructor**

A constructor defines how the fields of the object will be initialized (initial values assigned to the fields)

Some languages even have a **Destructor** (C++) and it allows to deallocate the memory used for the object and its fields

How to create an Object

```java
public class Car {
    // Fields (attributes)
    private String brand;
    private String model;
    private int year;

    // Constructor
    public Car(String brand, String model, int year) {
        this.brand = brand;
        this.model = model;
        this.year = year;
    }

    // Method to display information about the car
    public void displayInfo() {
        System.out.println("Brand: " + brand);
        System.out.println("Model: " + model);
        System.out.println("Year: " + year);
    }

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

        // Accessing the displayInfo method to print car information
        myCar.displayInfo();
    }
}
```

Encapsulation

Restricts **external access** to some of its components and **hides the internal implementation details** from the outside world

It allows data protection by using access modifiers like private or protected

The access modifiers can be applied to:

- **Classes**
- **Fields**
- **Methods**

Encapsulation

The class can be referenced by anyone

The fields can be accessed by the class itself (any method inside it can access to them)

```java
public class Car {
    // Fields (attributes)
    private String brand;
    private String model;
    private int year;

    // Constructor
    public Car(String brand, String model, int year) {
        this.brand = brand;
        this.model = model;
        this.year = year;
    }
```

The constructor can be accessed by anyone. How could we create an object of this class otherwise?

Overloading and Overriding

Two techniques that apply to **methods**

Overriding is **re-defining the behaviour** of a specific method

Overloading is creating a method with the **same name but with different input** parameters

Inheritance

A class can use the keyword ***extends*** in order to define that it's a subclass of another class (e.g. X **extends** Y)

Subclasses automatically **gain the attributes and methods** of the superclass

A subclass can add additional attributes and methods

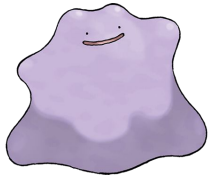
Inheritance

```java
class Bicycle {
    int gear, speed;

    Bicycle(int gear, int speed) {
        this.gear = gear;
        this.speed = speed;
    }

    void applyBrake(int decrement) {
        speed -= decrement;
    }

    void speedUp(int increment) {
        speed += increment;
    }
}

class MountainBike extends Bicycle {
    int seatHeight;

    MountainBike(int gear, int speed, int startHeight) {
        super(gear, speed);
        seatHeight = startHeight;
    }
}

public class Test {
    public static void main(String args[]) {
        MountainBike mb = new MountainBike(3, 100, 25);
        System.out.println("Gears: " + mb.gear + ", Speed: " + mb.speed + ", Seat Height: " + mb
            .seatHeight);
    }
}
```

Mountain Bike has gear, speed thanks to Bicycle, since it’s a **subclass**.
In addition it has seatHeight too!

Output: Gears: 3, Speed: 100, Seat Height: 25

Polymorphism



Ability of objects of different classes to be **treated as objects of a common base class**

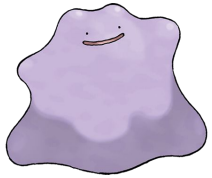
Polymorphism facilitates **code flexibility, extensibility, and the implementation of generic algorithms** that can work with diverse object types

It's a ***IS A*** relationship

Class ***Class*** ***Class***

Pokemon □ NormalTypePokemon □ Ditto ***Ditto is a Pokemon***

Polymorphism



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

class Dog extends Animal {
    @Override
    void makeSound() {
        System.out.println("Woof! Woof!");
    }
}

class Cat extends Animal {
    @Override
    void makeSound() {
        System.out.println("Meow!");
    }
}

public class PolymorphismExample {
    public static void main(String[] args) {
        Animal dog = new Dog(); // Polymorphism: Dog object treated as Animal
        Animal cat = new Cat(); // Polymorphism: Cat object treated as Animal

        dog.makeSound(); // Calls the overridden method in Dog class
        cat.makeSound(); // Calls the overridden method in Cat class
    }
}
```

A dog is an **Animal**!

This will produce “*Woof Woof!*”.

The JVM will call the more specific method because of the override!

It’s time to try Designing!

Evaluation Quiz

5 points

In object-oriented programming, what is the primary distinction between a class and an object? 0,5

- A class is an instance of an object.
- An object is an instance of a class.
- They are interchangeable terms.
- A class and an object have the same meaning.

In C++ and some other languages, a destructor is a special method used for: 0,5

- Creating objects.
- Initializing objects.
- Cleaning up resources before an object is destroyed.
- Accessing private members.

Composition involves creating relationships between classes where one class contains an object of another class. 0,5

- True
- False

Evaluation Quiz

5 points

In Java, the keyword used to indicate inheritance between classes is: 0,5

- inherit
- extends
- implements
- inherits

When a class in Java implements an interface, it must provide implementations for: 0,5

- All methods in the interface.
- At least one method in the interface.
- Only private methods in the interface.
- Static methods in the interface.

Which of the following statements is true? 0,5

- An abstract class can have both abstract and non-abstract methods.
- An interface can have instance variables.
- An abstract class can be instantiated.
- An interface can have constructors.

Evaluation Quiz

5 points

In a class, members (fields and methods) marked as static belong to: 0,5

- Every instance of the class.
- Only the first instance created.
- The class itself, rather than instances of the class.
- Instances of the class, but not the class.

Encapsulation involves bundling data and methods into a single unit, while abstraction involves: 0,5

- Hiding the implementation details and showing only the necessary features.
- Exposing all implementation details for better understanding.
- Ignoring necessary features for simplicity.
- Randomly selecting features for implementation.

In Java, the ***this*** keyword is commonly used within a constructor. What does it refer to? 0,5

- The superclass.
- The current object being instantiated.
- A static method.
- A different class.

Evaluation Quiz

5 points

What is the key difference between method overloading and method overriding?

0,5

- Method overloading is static, and method overriding is dynamic.
- Method overloading involves creating multiple methods with the same name, and method overriding involves providing a specific implementation in a subclass.
- Method overloading is only applicable to static methods, and method overriding is only applicable to instance methods.
- Method overloading and method overriding are synonymous terms.

Be cautious as selecting a wrong answer will result in a penalty of -0.25 points

Evaluation Open Questions

5 points

Question 1

Explain the concept of polymorphism and provide an example in a programming context

2,5

Question 2

Discuss the advantages and disadvantages of using inheritance in object-oriented programming, citing specific examples where it is beneficial and situations where it might be problematic

2,5

A correct and detailed answer will be awarded with 0.5 bonus points (for each correct answer)