Object-Oriented Programming Training Resource

Team SeDriCa, UMIC

November 1, 2024

Contents

1 Introduction to Object-Oriented Programming (OOP)

Object-Oriented Programming (OOP) is a programming paradigm that organizes code into objects and classes, allowing for modularity, reusability, and flexibility in software development. This guide will walk you through the core concepts of OOP and provide exercises to practice each concept.

2 Major Concepts of Object-Oriented Programming (OOP)

Object-Oriented Programming (OOP) is a programming paradigm that uses objects and classes to design and structure software. The main concepts of OOP include:

2.1 Classes and Objects

1. **Class**: A class is a blueprint for creating objects. It defines a set of attributes (data) and methods (functions) that the created objects will have. For example, a class named `Car` might have attributes like `make`, `model`, and `year`, and methods like `start()` and `stop()`.
2. **Object**: An object is an instance of a class. It represents a specific item that has the properties defined by the class. For instance, if `Car` is a class, then `my_car` could be an object of that class with specific values for its attributes (e.g., `make = "Toyota"`).

2.2 Encapsulation

Encapsulation is the concept of restricting access to certain details of an object and exposing only the necessary parts of it. This is achieved by:

1. Making attributes private (accessible only within the class) and providing public methods (getters and setters) to access and modify those attributes.
2. This helps in protecting the object's state and ensuring that it can only be changed in a controlled manner.

2.3 Inheritance

Inheritance allows a new class to inherit the properties and methods of an existing class. The existing class is known as the **parent class** or **base class**, while the new class is referred to as the **child class** or **subclass**. This promotes code reuse and establishes a hierarchical relationship between classes.

1. For example, if `Car` is a parent class, a subclass `ElectricCar` can inherit its properties and methods while also having additional attributes specific to electric cars, such as `battery_capacity`.

2.4 Polymorphism

Polymorphism enables objects of different classes to be treated as objects of a common superclass. It allows for methods to be defined in a parent class and overridden in child classes. This means that the same method name can perform different functions based on the object that invokes it.

1. For instance, both a `Car` and a `Truck` class might implement a method called `drive()`, but the implementation could differ depending on the vehicle type.

2.5 Abstraction

Abstraction is the process of simplifying complex reality by modeling classes based on the essential properties and behaviors of the objects they represent. It allows programmers to focus on the high-level functionalities without worrying about the intricate details.

1. Abstract classes and interfaces are used to define common behaviors that can be implemented by derived classes while hiding the complex implementation details.

3 Assignment: Car Inventory System

This assignment introduces the basics of Object-Oriented Programming (OOP) concepts such as classes, objects, inheritance, and encapsulation. You will create a program that simulates an inventory system for a car dealership, using classes to represent cars and specific types of cars (e.g., electric and sports cars).

3.1 Objective

Implement OOP concepts like encapsulation, inheritance, and polymorphism in a car-themed inventory management system.

3.2 Instructions

1. Complete each exercise below in your preferred programming language (C++ or Python).
2. Make a Git Repository and commit your code for each task. Add a README file with explanations and instructions.

3.3 Problem Statement

Define a class `Car` with the following attributes and methods:

1. Attributes:
   (a) `make`: the make of the car (e.g., Toyota, Ford) (string)
   (b) `model`: the model of the car (e.g., Corolla, Mustang) (string)
   (c) `price`: the price of the car (float)
2. Methods:
   (a) `apply_discount(%discount)`: a method to apply a discount on the price
   (b) `get_info()`: a method to display the car's information

3.4 Tasks

Task 1: Basic Class Creation

1. Create the `Car` class with the attributes and methods listed above.
2. Write a main program that creates a `Car` object, assigns values to its attributes, and displays the information.

Task 2: Encapsulation

1. Modify your `Car` class to make the `price` attribute private.
2. Create getter and setter methods for `price` to control its access.
3. Demonstrate that you can access and modify `price` only through these methods.

Task 3: Inheritance

1. Define two subclasses: `ElectricCar` and `SportsCar`.
2. `ElectricCar` should have an additional attribute called `battery_range` (float, representing the range in miles).
3. `SportsCar` should have an additional attribute called `top_speed` (float, representing the top speed in mph).

4. Override the `get_info()` method in each subclass to display all the car's information, including the additional attributes.

Task 4: Polymorphism

1. Create an array or list of `Car` objects that includes instances of `Car`, `ElectricCar`, and `SportsCar`.
2. Write a loop to call the `get_info()` method on each object, demonstrating polymorphism.

4 Resources

Here are some additional resources to help you learn more about Object-Oriented Programming:

1. Python Documentation: Classes and Objects
2. LearnCPP: Introduction to OOP in C++
3. Real Python: OOP in Python
4. YouTube: Object-Oriented Programming (OOP) in Python by freeCodeCamp.org