Beginner’s Guide to Object-Oriented Programming (OOP) with Mini Projects

Subtitle: Learn OOP concepts in a simple, practical way and apply them to small projects.

Website Name: haas.dev
Website Link: https://dev-roast-app.vercel.app

Introduction

Object-Oriented Programming (OOP) is a **core concept in CS**. It helps structure code efficiently, reuse logic, and build maintainable software. This guide introduces **classes, objects, inheritance, polymorphism, encapsulation, and abstraction** with simple examples.

Step 1: Understanding OOP Basics

- **Class:** Blueprint for creating objects
- **Object:** Instance of a class
- **Attribute/Property:** Variables inside a class
- **Method:** Functions inside a class

Example in Python:

```
class Car:
    def __init__(self, brand, color):
        self.brand = brand
        self.color = color

    def drive(self):
        print(f"{self.brand} is driving!")

# Object
my_car = Car("Toyota", "Red")
my_car.drive()
```

Step 2: Encapsulation

- Hides internal details, exposes only what’s needed
- Use **private variables** or getters/setters

- Use **private variables** or getters/setters

Example:

```
class BankAccount:
    def __init__(self, balance):
        self.__balance = balance  # private

    def deposit(self, amount):
        self.__balance += amount

    def get_balance(self):
        return self.__balance
```

Step 3: Inheritance

- Reuse code from parent class in child class
- Avoid duplication

Example:

```
class Vehicle:
    def __init__(self, type):
        self.type = type

class Car(Vehicle):
    def __init__(self, brand, type):
        super().__init__(type)
        self.brand = brand
```

Step 4: Polymorphism

- Same method name, different behavior
- Example: method overriding

```
class Bird:
    def speak(self):
        print("Chirp!")

class Parrot(Bird):
    def speak(self):
```

```
        print("Squawk!")

bird = Bird()
parrot = Parrot()
bird.speak()   # Chirp!
parrot.speak() # Squawk!
```

Step 5: Abstraction

- Hide complex implementation, show only essential features
- Example using Python abstract class:

```
from abc import ABC, abstractmethod

class Shape(ABC):
    @abstractmethod
    def area(self):
        pass

class Square(Shape):
    def __init__(self, side):
        self.side = side

    def area(self):
        return self.side * self.side
```

Step 6: Mini Project Ideas

1. **Bank Management System** – Use classes for accounts, transactions
2. **Library Management System** – Books, users, borrow/return functionality
3. **Simple To-Do App** – Task class, add/remove tasks

Step 7: Best Practices

- Keep methods small and focused
- Follow **naming conventions** for readability
- Use **inheritance** to avoid code duplication
- Practice with **mini-projects** to strengthen understanding

Step 8: Key Takeaways

- OOP organizes code into reusable, maintainable structures
- Understand and practice **classes, objects, encapsulation, inheritance, polymorphism, abstraction**
- Start applying OOP in mini-projects to prepare for larger applications and final-year projects

---

Visit **haas.dev** for beginner-friendly OOP tutorials, mini-projects, and CS resources.

Website Name: haas.dev
Website Link: https://dev-roast-app.vercel.app
