Jacob Archer
6/28/24

Lab 6

Question 1

The Factory Design Pattern is super useful because it simplifies the process of creating objects, especially when the exact type of object might not be known until runtime. Instead of directly instantiating objects, which can get messy and hard to maintain, a factory handles the creation logic. This keeps your code clean and organized, making it easier to manage and extend. For example, if you need to add a new type of car in a car inventory system, you just update the factory method without touching other parts of your code. This also promotes loose coupling since the code that uses the objects doesn't need to know about the details of how they are created. Overall, the Factory Design Pattern helps in writing more modular, scalable, and maintainable code.

CarFactory.java - School - Visual Studio Code

File Edit Selection View Go Run Terminal Help

EXPLORER

SCHOOL
- CS160
- CS160L
  - Lab1
  - Lab2
    - ArcherJacob_Lab2.pdf
    - Car.class
    - Car.java 1
    - CarAPI.java 9+
    - CarFactory.java 6, U
    - DieselCar.class
    - DieselCar.java 4
    - ElectricCar.class
    - ElectricCar.java 4
    - GasCar.class
    - GasCar.java 4
    - Inventory.class
    - Inventory.java 1
    - lab5.pdf
    - main.class
    - main.java 1, M
    - umlDiagram.png
- CS210
- Phys317

CarFactory.java 6, U | main.java 1, M

CS160L > Lab2 > CarFactory.java > CarFactory

```java
/**
 * Lab 6
 * This class is used to create a CarFactory class that creates instances of the GasCar, ElectricCar, and DieselCar classes and prints out their information.
 * CS160L
 * 6/28/24
 * @author  Jacob Archer
 */

public class CarFactory {
    public static Car createCar(int year, String make, String model) {
        String carType = CarAPI.getCarType(year, make, model);

        switch (carType) {
            case "diesel":
                return new DieselCar(year, Car.API.getCarPrice(year, make, model), make, model, Car.API.getFuelTankCapacity(year, make, model));
            case "gas":
                return new GasCar(year, Car.API.getCarPrice(year, make, model), make, model, Car.API.getFuelTankCapacity(year, make, model));
            case "electric":
                return new ElectricCar(year, Car.API.getCarPrice(year, make, model), make, model, Car.API.getBatteryCapacity(year, make, model));
            default:
                throw new IllegalArgumentException("Unknown car type: " + carType);
        }
    }
}
```

PROBLEMS 40 | OUTPUT | DEBUG CONSOLE | TERMINAL | PORTS | COMMENTS

Tasks

OUTLINE

TIMELINE

master* 37 3 0 Java: Ready

Ln 3, Col 159 Spaces: 4 UTF-8 LF {} Java

EN 7:11 PM