#include <iostream>

#include <string>

// Forward declaration of VehicleFactory

class VehicleFactory;

// The code below is for builder pattern

class VehicleBuilder {

public:

// Function to construct a vehicle

void constructVehicle(VehicleFactory\* factory, const std::string& fuel, const std::string& wheels,

const std::string& seats, int airbags);

};

// The code below is for singleton pattern

class ConfigurationManager {

private:

ConfigurationManager() {}

public:

static ConfigurationManager& getInstance() {

static ConfigurationManager instance;

return instance;

}

};

// Implementing abstract factory pattern

class VehicleFactory {

public:

virtual ~VehicleFactory() {}

// Pure virtual function to create a vehicle of a specific model

virtual void createVehicle(const std::string& model) = 0;

// Pure virtual function to get configurations of the vehicle

virtual std::string getConfigurations() const = 0;

};

// Creating concrete factories for Tata cars

class TataFactory : public VehicleFactory {

public:

// Constructor with parameters for the Tata car's specifications

TataFactory(const std::string& \_model, const std::string& \_fuel, const std::string& \_wheels,

const std::string& \_seats, int \_airbags)

: model(\_model), fuel(\_fuel), wheels(\_wheels), seats(\_seats), airbags(\_airbags) {} // parameters of constructor

void createVehicle(const std::string& /\*model\*/) override {

// Create Tata car with given model

std::cout << "Creating Tata " << model << " car..." << std::endl;

}

// Displaying the configurations of the vehicle chosen

std::string getConfigurations() const override {

return "Model: " + model + "\nFuel: " + fuel + "\nWheels: " + wheels +

"\nSeats: " + seats + "\nAirbags: " + std::to\_string(airbags) + "\n";

}

private:

std::string model;

std::string fuel;

std::string wheels;

std::string seats;

int airbags;

};

// Implementing the constructVehicle function of VehicleBuilder

void VehicleBuilder::constructVehicle(VehicleFactory\* factory, const std::string& fuel, const std::string& wheels,

const std::string& seats, int airbags) {

// Implementation for constructing the vehicle here

if (factory) {

factory->createVehicle("Tata"); // Creating a Tata car

std::cout << "Configurations:\n" << factory->getConfigurations() << std::endl;

}

}

int main() {

// Implementing builder pattern here

VehicleBuilder builder;

ConfigurationManager& configManager = ConfigurationManager::getInstance();

std::string model, fuel, wheels, seats;

int airbags;

// Letting user select the car model

std::cout << "Car models of Tata:\n";

std::cout << "Select among (Altroz/Nexon/Tigor/Safari): ";

std::cin >> model;

std::cout << std::endl;

// Representation : Configurations of the selected model

std::cout << "Enter fuel (EV/Petrol): ";

std::cin >> fuel;

std::cout << "\nEnter wheels (Alloy/BlackRim): ";

std::cin >> wheels;

std::cout << "\nEnter seats (Ventilated/Regular): ";

std::cin >> seats;

std::cout << "\nEnter airbags needed (2/6/8): ";

std::cin >> airbags;

std::cout << std::endl;

// Creating a TataFactory object with specified parameters

VehicleFactory\* factory = new TataFactory(model, fuel, wheels, seats, airbags);

// Constructing the vehicle using builder

builder.constructVehicle(factory, fuel, wheels, seats, airbags);

delete factory; // Deleting the factory object to free memory

return 0;

}