#include <iostream>

#include <string>

#include <unordered\_map>

// Forward declaration of ConfigurationManager

class ConfigurationManager;

// Abstract class for the VehiclePrototype

class VehiclePrototype {

protected:

std::string fuel;

std::string wheels;

std::string seats;

int airbags;

public:

virtual ~VehiclePrototype() {}

// Pure virtual function to display vehicle specifications

virtual void displaySpecifications() = 0;

// Pure virtual function to clone a VehiclePrototype object

virtual VehiclePrototype\* clone() const = 0;

// Getter for Fuel

std::string getFuel() const {

return fuel;

}

// Setter for Fuel

void setFuel(const std::string& \_fuel) {

fuel = \_fuel;

}

// Getter for wheels

std::string getWheels() const {

return wheels;

}

// Setter for wheels

void setWheels(const std::string& \_wheels) {

wheels = \_wheels;

}

// Getter for seats

std::string getSeats() const {

return seats;

}

// Setter for Seats

void setSeats(const std::string& \_seats) {

seats = \_seats;

}

// Getter for airbags

int getAirbags() const{

return airbags;

}

// Setter for airbags

void setAirbags(const int& \_airbags) {

airbags = \_airbags;

}

};

// Concrete prototype class for Altroz

class AltrozPrototype : public VehiclePrototype {

public:

AltrozPrototype() {

fuel = "Petrol";

wheels = "Alloy";

seats = "Ventilated";

airbags = 2;

}

void displaySpecifications() override {

std::cout << "Altroz Specifications:" << std::endl;

std::cout << "Fuel: " << fuel << std::endl;

std::cout << "Wheels: " << wheels << std::endl;

std::cout << "Seats: " << seats << std::endl;

std::cout << "Airbags: " << airbags << std::endl;

}

AltrozPrototype\* clone() const override {

return new AltrozPrototype(\*this);

}

};

// Concrete prototype class for Safari

class SafariPrototype : public VehiclePrototype {

public:

SafariPrototype() {

fuel = "EV";

wheels = "BlackRim";

seats = "Ventilated";

airbags = 8;

}

void displaySpecifications() override {

std::cout << "Safari Specifications:" << std::endl;

std::cout << "Fuel: " << fuel << std::endl;

std::cout << "Wheels: " << wheels << std::endl;

std::cout << "Seats: " << seats << std::endl;

std::cout << "Airbags: " << airbags << std::endl;

}

SafariPrototype\* clone() const override {

return new SafariPrototype(\*this);

}

};

// Concrete prototype class for Nexon

class NexonPrototype : public VehiclePrototype {

public:

NexonPrototype() {

fuel = "Petrol";

wheels = "Alloy";

seats = "Regular";

airbags = 6;

}

void displaySpecifications() override {

std::cout << "Nexon Specifications:" << std::endl;

std::cout << "Fuel: " << fuel << std::endl;

std::cout << "Wheels: " << wheels << std::endl;

std::cout << "Seats: " << seats << std::endl;

std::cout << "Airbags: " << airbags << std::endl;

}

NexonPrototype\* clone() const override {

return new NexonPrototype(\*this);

}

};

// Concrete prototype class for Tigor

class TigorPrototype : public VehiclePrototype {

public:

TigorPrototype() {

fuel = "EV";

wheels = "Alloy";

seats = "Ventilated";

airbags = 6;

}

void displaySpecifications() override {

std::cout << "Tigor Specifications:" << std::endl;

std::cout << "Fuel: " << fuel << std::endl;

std::cout << "Wheels: " << wheels << std::endl;

std::cout << "Seats: " << seats << std::endl;

std::cout << "Airbags: " << airbags << std::endl;

}

TigorPrototype\* clone() const override {

return new TigorPrototype(\*this);

}

};

// Singleton ConfigurationManager class

class ConfigurationManager {

private:

ConfigurationManager() {}

public:

static ConfigurationManager& getInstance() {

static ConfigurationManager instance;

return instance;

}

};

// Director for the Builder Pattern

class VehicleBuilder {

private:

// Map to store prototypes of different Tata vehicles

std::unordered\_map<std::string, VehiclePrototype\*> prototypes;

public:

// Constructor to initialize prototypes

VehicleBuilder() {

prototypes["Altroz"] = new AltrozPrototype();

prototypes["Safari"] = new SafariPrototype();

prototypes["Nexon"] = new NexonPrototype();

prototypes["Tigor"] = new TigorPrototype();

}

// Function to construct a vehicle with specified specifications

void constructVehicle(const std::string& model, const std::string& fuel,

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

VehiclePrototype\* prototype = prototypes[model];

if (prototype != nullptr) {

VehiclePrototype\* vehicle = prototype->clone();

vehicle->setFuel(fuel);

vehicle->setWheels(wheels);

vehicle->setSeats(seats);

vehicle->setAirbags(airbags);

vehicle->displaySpecifications();

delete vehicle;

} else {

throw std::invalid\_argument("Invalid Tata car model: " + model);

}

}

// Destructor to free allocated memory

~VehicleBuilder() {

for (auto& pair : prototypes) {

delete pair.second;

}

prototypes.clear();

}

};

int main() {

VehicleBuilder builder;

ConfigurationManager& configManager = ConfigurationManager::getInstance();

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

int airbags;

// User input for Tata car model

std::cout << "Select Tata car model (Altroz/Nexon/Tigor/Safari):" << std::endl;

std::cin >> model;

// User input for vehicle specifications

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;

// Construct and display the vehicle

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

return 0;

}