Code:

#include <iostream>

#include <string>

using namespace std;

template <typename T>

bool operator==(const T& a, const T& b) {

return a.price == b.price;

}

template <typename T>

bool operator<(const T& a, const T& b) {

return a.price < b.price;

}

// Base class representing a general vehicle

class Vehicle {

private:

string manufacturer, model, color;

int model\_year, stock;

long double engine\_capacity, price;

static int total\_vehicles;

static long double total\_worth\_cars;

public:

// Constructors

Vehicle() { total\_vehicles++; }

Vehicle(string manufacturer, string model, int model\_year,

long double engine\_capacity, long double price,

string color, int stock) :

manufacturer(manufacturer), model(model),

model\_year(model\_year), engine\_capacity(engine\_capacity),

price(price), color(color), stock(stock) {

total\_vehicles++;

total\_worth\_cars += (price \* stock);

}

// Setters

void setManufacturer(string manufacturer) { this->manufacturer = manufacturer; }

void setModel(string model) { this->model = model; }

void setColor(string color) { this->color = color; }

void setModelYear(int model\_year) { this->model\_year = model\_year; }

void setEngineCapacity(float engine\_capacity) { this->engine\_capacity = engine\_capacity; }

void setPrice(float price) { this->price = price; }

void setStock(int stock) { this->stock = stock; }

// Getters

string getManufacturer() const { return manufacturer; }

string getModel() const { return model; }

string getColor() const { return color; }

int getModelYear() const { return model\_year; }

float getEngineCapacity() const { return engine\_capacity; }

long double getPrice() const { return price; }

int getStock() const { return stock; }

// Display function

virtual void display() const {

cout << "----------------------------------------" << endl;

cout << "Manufacturer: " << manufacturer << endl;

cout << "Model: " << model << endl;

cout << "Color: " << color << endl;

cout << "Model Year: " << model\_year << endl;

cout << "Engine Cap: " << engine\_capacity << " CC" << endl;

cout << "Price: Rs." << price << endl;

cout << "Stock: " << stock << endl;

}

static int getTotalVehiclesMade() { return total\_vehicles; }

static long double getTotalWorth() { return total\_worth\_cars; }

friend class Showroom;

friend bool operator==<>(const Vehicle& a, const Vehicle& b);

friend bool operator< <>(const Vehicle& a, const Vehicle& b);

};

int Vehicle::total\_vehicles = 0;

long double Vehicle::total\_worth\_cars = 0;

// Car class inherits from Vehicle

class Car : public Vehicle {

private:

string body\_type;

bool convertible;

int seating\_capacity;

static int total\_cars;

public:

Car() { total\_cars++; }

Car(string manufacturer, string model, int model\_year,

float engine\_capacity, float price, string color, int stock) :

Vehicle(manufacturer, model, model\_year, engine\_capacity,

price, color, stock) { total\_cars++; }

// Setters

void setBodyType(string body\_type) { this->body\_type = body\_type; }

void setConvertible(bool convertible) { this->convertible = convertible; }

void setSeatingCapacity(int seating\_capacity) { this->seating\_capacity = seating\_capacity; }

// Getters

string getBodyType() const { return body\_type; }

bool isConvertible() const { return convertible; }

int getSeatingCapacity() const { return seating\_capacity; }

// Override display

void display() const override {

cout << "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Car Details \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*" << endl;

Vehicle::display();

cout << "Body Type: " << body\_type << endl;

cout << "Convertible: " << (convertible ? "Yes" : "No") << endl;

cout << "Seating Capacity: " << seating\_capacity << endl;

cout << "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*" << endl << endl;

}

static int getTotalCarsMade() { return total\_cars; }

};

int Car::total\_cars = 0;

// Bike class inherits from Vehicle

class Bike : public Vehicle {

private:

bool is\_chain\_drive, has\_kick\_start;

string bike\_type;

static int total\_bikes;

public:

Bike() { total\_bikes++; }

Bike(string manufacturer, string model, int model\_year,

float engine\_capacity, float price, string color, int stock) :

Vehicle(manufacturer, model, model\_year, engine\_capacity,

price, color, stock) { total\_bikes++; }

// Setters

void setIsChainDrive(bool is\_chain\_drive) { this->is\_chain\_drive = is\_chain\_drive; }

void setHasKickStart(bool has\_kick\_start) { this->has\_kick\_start = has\_kick\_start; }

void setBikeType(string bike\_type) { this->bike\_type = bike\_type; }

// Getters

bool getIsChainDrive() const { return is\_chain\_drive; }

bool getHasKickStart() const { return has\_kick\_start; }

string getBikeType() const { return bike\_type; }

// Override display

void display() const override {

cout << "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Bike Details \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*" << endl;

Vehicle::display();

cout << "Bike Type: " << bike\_type << endl;

cout << "Chain Drive: " << (is\_chain\_drive ? "Yes" : "No") << endl;

cout << "Has Kick Start: " << (has\_kick\_start ? "Yes" : "No") << endl;

cout << "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*" << endl << endl;

}

static int getTotalBikesMade() { return total\_bikes; }

};

int Bike::total\_bikes = 0;

// EV class inherits from Car

class EV : public Car {

private:

float battery\_capacity, range\_per\_charge, charge\_time;

static int total\_evs;

public:

EV() { total\_evs++; }

EV(string manufacturer, string model, int model\_year,

float engine\_capacity, float price, string color, int stock) :

Car(manufacturer, model, model\_year, engine\_capacity,

price, color, stock) { total\_evs++; }

// Setters

void setBatteryCapacity(float battery\_capacity) { this->battery\_capacity = battery\_capacity; }

void setRange(float range\_per\_charge) { this->range\_per\_charge = range\_per\_charge; }

void setChargeTime(float charge\_time) { this->charge\_time = charge\_time; }

// Getters

float getBatteryCapacity() const { return battery\_capacity; }

float getRange() const { return range\_per\_charge; }

float getChargeTime() const { return charge\_time; }

// Override display

void display() const override {

cout << "\*\*\*\*\*\*\*\*\*\*\*\*\*\* EV Car Details \*\*\*\*\*\*\*\*\*\*\*\*\*\*" << endl;

Vehicle::display();

cout << "Battery Capacity: " << battery\_capacity << " kWh" << endl;

cout << "Range per Charge: " << range\_per\_charge << " km" << endl;

cout << "Charge Time: " << charge\_time << " hours" << endl;

cout << "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*" << endl << endl;

}

static int getTotalEVsMade() { return total\_evs; }

};

int EV::total\_evs = 0;

// Truck class inherits from Vehicle

class Truck : public Vehicle {

private:

float load\_capacity, axles;

static int total\_trucks;

public:

Truck() { total\_trucks++; }

Truck(string manufacturer, string model, int model\_year,

float engine\_capacity, float price, string color, int stock) :

Vehicle(manufacturer, model, model\_year, engine\_capacity,

price, color, stock) { total\_trucks++; }

// Setters

void setLoadCapacity(float load\_capacity) { this->load\_capacity = load\_capacity; }

void setAxles(float axles) { this->axles = axles; }

// Getters

float getLoadCapacity() const { return load\_capacity; }

float getAxles() const { return axles; }

// Override display

void display() const override {

cout << "\*\*\*\*\*\*\*\*\*\*\*\*\*\* Truck Details \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*" << endl;

Vehicle::display();

cout << "Load Capacity: " << load\_capacity << " kg" << endl;

cout << "Axles: " << axles << endl;

cout << "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*" << endl << endl;

}

static int getTotalTrucksMade() { return total\_trucks; }

};

int Truck::total\_trucks = 0;

// Person base class

class Person {

protected:

string name;

int age;

string contact;

public:

Person() {}

Person(string n, int a, string c) : name(n), age(a), contact(c) {}

// Setters

void setName(string n) { name = n; }

void setAge(int a) { age = a; }

void setContact(string c) { contact = c; }

// Getters

string getName() const { return name; }

int getAge() const { return age; }

string getContact() const { return contact; }

// Display function

virtual void displayInfo() const {

cout << "Name: " << name << endl;

cout << "Age: " << age << endl;

cout << "Contact: " << contact << endl;

}

};

// Customer class inherits from Person

class Customer : public Person {

private:

int customerID;

string customerUID;

static int nextCustomerID;

static int customerCount;

public:

Customer() {

customerID = nextCustomerID++;

customerUID = "CUSTOMER-" + to\_string(customerID);

customerCount++;

}

Customer(string n, int a, string c) : Person(n, a, c) {

customerID = nextCustomerID++;

customerUID = "CUSTOMER-" + to\_string(customerID);

customerCount++;

}

// Getters

int getCustomerID() const { return customerID; }

string getCustomerUID() const { return customerUID; }

// Override display

void displayInfo() const override {

cout << "\*\*\*\*\*\*\*\*\*\*\*\* Customer Details \*\*\*\*\*\*\*\*\*\*\*\*" << endl;

cout << "Customer UID: " << customerUID << endl;

Person::displayInfo();

cout << "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*" << endl << endl;

}

static int getCustomerCount() { return customerCount; }

friend class Showroom;

};

int Customer::nextCustomerID = 1001;

int Customer::customerCount = 0;

// Employee class inherits from Person

class Employee : public Person {

private:

int employeeID;

string position, employeeUID;

static int nextEmployeeID;

static int employeeCount;

public:

Employee() {

employeeID = nextEmployeeID++;

employeeUID = "EMP-" + to\_string(employeeID);

employeeCount++;

}

Employee(string n, int a, string c, string p) : Person(n, a, c), position(p) {

employeeID = nextEmployeeID++;

employeeUID = "EMP-" + to\_string(employeeID);

employeeCount++;

}

// Setters

void setPosition(string pos) { position = pos; }

// Getters

int getEmployeeID() const { return employeeID; }

string getPosition() const { return position; }

string getEmployeeUID() const { return employeeUID; }

// Override display

void displayInfo() const override {

cout << "\*\*\*\*\*\*\*\*\*\*\*\* Employee Details \*\*\*\*\*\*\*\*\*\*\*\*" << endl;

cout << "Employee UID: " << employeeUID << endl;

Person::displayInfo();

cout << "Position: " << position << endl;

cout << "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*" << endl << endl;

}

static int getEmployeeCount() { return employeeCount; }

friend class Showroom;

};

int Employee::nextEmployeeID = 1001;

int Employee::employeeCount = 0;

// Sale class

class Sale {

private:

int sale\_id;

string sale\_UID;

long double price;

Vehicle\* v;

Customer\* c;

Employee\* e;

static int sale\_count;

static int next\_saleid;

static long double total\_profit;

public:

Sale(Customer\* cust, Vehicle\* veh, Employee\* emp) : c(cust), v(veh), e(emp) {

sale\_count++;

sale\_id = next\_saleid++;

sale\_UID = "S-" + to\_string(sale\_id);

total\_profit += v->getPrice();

price = v->getPrice();

v->setStock(v->getStock() - 1);

}

// Display sale details

void display() const {

cout << "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Sale Details \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*" << endl;

cout << "Sale UID: " << sale\_UID << endl;

v->display();

c->displayInfo();

e->displayInfo();

cout << "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*" << endl << endl;

}

// Getters

int getSaleID() const { return sale\_id; }

string getVehicleModel() const { return v->getModel(); }

int getCustomerID() const { return c->getCustomerID(); }

string getCustomerName() const { return c->getName(); }

string getSaleUID() const { return sale\_UID; }

long double getPrice() const { return price; }

static int getSaleCount() { return sale\_count; }

static long double getTotalProfit() { return total\_profit; }

friend class Showroom;

friend bool operator==<>(const Sale& a, const Sale& b);

friend bool operator< <>(const Sale& a, const Sale& b);

};

int Sale::next\_saleid = 1001;

int Sale::sale\_count = 0;

long double Sale::total\_profit = 0;

// Showroom class

class Showroom {

private:

Vehicle\* vehicles[100];

Customer\* customers[100];

Employee\* employees[30];

Sale\* sales[100];

int vehicleCount = 0;

int customerCount = 0;

int employeeCount = 0;

int saleCount = 0;

public:

// Add a vehicle

void addVehicle(Vehicle\* v) {

if (vehicleCount < 100) {

vehicles[vehicleCount++] = v;

} else {

cout << "Vehicle list full. Cannot add more vehicles." << endl;

}

}

// Display all vehicles

void displayVehicles() const {

cout << "\n========== All Vehicles in Showroom ==========\n";

for (int i = 0; i < vehicleCount; i++) {

vehicles[i]->display();

}

cout << "==============================================" << endl << endl;

}

// Search for a vehicle by model

void searchVehicle(string model) const {

bool found = false;

for (int i = 0; i < vehicleCount; i++) {

if (vehicles[i]->getModel() == model) {

vehicles[i]->display();

found = true;

}

}

if (!found) {

cout << "Vehicle not found.\n";

}

}

// Delete a vehicle by model

void deleteVehicle(string model) {

bool found = false;

for (int i = 0; i < vehicleCount; i++) {

if (vehicles[i]->getModel() == model) {

Vehicle::total\_worth\_cars -= (vehicles[i]->getPrice() \* vehicles[i]->getStock());

delete vehicles[i];

for (int j = i; j < vehicleCount - 1; j++) {

vehicles[j] = vehicles[j + 1];

}

vehicleCount--;

found = true;

cout << "Vehicle Deleted!\n";

break;

}

}

if (!found) {

cout << "Vehicle not found.\n";

}

}

// Add Customer

void addCustomer(string name, int age, string contact) {

if (customerCount < 100) {

customers[customerCount++] = new Customer(name, age, contact);

} else {

cout << "Customer list full. Cannot add more customers." << endl;

}

}

// Display all customers

void displayCustomers() const {

cout << "\n=========== All Customers ===========\n";

for (int i = 0; i < customerCount; i++) {

customers[i]->displayInfo();

}

cout << "====================================" << endl << endl;

}

// Add Employee

void addEmployee(string name, int age, string contact, string position) {

if (employeeCount < 30) {

employees[employeeCount++] = new Employee(name, age, contact, position);

} else {

cout << "Employee list full. Cannot add more employees." << endl;

}

}

// Display all employees

void displayEmployees() const {

cout << "\n========== All Employees ==========\n";

for (int i = 0; i < employeeCount; i++) {

employees[i]->displayInfo();

}

cout << "==================================" << endl << endl;

}

// Sell vehicle to customer by employee

void sellVehicle(string model, int customerID, int employeeID) {

Vehicle\* v = nullptr;

for (int i = 0; i < vehicleCount; i++) {

if (vehicles[i]->getModel() == model && vehicles[i]->getStock() > 0) {

v = vehicles[i];

break;

}

}

if (v == nullptr) {

cout << "Vehicle not available or out of stock.\n";

return;

}

Customer\* c = nullptr;

Employee\* e = nullptr;

bool foundCustomer = false, foundEmployee = false;

for (int i = 0; i < customerCount; i++) {

if (customers[i]->getCustomerID() == customerID) {

c = customers[i];

foundCustomer = true;

break;

}

}

for (int i = 0; i < employeeCount; i++) {

if (employees[i]->getEmployeeID() == employeeID) {

e = employees[i];

foundEmployee = true;

break;

}

}

if (!foundCustomer || !foundEmployee) {

cout << "Invalid Customer or Employee ID.\n";

return;

}

if (saleCount < 100) {

Sale\* s = new Sale(c, v, e); // Dynamically allocate Sale object

sales[saleCount++] = s; // Store the pointer to the dynamically allocated object

cout << "\n\*\*\nSale Successful\nSale ID: " << s->getSaleID() << endl;

} else {

cout << "Sale list full. Cannot add more sales." << endl;

}

}

// Display all sales transactions

void displayTransactions() const {

cout << "\n========== All Sales Transactions ==========\n";

for (int i = 0; i < saleCount; i++) {

sales[i]->display();

}

cout << "============================================" << endl << endl;

}

// Alert for low stock vehicles

void lowStockAlert() const {

cout << "\n\*\*\*\*\*\*\*\*\*\* Low Stock Alert \*\*\*\*\*\*\*\*\*\*\n";

bool found = false;

for (int i = 0; i < vehicleCount; i++) {

if (vehicles[i]->getStock() < 3) {

cout << "Low stock alert: " << endl;

vehicles[i]->display();

found = true;

}

}

if (!found) {

cout << "No low stock vehicles currently.\n";

}

cout << "\*\n\n";

}

// Generate report

void getReport() const {

cout << "\*AA WHEELS REPORT" << endl << endl;

cout << "Total Vehicles: " << Vehicle::getTotalVehiclesMade() << endl;

cout << "Total Cars: " << Car::getTotalCarsMade() << endl;

cout << "Total Bikes: " << Bike::getTotalBikesMade() << endl;

cout << "Total EVs: " << EV::getTotalEVsMade() << endl;

cout << "Total Trucks: " << Truck::getTotalTrucksMade() << endl;

cout << "Total Vehicles Sold: " << Sale::getSaleCount() << endl;

cout << "\nTotal Customers: " << Customer::getCustomerCount() << endl;

cout << "Total Employees: " << Employee::getEmployeeCount() << endl;

cout << "\nTotal Worth Available: Rs." << Vehicle::getTotalWorth() << endl;

cout << "Total Money Made: Rs." << Sale::getTotalProfit() << endl;

}

// Price comparison between two vehicles

void priceComparison(string model1, string model2) const {

Vehicle\* v1 = nullptr;

Vehicle\* v2 = nullptr;

for (int i = 0; i < vehicleCount; i++) {

if (vehicles[i]->getModel() == model1) {

v1 = vehicles[i];

break;

}

}

for (int i = 0; i < vehicleCount; i++) {

if (vehicles[i]->getModel() == model2) {

v2 = vehicles[i];

break;

}

}

if (v1 == nullptr || v2 == nullptr) {

cout << "One or both vehicles not found.\n";

return;

}

if (\*v1 == \*v2) {

cout << "Both vehicles have the same price." << endl;

} else if (\*v1 < \*v2) {

cout << v1->getModel() << " is cheaper." << endl;

} else {

cout << v2->getModel() << " is more expensive." << endl;

}

}

// Sale price comparison

void salePriceComparison(int id1, int id2) const {

Sale\* s1 = nullptr;

Sale\* s2 = nullptr;

for (int i = 0; i < saleCount; i++) {

if (sales[i]->getSaleID() == id1) {

s1 = sales[i];

break;

}

}

for (int i = 0; i < saleCount; i++) {

if (sales[i]->getSaleID() == id2) {

s2 = sales[i];

break;

}

}

if (s1 == nullptr || s2 == nullptr) {

cout << "One or both sales not found.\n";

return;

}

if (\*s1 == \*s2) {

cout << "Both sales have the same price." << endl;

} else if (\*s1 < \*s2) {

cout << s1->getSaleUID() << " is cheaper." << endl;

} else {

cout << s2->getSaleUID() << " is more expensive." << endl;

}

}

// Destructor to clean up dynamically allocated memory

~Showroom() {

for (int i = 0; i < customerCount; i++) {

delete customers[i];

}

for (int i = 0; i < saleCount; i++) {

delete sales[i];

}

for (int i = 0; i < vehicleCount; i++) {

delete vehicles[i];

}

for (int i = 0; i < employeeCount; i++) {

delete employees[i];

}

}

};

int main() {

Showroom showroom;

int choice;

cout << "----- AA WHEELS -----\n\n";

do {

cout << "\n--- Showroom Management System Menu ---\n";

cout << "1. Add Vehicle\n";

cout << "2. Add Customer\n";

cout << "3. Add Employee\n";

cout << "4. Display All Vehicles\n";

cout << "5. Display All Customers\n";

cout << "6. Display All Employees\n";

cout << "7. Make a Sale\n";

cout << "8. View All Sales\n";

cout << "9. Low Stock Check\n";

cout << "10. Generate Report\n";

cout << "11. Search Vehicle\n";

cout << "12. Delete Vehicle\n";

cout << "13. Vehicle Price Comparison\n";

cout << "14. Sale Price Comparison\n";

cout << "15. Exit\n";

cout << "Enter your choice: ";

cin >> choice;

switch (choice) {

case 1: {

if (Vehicle::getTotalVehiclesMade() < 100) {

int type;

cout << "\nSelect Vehicle Type:\n1. Car\n2. Bike\n3. EV\n4. Truck\nChoice: ";

cin >> type;

string manu, model, color;

int year, stock;

float eng, price;

cout << "Enter Manufacturer: "; cin >> manu;

cout << "Enter Model: "; cin >> model;

cout << "Enter Model Year: "; cin >> year;

cout << "Enter Color: "; cin >> color;

cout << "Enter Engine Capacity: "; cin >> eng;

cout << "Enter Price: "; cin >> price;

cout << "Enter Stock: "; cin >> stock;

if (type == 1) {

Car\* c = new Car(manu, model, year, eng, price, color, stock);

string bodytype;

bool convertible;

int seating\_cap;

cout << "\nEnter Body Type: "; cin >> bodytype;

cout << "\nEnter Convertible (1 for true, 0 for false): "; cin >> convertible;

cout << "\nEnter Seating Capacity: "; cin >> seating\_cap;

c->setBodyType(bodytype);

c->setConvertible(convertible);

c->setSeatingCapacity(seating\_cap);

showroom.addVehicle(c);

} else if (type == 2) {

Bike\* b = new Bike(manu, model, year, eng, price, color, stock);

bool kickstart, chaindrive;

string biketype;

cout << "\nIs Bike Kick Start? (1 for true, 0 for false): "; cin >> kickstart;

cout << "\nIs Bike with Chain Drive? (1 for true, 0 for false): "; cin >> chaindrive;

cout << "\nEnter Bike Type: "; cin >> biketype;

b->setBikeType(biketype);

b->setHasKickStart(kickstart);

b->setIsChainDrive(chaindrive);

showroom.addVehicle(b);

} else if (type == 3) {

EV\* ev = new EV(manu, model, year, eng, price, color, stock);

float chargetime, batterycapacity, range;

cout << "\nEnter Charge Time: "; cin >> chargetime;

cout << "\nEnter Battery Capacity: "; cin >> batterycapacity;

cout << "\nEnter Range per Charge: "; cin >> range;

ev->setBatteryCapacity(batterycapacity);

ev->setChargeTime(chargetime);

ev->setRange(range);

showroom.addVehicle(ev);

} else if (type == 4) {

Truck\* truck = new Truck(manu, model, year, eng, price, color, stock);

float loadcapacity, axles;

cout << "\nEnter Load Capacity: "; cin >> loadcapacity;

cout << "\nEnter Axles: "; cin >> axles;

truck->setLoadCapacity(loadcapacity);

truck->setAxles(axles);

showroom.addVehicle(truck);

} else {

cout << "Invalid vehicle type.\n";

}

} else {

cout << "\nMax Vehicles Reached!!!" << endl;

}

break;

}

case 2: {

string name;

int age;

string contact;

cout << "Enter Customer Name: ";

cin >> name;

cout << "Enter Customer Age: ";

cin >> age;

cout << "Enter Customer Contact: ";

cin >> contact;

showroom.addCustomer(name, age, contact);

break;

}

case 3: {

string name;

int age;

string contact;

string position;

cout << "Enter Employee Name: ";

cin >> name;

cout << "Enter Employee Age: ";

cin >> age;

cout << "Enter Employee Contact: ";

cin >> contact;

cout << "Enter Employee Position: ";

cin >> position;

showroom.addEmployee(name, age, contact, position);

break;

}

case 4:

showroom.displayVehicles();

break;

case 5:

showroom.displayCustomers();

break;

case 6:

showroom.displayEmployees();

break;

case 7: {

string model;

int cid, eid;

cout << "Enter Model: ";

cin >> model;

cout << "\nEnter Customer ID: ";

cin >> cid;

cout << "\nEnter Employee ID: ";

cin >> eid;

showroom.sellVehicle(model, cid, eid);

break;

}

case 8:

showroom.displayTransactions();

break;

case 9:

showroom.lowStockAlert();

break;

case 10:

showroom.getReport();

break;

case 11: {

cout << "Enter Model Name: ";

string model;

cin >> model;

showroom.searchVehicle(model);

break;

}

case 12: {

cout << "Enter Model Name: ";

string model;

cin >> model;

showroom.deleteVehicle(model);

break;

}

case 13: {

string m1, m2;

cout << "Enter model 1: ";

cin >> m1;

cout << "Enter model 2: ";

cin >> m2;

showroom.priceComparison(m1, m2);

break;

}

case 14: {

int m1, m2;

cout << "Enter Sale ID 1: ";

cin >> m1;

cout << "Enter Sale ID 2: ";

cin >> m2;

showroom.salePriceComparison(m1, m2);

break;

}

case 15:

cout << "Exiting...\n";

break;

default:

cout << "Invalid choice.\n";

}

} while (choice != 15);

return 0;

}

**Output:**

