МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РФ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«ВОРОНЕЖСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ИНЖЕНЕРНЫХ ТЕХНОЛОГИЙ»

Факультет управления и информатики в технологических системах
Кафедра Информационной безопасности
Направление подготовки
(специальность) 10.05.03 Информационная безопасность автоматизированных систем

Отчет

по практике по технологиям и методам программирования
наименование (вид) практики

Выполнил студент гр. УБ-42
Крылов Никита Романович
(Ф.И.О.)

_____
(подпись)

Проверили:

Маслов А.А.
(Ф.И.О.)

_____ _____
(оценка) *(подпись)*

_____
(дата)

Воронеж - 2025

Задание 15.

Вариант 15). Продажа автомобилей. Создать родительский класс «Автомобили» (марка автомобиля, год выпуска, цена автомобиля, комплетация, страна производитель, дата продажи, ФИО покупателя) и дочерние классы:

– «Поддержанные авто» (степень сохранности, ФИО владельца, пробег);

– «Спортивные» (кол-во секунд до набора скорости, объем двигателя, мощность);

– «Спецтехника» (вид (строительная, грузовая, дорожная и т.д.), масса, габаритные размеры).

Реализовать класс для хранения списка проданных автомобилей с методом добавления нового автомобиля и методом печати списка автомобилей.

Код программы:

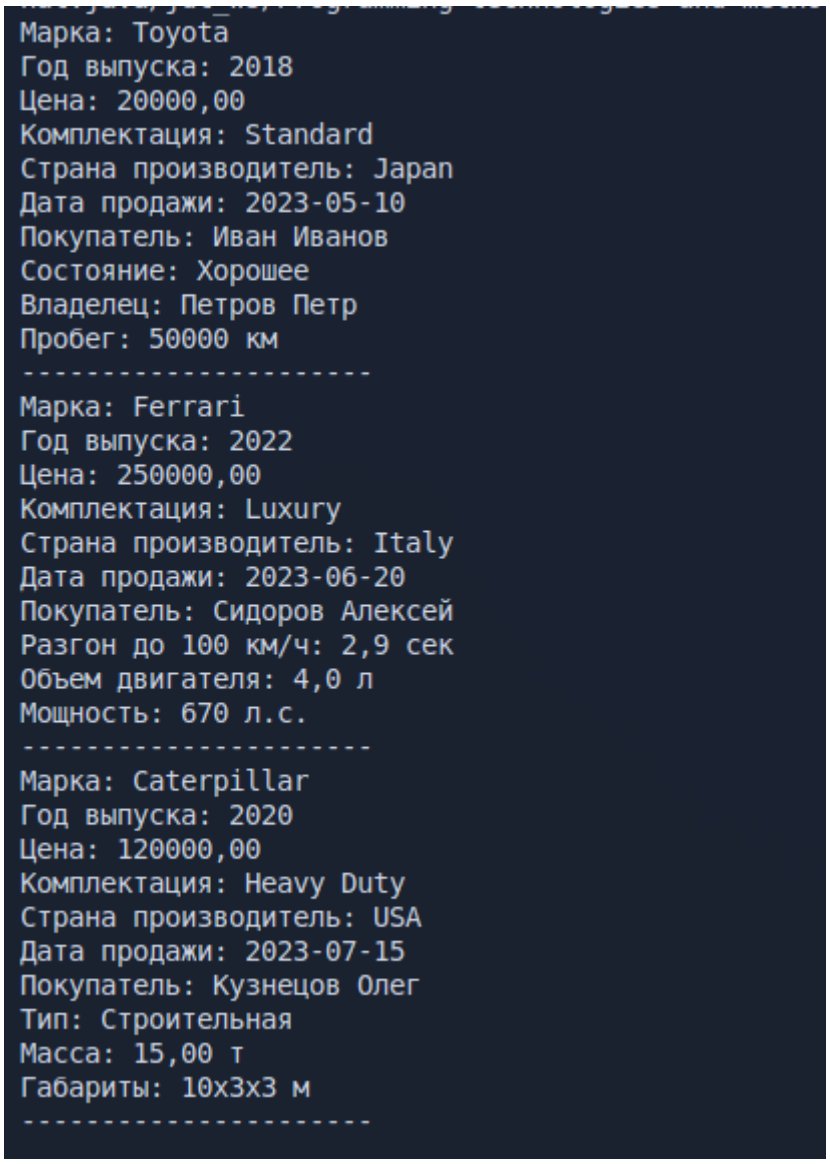
```
package Pr4;


import java.time.LocalDate;
import java.util.ArrayList;
import java.util.List;


class Automobile {
protected String brand;
protected int year;
protected double price;
protected String configuration;
protected String countryOfOrigin;
protected LocalDate saleDate;
protected String buyerName;


public Automobile(String brand, int year, double price, String configuration,
String countryOfOrigin, LocalDate saleDate, String buyerName) {
this.brand = brand;
this.year = year;
this.price = price;
this.configuration = configuration;
this.countryOfOrigin = countryOfOrigin;
this.saleDate = saleDate;
this.buyerName = buyerName;
}
```

```
@Override
public String toString() {
return String.format("Марка: %s\nГод выпуска: %d\nЦена: %.2f\nКомплектация:
%s\nСтрана производитель: %s\nДата продажи: %s\nПокупатель: %s",
brand, year, price, configuration, countryOfOrigin, saleDate, buyerName);
}
}

class UsedCar extends Automobile {
private String condition;//состояние автомобиля
private String ownerName;//имя владельца
private int mileage;//пробег

public UsedCar(String brand, int year, double price, String configuration,
String countryOfOrigin,LocalDate saleDate, String buyerName, String condition, String
ownerName, int mileage) {
super(brand, year, price, configuration, countryOfOrigin, saleDate, buyerName);
this.condition = condition;
this.ownerName = ownerName;
this.mileage = mileage;
}

@Override
public String toString() {
return super.toString() + String.format("\nСостояние: %s\nВладелец: %s\nПробег: %d км",
condition, ownerName, mileage);
}
}

class SportsCar extends Automobile {
private double acceleration;//время разгона до 100 км/ч
private double engineVolume;//объем двигателя
private int power;//мощность
```

```
public SportsCar(String brand, int year, double price, String configuration, String
countryOfOrigin,
LocalDate saleDate, String buyerName, double acceleration, double engineVolume, int power) {
super(brand, year, price, configuration, countryOfOrigin, saleDate, buyerName);
this.acceleration = acceleration;
this.engineVolume = engineVolume;
this.power = power;
}

@Override
public String toString() {
return super.toString() + String.format("\nРазгон до 100 км/ч: %.1f сек\nОбъем двигателя:
%.1f л\nМощность: %d л.с.", acceleration, engineVolume, power);
}
}

class SpecialEquipment extends Automobile {
private String type;//тип техники
private double weight;//вес
private String dimensions;//габариты

public SpecialEquipment(String brand, int year, double price, String configuration, String
countryOfOrigin,
LocalDate saleDate, String buyerName, String type, double weight, String dimensions) {
super(brand, year, price, configuration, countryOfOrigin, saleDate, buyerName);
this.type = type;
this.weight = weight;
this.dimensions = dimensions;
}

@Override
public String toString() {
return super.toString() + String.format("\nТип: %s\nМасса: %.2f т\nГабариты: %s", type,
weight, dimensions);
}
```

```
}

class CarSales {
private List<Automobile> soldCars = new ArrayList<>();

public void addCar(Automobile car) {
soldCars.add(car);
}

public void printSales() {
for (Automobile car : soldCars) {
System.out.println(car);
System.out.println("----------------------");
}
}
}

public class Main {
public static void main(String[] args) {
CarSales sales = new CarSales();

sales.addCar(new UsedCar("Toyota", 2018, 20000.0, "Standard", "Japan", LocalDate.of(2023, 5,
10), "Иван Иванов", "Хорошее", "Петров Петр", 50000));
sales.addCar(new SportsCar("Ferrari", 2022, 250000.0, "Luxury", "Italy", LocalDate.of(2023, 6,
20), "Сидоров Алексей", 2.9, 4.0, 670));
sales.addCar(new SpecialEquipment("Caterpillar", 2020, 120000.0, "Heavy Duty", "USA",
LocalDate.of(2023, 7, 15), "Кузнецов Олег", "Строительная", 15.0, "10x3x3 м"));

sales.printSales();
}
}
```

Результат программы.



Вывод.

Сегодня я изучил и научился работать с ключевыми принципами ООП на примере программы по учету продаж автомобилей. Этот пример позволил мне закрепить понимание таких концепций, как наследование, полиморфизм, инкапсуляция и абстракция, а также работу с классами, объектами и конструкторами.

1.Классы и объекты

Программа демонстрирует, как класс `Automobile` служит базовым шаблоном для создания объектов, представляющих автомобили. Его поля (`brand`, `year`, `price` и др.) хранят состояние объекта, а конструктор инициализирует их при создании.

2. Наследование

Классы `UsedCar`, `SportsCar` и `SpecialEquipment` наследуют общие свойства от `Automobile` и добавляют свои уникальные характеристики:

- `UsedCar` — состояние, владельца и пробег,
- `SportsCar` — разгон, объем двигателя и мощность,
- `SpecialEquipment` — тип, вес и габариты.

Этот подход позволяет избежать дублирования кода и логично структурировать иерархию классов.
3. Полиморфизм
Проявился в двух аспектах:
- Переопределение методов (`@Override`)— каждый подкласс изменяет `toString()`, чтобы выводить свою специфическую информацию.
- Использование общего типа (`Automobile`)— в классе `CarSales` все автомобили хранятся в списке `List<Automobile>`, но при вызове `toString()` для каждого объекта выполняется его собственная версия метода.
4. Инкапсуляция
Поля классов объявлены как `protected` или `private`, что ограничивает прямой доступ к ним извне и обеспечивает контроль через методы (например, через конструкторы и `toString()`).
5.Работа с коллекциями
Класс `CarSales` использует `List<Automobile>` для хранения проданных автомобилей, демонстрируя, как полиморфизм позволяет работать с разными типами через единый интерфейс.
Эта программа — отличный пример применения ООП в Java. Она помогла мне понять:
- Как создавать иерархии классов через наследование,
- Как использовать полиморфизм для гибкости кода,
- Почему важна инкапсуляция для защиты данных,
- Как переопределять методы для адаптации поведения подклассов.