

Лабораторная работа №6

Интерфейсы

Часть 1 – разработка и реализация собственных интерфейсов

Требуется разработать библиотеку классов, в которой заданы классы **Person** (человек), **Client** (клиент банка) и **Firm** (фирма). Класс **Client** является производным от класса **Person**.

Класс **Client** должен реализовывать интерфейс **IAccount** (счёт в банке), в котором должны быть определены следующие элементы:

- **string AccNumber** – свойство, возвращающее номер счёта;
- **int CurrentSum** – свойство, возвращающее текущую сумму;
- **int Percentage** – свойство, возвращающее процент начислений;
- **void Put(int sum)** – метод, обеспечивающий пополнение счёта на сумму **sum**;
- **void Withdraw(int sum)** – метод, позволяющий снять со счёта сумму **sum**.

Для того, чтобы разработанные классы можно было потом использовать в других проектах необходимо их реализовать в виде библиотеки классов (библиотеки dll). Для этого при создании проекта необходимо выбрать шаблон «Библиотека классов» (рисунок 1).

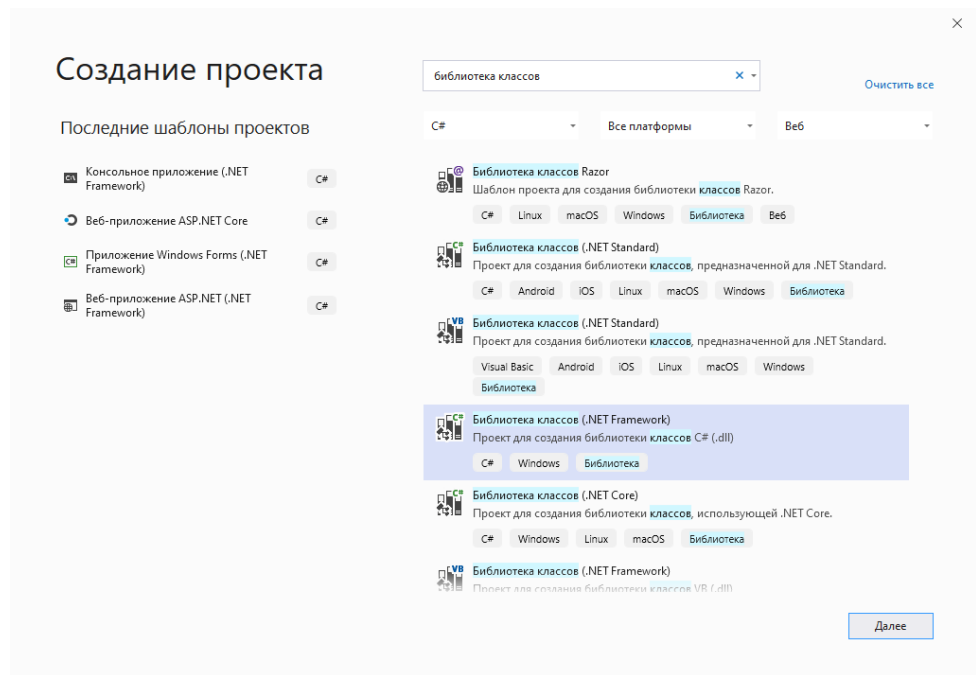


Рисунок 1 – шаблон проекта «Библиотека классов (.NET Framework)»

Код интерфейса IAccount:

```
namespace AccountLib
{
    public interface IAccount
    {
        string AccNumber { get; }
        int CurrentSum { get; }
        int Percentage { get; }
        void Put(int sum);
        void Withdraw(int sum);
    }
}
```

Класс **Firm** должен реализовывать интерфейсы **ISCommercial** (торговая организация) и **IAccount**. В интерфейсе **ISCommercial** должны быть определены следующие элементы:

- **int Funds** – свойство, возвращающее размер денежного капитала фирмы;
- **void Buy(int sum)** – метод, позволяющий купить товары на сумму **sum**;
- **void Sell(int sum)** – метод, обеспечивающий продажу товаров на сумму **sum**.

Код интерфейса ISCommercial:

```

namespace AccountLib
{
    public interface ICommercial
    {
        int Funds { get; }
        void Buy(int sum);
        void Sell(int sum);
    }
}

```

Код класса Person:

```

public class Person
{
    public string FirstName { get; set; }
    public string LastName { get; set; }
    public string Gender { get; set; }
    public DateTime Birthday { get; set; }

    public Person(string firstName, string lastName, string gender, DateTime
birthday)
    {
        FirstName = firstName;
        LastName = lastName;
        Gender = gender;
        Birthday = birthday;
    }

    public override string ToString()
    {
        return $"Имя: {FirstName} \n" +
            $"Фамилия: {LastName} \n" +
            $"Пол: {Gender} \n" +
            $"Дата рождения: {Birthday.ToShortDateString()}";
    }
}

```

Код класса Client до реализации интерфейса:

```

public class Client : Person, IAccount
{
    string accNumber;
    int currentSum;
    int percentage;

    public Client(string firstName, string lastName, string gender, DateTime
birthday, string accNumber, int currentSum, int percentage)
        :base(firstName, lastName, gender, birthday)
    {
        this.accNumber = accNumber;
        this.currentSum = currentSum;
        this.percentage = percentage;
    }

    public override string ToString()
    {

```

```

        return "Информация о клиенте:\n"+
            base.ToString()+
            $"Номер счета: {accNumber}\n" +
            $"Текущая сумма: {currentSum}\n" +
            $"Процент начислений: {percentage}";
    }
}

```

Если используется наследование от интерфейса, но нет реализации интерфейса в классе Visual Studio подчеркнет имя нереализованного интерфейса (рисунок 2)

```

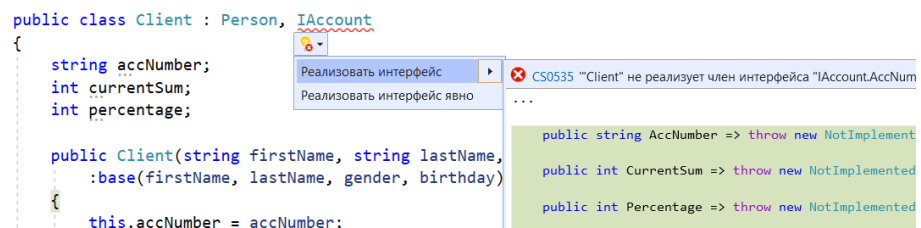
namespace AccountLib
{
    public class Client : Person, IAccount
    {
        string accNumber;
        int currentSum;
        int percentage;

        public Client(string firstName, string lastName,
            :base(firstName, lastName, gender, birth
        {
            this.accNumber = accNumber;

```

Рисунок 2 – ошибка: интерфейс не реализован

Для быстрой реализации интерфейса нужно вызвать помощника решения проблем Visual Studio (кнопка с лампочкой) и выбрать вариант «Реализовать интерфейс» (рисунок 3)



```

public class Client : Person, IAccount
{
    string accNumber;
    int currentSum;
    int percentage;

    public Client(string firstName, string lastName,
        :base(firstName, lastName, gender, birthday)
    {
        this.accNumber = accNumber;

```

Рисунок 3 – быстрое решение проблем

После этого автоматически добавятся пустые реализации методов и свойств интерфейса:

```

public string AccNumber => throw new NotImplementedException();

public int CurrentSum => throw new NotImplementedException();

```

```

public int Percentage => throw new NotImplementedException();

public void Put(int sum)
{
    throw new NotImplementedException();
}

public void Withdraw(int sum)
{
    throw new NotImplementedException();
}

```

Пустая реализация подразумевает генерацию исключения класса **NotImplementedException**. После реализации интерфейса код методов и свойств:

```

public string AccNumber => accNumber;

public int CurrentSum => currentSum;

public int Percentage => percentage;

public void Put(int sum)
{
    currentSum += sum;
}

public void Withdraw(int sum)
{
    if (currentSum > sum)
        currentSum -= sum;
}

```

Так как в классе (интерфейсе) свойства только на чтение, то можно их реализовывать в сокращенном виде. Реализация в полном варианте приведена ниже:

```

public string AccNumber
{
    get
    {
        return accNumber;
    }
}

public int CurrentSum
{
    get
    {
        return currentSum;
    }
}

```

```

public int Percentage
{
    get
    {
        return percentage;
    }
}

```

Полный код класса **Firm**:

```

class Firm : ICommercial, IAccount
{
    string name;
    Person director;
    int funds;
    string accNumber;
    int currentSum;
    int percentage;

    public Firm(string name, Person director, int funds, string accNumber, int
currentSum, int percentage)
    {
        this.name = name;
        this.director = director;
        this.funds = funds;
        this.accNumber = accNumber;
        this.currentSum = currentSum;
        this.percentage = percentage;
    }

    public string Name => name;

    public int Funds => funds;

    public string AccNumber => accNumber;

    public int CurrentSum => currentSum;

    public int Percentage => percentage;

    public void Buy(int sum)
    {
        if(sum < funds)
            funds -= sum;
    }

    public void Put(int sum)
    {
        if(sum <= funds)
        {
            funds -= sum;
            currentSum += sum;
        }
    }

    public void Sell(int sum)
    {
        funds += sum;
    }
}

```

```

public void Withdraw(int sum)
{
    if(sum <= currentSum)
    {
        currentSum -= sum;
        funds += sum;
    }
}

public override string ToString()
{
    return $"Данные о фирме: \n" +
        $"Название: {name}\n" +
        $"Директор: {director.LastName}\n" +
        $"Капитал: {funds}\n" +
        $"Номер счета: {accNumber}\n" +
        $"Сумма на счету: {currentSum:C}";
}
}

```

Схема классов, созданная в Visual Studio, приведена на рисунке 4.

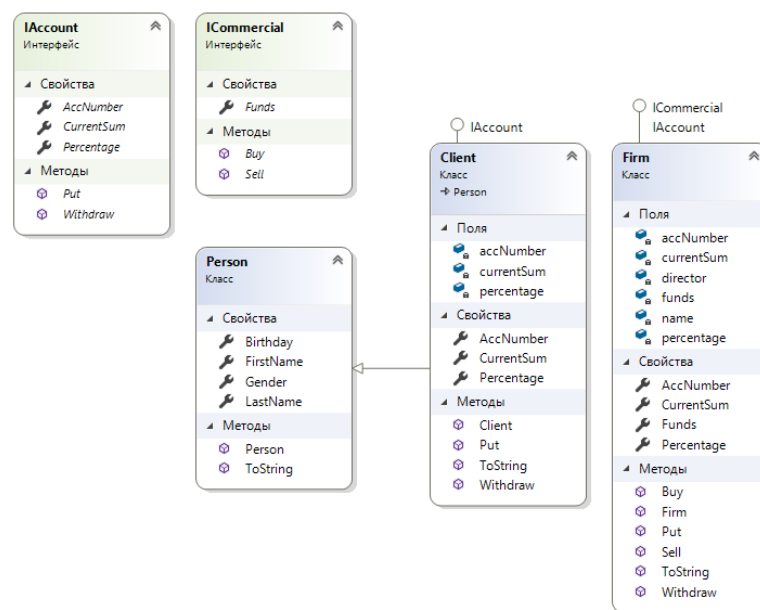


Рисунок 4 – диаграмма классов

После сборки проекта в папке bin (в подпапках Debug или Release – зависит от выбранной конфигурации) появятся dll файл с библиотекой классов (рисунок 5).

ЛБ > ЛБ Интерфейсы > AccountLib > bin > Release				
Имя	Дата изменения	Тип	Размер	
AccountLib.dll	02.11.2020 20:24	Расширение при...	7 КБ	
AccountLib.pdb	02.11.2020 20:24	Program Debug D...	34 КБ	

Рисунок 5 – собранный проект

Для демонстрации работы библиотеки создадим новый проект – консольное приложение. В свойствах проекта добавим ссылку на созданную библиотеку (рисунки 6-7).

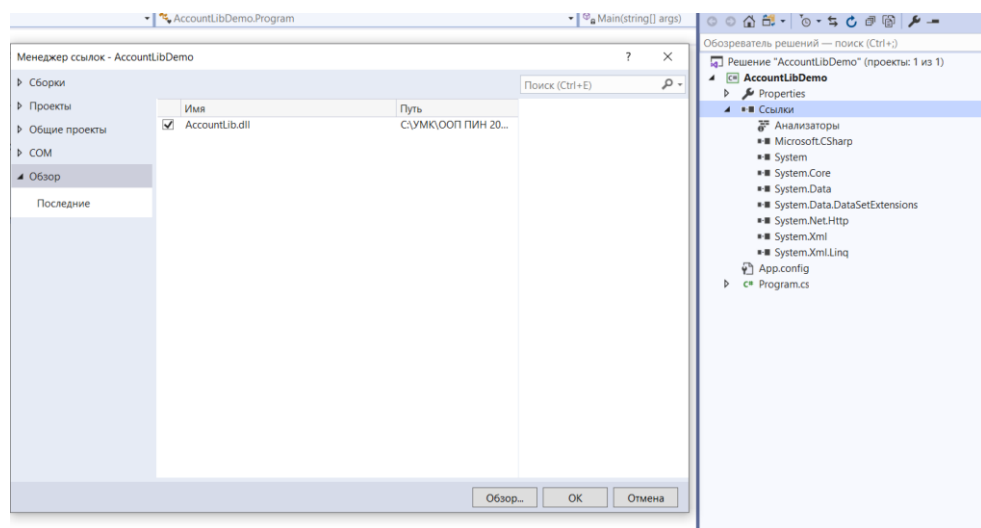


Рисунок 6 – добавление ссылки на библиотеку

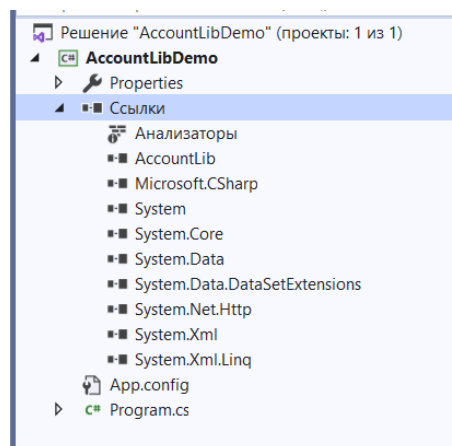


Рисунок 7 – добавленная ссылка

Теперь мы можем использовать классы, созданные в библиотеки в новом проекте. Для начала, естественно, нужно подключить требуемое пространство имен:

```
using AccountLib;
```

Код функции main демо проекта:

```
Console.Title = "Реализация интерфейсов";
Console.OutputEncoding = Encoding.Unicode;

Client client1 = new Client("Иванов", "Иван", "мужской", new
DateTime(1987, 09, 12), "452115125", 200000, 150);
Client client2 = new Client("Петров", "Матвей", "мужской", new
DateTime(1995, 04, 25), "548358348", 10000, 50);
Client client3 = new Client("Сидоров", "Сергей", "мужской", new
DateTime(2000, 01, 25), "383868322", 360000, 250);

Client[] clients = new Client[] { client1, client2, client3 };
Console.WriteLine("Клиенты банка: ");
foreach (var client in clients)
    Console.WriteLine(client.ToString());

client1.Put(35000);
Console.WriteLine("На счету {0} сумма: {1:C}", client1.AccNumber,
client1.CurrentSum);

client2.Withdraw(7000);
Console.WriteLine("На счету {0} сумма: {1:C}", client2.AccNumber,
client2.CurrentSum);

Firm firm1 = new Firm("ООО Кошкин дом", client1, 2500000, "235235135",
1200000, 250);
Console.WriteLine(firm1.ToString());

firm1.Buy(300000);
Console.WriteLine("Капитал фирмы {0} составляет {1:C}", firm1.Name,
firm1.Funds);

firm1.Sell(1000000);
Console.WriteLine("Капитал фирмы {0} составляет {1:C}", firm1.Name,
firm1.Funds);

firm1.Put(500000);
Console.WriteLine("На счету {0} сумма: {1:C}", firm1.AccNumber,
firm1.CurrentSum);

firm1.Withdraw(5000);
Console.WriteLine("На счету {0} сумма: {1:C}", firm1.AccNumber,
firm1.CurrentSum);

Console.ReadKey();
```

Результат работы консольного приложения показан на рисунке 8.

```
Реализация интерфейсов
Клиенты банка:
Информация о клиенте:
Имя: Иванов
Фамилия: Иван
Пол: мужской
Дата рождения: 12.09.1987
Номер счета: 452115125
Текущая сумма: 200000
Процент начислений: 150
Информация о клиенте:
Имя: Петров
Фамилия: Матвей
Пол: мужской
Дата рождения: 25.04.1995
Номер счета: 548358348
Текущая сумма: 10000
Процент начислений: 50
Информация о клиенте:
Имя: Сидоров
Фамилия: Сергей
Пол: мужской
Дата рождения: 25.01.2000
Номер счета: 383868322
Текущая сумма: 360000
Процент начислений: 250
На счету 452115125 сумма: 235 000,00 Р
На счету 548358348 сумма: 3 000,00 Р
Данные о фирме:
Название: ООО Кошкин дом
Директор: Иван
Капитал: 2500000
Номер счета: 235235135
Сума на счету: 1 200 000,00 Р
Капитал фирмы ООО Кошкин дом составляет 2 200 000,00 Р
Капитал фирмы ООО Кошкин дом составляет 3 200 000,00 Р
На счету 235235135 сумма: 1 700 000,00 Р
На счету 235235135 сумма: 1 695 000,00 Р
Для продолжения нажмите любую клавишу . . .
```

Рисунок 8 – результат работы системы

Часть 2 – реализация стандартных интерфейсов

Стандартные интерфейсы .NET

В библиотеке классов .NET определено множество стандартных интерфейсов, задающих желаемое поведение объектов.

Например, интерфейс **Comparable** задаёт метод сравнения объектов по критерию больше или меньше, что позволяет выполнять их сортировку. Интерфейсы **Enumerable** и **IEnumerator** дают возможность просматривать содержимое объекта с помощью цикла `foreach`. Реализация интерфейса **Cloneable** позволяет клонировать объекты.

В интерфейсе **Comparable** определен единственный метод **CompareTo(object obj)**, возвращающий результат сравнения двух объектов – текущего и переданного ему в качестве параметра **obj**.

Метод должен возвращать:

- 0, если текущий объект и параметр равны;

- отрицательное число, если текущий объект меньше параметра;
- положительное число, если текущий объект больше параметра.

Интерфейс **IEnumerable** определяет метод **GetEnumerator**, возвращающий объект типа **IEnumerator**, который можно использовать для просмотра элементов объекта.

Интерфейс **IEnumerator** задаёт три функциональных элемента:

- свойство **Current**, возвращающее текущий элемент объекта;
- метод **MoveNext**, передвигающий к следующему элементу объекта;
- метод **Reset**, устанавливающий в начало просмотра.

Для упрощения перебора в объекте используются средства, называемые итераторами.

Итератор представляет собой блок кода, задающий последовательность перебора элементов объекта.

Преимущество использования итераторов заключается в том, что для одного и того же класса можно задать различный порядок перебора элементов.

В теле итератора должен присутствовать оператор **yield**, имеющий следующий синтаксис:

```
yield return выражение;
```

При выполнении итератора автоматически создаётся контейнер, в который добавляется элемент при каждом выполнении оператора **yield**. Порядок выполнения оператора **yield** определяет порядок перечислимости элементов контейнера.

Операции **is** и **as**.

При работе с объектом часто требуется проверить, поддерживает ли объект определённый интерфейс. Проверка может быть выполнена с помощью бинарной операции **is**.

Операция **is** определяет, совместим ли тип объекта, находящегося слева от ключевого слова **is**, с типом, заданным справа. Результат операции равен

true, если объект можно преобразовать к заданному типу, и **false** в противном случае.

Недостатком использования операции **is** является то, что преобразование фактически выполняется дважды: при проверке и при собственно преобразовании.

Операция **as** выполняет преобразование к заданному типу, а если это невозможно, то формирует результат **null**.

Пример

Требуется разработать проект библиотеки классов, содержащий следующие классы:

- **Car** – представляет автомобили; реализует интерфейс **Comparable**;
- **Garage** – представляет автомобильные гаражи; реализует интерфейс **INumerable**.

Сравнение экземпляров класса **Car** с помощью метода **CompareTo()** будем производить по значению свойства **Probeg** (пробег автомобиля, км).

В классе **Garage** должен быть реализован метод **GetEnumerator()**, возвращающий интерфейсный объект типа **IEnumerator**, который позволяет просматривать элементы массива **carLot** (состоит из экземпляров класса **Car**).

Дополнительно добавим в класс **Garage** метод **CarsIterator(bool reverse)**, который возвращает элементы массива в прямом или обратном порядке в зависимости от значения параметра **reverse**.

Исходный код класса **Car**:

```
public class Car : IComparable
{
    public string Mark { get; set; }
    public string Number { get; set; }
    public int Probeg { get; set; }

    public Car(string mark, string number, int probeg)
    {
        Mark = mark;
        Number = number;
        Probeg = probeg;
    }

    public override string ToString()
    {
```

```

        return $"Данные об автомобиле: \n" +
            $"Марка: {Mark}\n" +
            $"Номер: {Number}\n" +
            $"Пробег: {Probeg} км.";
    }

    public int CompareTo(object obj)
    {
        Car car = obj as Car;

        if(car != null)
        {
            if (this.Probeg > car.Probeg) return 1;
            else if (this.Probeg < car.Probeg) return -1;
            else return 0;
        }
        else
        {
            throw new ArgumentException("Сравниваемый объект не принадлежит
классу Car!");
        }
    }
}

```

Исходный класс класса **Garage**:

```

class Garage : IEnumerable
{
    Car[] carArray;

    public Garage(Car[] cars)
    {
        carArray = cars;
    }

    public IEnumerator GetEnumerator()
    {
        return carArray.GetEnumerator();
    }

    public IEnumerable CarsIterator(bool reverse)
    {
        if(reverse)
        {
            for (int i = carArray.Length - 1; i >= 0; i--)
            {
                yield return carArray[i];
            }
        }
        else
        {
            foreach(Car c in carArray)
            {
                yield return c;
            }
        }
    }
}

```

Схема классов для полученной библиотеки классов показана на рисунке

9.

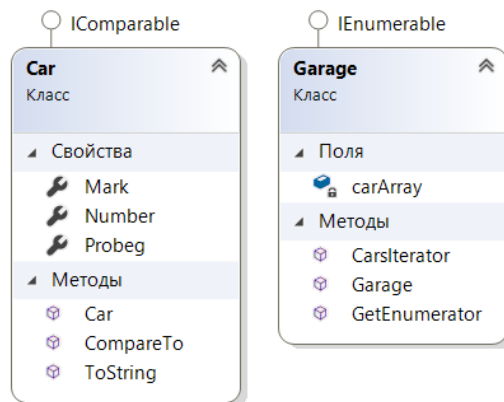


Рисунок 9 – диаграмма классов

Для демонстрации работы с классами, реализующими стандартные интерфейсы, добавим в решение проект консольного приложения.

Исходный код класса Program консольного приложения

```
Console.Title = "Реализация стандартных интерфейсов";

Car car1 = new Car("VAZ", "p432кв33", 78000);
Car car2 = new Car("Volvo", "a912щу33", 178000);
Car car3 = new Car("Opel", "л099ув52", 5400);

Car[] carLot = new Car[] { car1, car2, car3 };

Array.Sort(carLot);

Console.WriteLine("Автомобили, отсортированные по пробегу:");
foreach(Car c in carLot)
    Console.WriteLine(c.ToString());

Garage garage = new Garage(carLot);

Console.WriteLine("Все автомобили в гараже:");
foreach(Car c in garage)
    Console.WriteLine(c.ToString());

Console.WriteLine("Автомобили, отсортированные в обратном порядке:");
foreach(Car c in garage.CarsIterator(true))
    Console.WriteLine(c.ToString());

Console.ReadKey();
```

Задание на лабораторную работу

Задание 1. Создать библиотеку с указанными классами и интерфейсами (табл. 1). Реализовать интерфейсы в подходящих для них классах. Разработать проект консольного приложения для работы с полученными классами.

Таблица 1

№ вар.	Классы	Интерфейсы
1,9,17	<i>Здание</i> (адрес, площадь, число этажей, дата постройки, получить данные). <i>Автомобиль</i> (регистрационный номер, марка, дата выпуска, пробег, получить данные).	<i>Ремонтируемый</i> (износ, определить стоимость ремонта, ремонтировать). <i>Заправляемый топливом</i> (текущий уровень топлива, объем топливного бака, заправить топливом).
2,10,18	<i>Книга</i> (название, автор, издательство, год выпуска, число страниц, получить данные). <i>Электрочайник</i> (модель, цвет, объем, мощность, получить данные).	<i>Товар</i> (цена, скидка, производитель, дата выпуска, определить цену с учетом скидки). <i>Потребляющий электроэнергию</i> (напряжение питания, сила тока, подключить к сети, отключить от сети, определить затраты энергии).
3,11,19	<i>Квартира</i> (адрес дома, номер, этаж, площадь, число комнат, получить данные). <i>Локомотив</i> (модель, год выпуска, мощность, максимальная скорость, получить данные).	<i>Ремонтируемый</i> (износ, определить стоимость ремонта, ремонтировать). <i>Перемещаемый в пространстве</i> (текущие координаты, за данные координаты, двигаться к цели).
4,12,20	<i>Легковой автомобиль</i> (марка, тип кузова, цвет, расход топлива, получить данные). <i>Обувь</i> (название, сезон, материал, цвет, размер, получить данные).	<i>Товар</i> (цена, скидка, производитель, дата выпуска, определить цену с учетом скидки). <i>Очищаемый</i> (степень загрязнения, очистить, определить время очистки).

5,13,21	<p>Посылка (код, отправитель, адрес получателя, вес, получить данные).</p> <p>Станок (тип, модель, мощность привода, точность, получить данные).</p>	<p>Перемещаемый в пространстве (текущие координаты, за данные координаты, перемещать к цели).</p> <p>Потребляющий электроэнергию (напряжение питания, сила тока, подключить к сети, отключить от сети, определить затраты энергии).</p>
6,14,22	<p>Дорога (длина, ширина, материал полотна, получить данные).</p> <p>Обувь (название, сезон, материал, цвет, размер, получить данные).</p>	<p>Ремонтируемый (износ, определить стоимость ремонта, отремонтировать).</p> <p>Товар (цена, скидка, производитель, дата выпуска, определить цену с учётом скидки).</p>
7,15,23	<p>Принтер (модель, скорость печати, объём картриджа, максимальное число листов в лотке, получить данные).</p> <p>Квартира (адрес дома, номер, этаж, площадь, число комнат, получить данные).</p>	<p>Товар (цена, скидка, производитель, дата выпуска, определить цену с учётом скидки).</p> <p>Очищаемый (степень загрязнения, очистить, определить время очистки).</p>
8,16,24	<p>Грузовой контейнер (код, перевозчик, получатель груза, вес груза, получить данные).</p> <p>Пассажирский самолёт (модель, авиакомпания, макс. число пассажиров, крейсерская скорость).</p>	<p>Перемещаемый в пространстве (текущие координаты, за данные координаты, перемещать к цели).</p> <p>Заправляемый топливом (текущий уровень топлива, объём топливного бака, заправить топливом).</p>

Задание 2. Создать заданные классы (табл. 2), реализующие стандартные интерфейсы **Comparable** и **Enumerable**. Разработать проект консольного приложения для работы с экземплярами созданных классов.

№ вар.	Comparable	Enumerable
1,13	Квартира. Номер, этаж, площадь, число комнат.	Многоквартирный дом. Улица, номер, квартиры.

2,14	Заказ на перевозку груза. Номер, дата, адрес доставки, вес груза, стоимость перевозки.	Транспортная компания. Название, телефон, заказы.
3,15	Спортсмен. ФИО, вид спорта, дата рождения, пол, рост, вес.	Спортивная команда. Название, тренер, спортсмены.
4,16	Ноутбук. Модель, процессор, размер экрана, вес, цена.	Компьютерный магазин. Название, адрес, ноутбуки.
5,17	Студент. ФИО, группа, пол, дата рождения, средний бал.	Студенческая группа. Название, куратор, студенты.
6,18	Книга. Название, автор, цена, число страниц, год издания.	Книжный магазин. Название, адрес, книги.
7,19	Сотрудник. ФИО, пол, дата рождения, должность, зарплата.	Организация. Название, адрес, сотрудники.
8,20	Учебная дисциплина. Название, ФИО преп-ля, форма контроля, семестр, число часов.	Учебный план. Направление подготовки, про филь, дисциплины.
9,21	Кредит. Получатель, сумма, процент, дата получения, срок.	Банк. Название, адрес центрального офиса, кредиты.
10,22	Предмет обуви. Наименование, производитель, число пар, размер, цена.	Обувной магазин. Название, адрес, обувь.
11,23	Телевизор. Фирма, модель, размер экрана, вес, цена.	Магазин бытовой электроники. Название, адрес, телевизоры.
12,24	Билет на междугородный транспорт. Рейс, пункт назнач., время отправления, длительность, номер места.	Автовокзал. Город, число касс, билеты.