Лабораторная работа №2.1: Создание геоинформационного приложения

Цель работы:

- освоить технологии создания геоинформационных приложений;
- изучить основы наследования и полиморфизма на примере их реализации в языке C#.

Краткие теоретические сведения:

Механизм **наследования** выполняет три основные функции:

- обеспечивает иерархичность данных;
- позволяет унаследовать функциональность другого класса;
- позволяет реализовать полиморфизм подтипов.

Примером простого наследования может служить следующий исходный код:

```
// базовый класс (определяет базовый функционал)
class Person
{
    public string Name;
    public void Display()
    {
        Console.WriteLine(Name);
    }
}

// дочерний класс
class Student : Person
{

}
```

Базовым (или **родительским**) называется класс, от которого происходит наследование. **Дочерним** называется класс, наследуемый от базового. Имя базового класса указывается после двоеточия. Таким образом, для класса Student базовым является Person, и поэтому класс Person наследует все те же свойства, методы, поля, которые есть в классе Person. Объект класса Student может использовать поля и методы класса Person:

```
static void Main(string[] args)
{
    Person person = new Person { Name = "Tom" };
    person.Display();
    Person student = new Student { Name = "Sam" };
    student.Display();
    Console.Read();
}
```

При наследовании нередко возникает необходимость изменить в дочернем классе функционал метода, который был унаследован от базового класса. В этом случае класс-наследник может переопределять методы и свойства базового класса.

Те методы и свойства, которые необходимо сделать доступными для переопределения, в базовом классе помечаются модификатором **virtual**. Такие методы и свойства называют виртуальными. А чтобы переопределить метод в дочернем классе, этот метод определяется с модификатором **override**. Переопределенный метод в дочернем классе должен иметь тот же набор параметров, что и виртуальный метод в базовом классе. Данный механизм называется **полиморфизм подтипов**.

Рассмотрим пример с классами Person и Student:

```
class Person
{
    public string FirstName { get; set; }
    public string LastName { get; set; }
    public virtual void Display()
```

```
    {
        Console.WriteLine($"{FirstName} {LastName}");
    }
}

class Student : Person
{
    public string Group { get; set; }
    public override void Display()
    {
        Console.WriteLine($"{FirstName} {LastName} гр. {Group}");
    }
}
```

Помимо унаследованных свойств и методов, класс Student содержит свойство `Group`, означающее группу, в которой студент учится. В классе Student можно переопределить реализацию метода `Display`, чтобы он выводил еще и группу. Для этого базовом классе метод `Display` помечается модификатором **virtual,** а в дочернем классе добавляется метод с тем же именем, но с другой реализацией и модификатором **override.**

Также при переопределении методов можно использовать ключевое слово **base,** в частности, для вызова базового метода внутри дочернего. В нашем случае вызов base.Display(); будет обращением к методу Display() в классе Person:

```
class Student : Person
{
    public string Group { get; set; }
    public override void Display()
    {
        base.Display();
        Console.WriteLine($"гр. {Group}");
    }
}
```

Несмотря на то, что синтаксически вызывается один и тот же метод, фактически, в зависимости от типа объекта, будет вызываться одна из реализаций метода Display():

```
static void Main(string[] args)
{
    Person person = new Person { FirstName = "Tom", LastName = "Hanks" };
    person.Display(); // вызовется базовая реализация из класса Person
    Person student = new Student { FirstName = "Tom", LastName = "Hardy", Group = "RI-78" };
    student.Display(); // вызовется дочерняя реализация из класса Student
    Console.Read();
}
```

Абстрактные классы и интерфейсы:

Абстрактный класс - это класс, содержащий абстрактные (не имеющие реализации) методы. Наглядным примером абстрактного класса является фигура. В реальности не существует геометрической фигуры как таковой. Есть круг, прямоугольник, квадрат, но просто фигуры нет. Однако же и круг, и прямоугольник имеют что-то общее и являются фигурами.

Рассмотрим пример вычисления периметра и площади фигуры. Для абстрактной фигуры это не представляется возможным, однако для конкретных фигур (круг, прямоугольник, квадрат) существуют соответствующие формулы.

```
// абстрактный класс фигуры
abstract class Figure
{
    // координаты фигуры
    private double x;
    private double y;

    // конструктор
```

```
    public Figure(double x, double y)
    {
        this.x = x;
        this.y = y;
    }

    // абстрактный метод получения периметра
    public abstract double getPerimeter();
    // абстрактный метод получения площади
    public abstract double getArea();
}

// производный класс прямоугольника
class Rectangle : Figure
{
    // ширина и высота прямоугольника
    private double width;
    private double height;
    // конструктор с обращением к конструктору базового класса
    public Rectangle(double x, double y, double width, double height) : base(x, y)
    {
        this.width = width;
        this.height = height;
    }

    // реализация метода получения периметра прямоугольника
    public override double getPerimeter()
    {
        return 2 * (width + height);
    }

    // реализация метода получения площади прямоугольника
    public override double getArea()
    {
        return width * height;
    }
}
```

В абстрактном классе Figure объявлены (но не реализованы) абстрактные методы вычисления периметра и площади фигуры. Данные методы реализованы в производном классе прямоугольника с помощью соответствующих формул. Для остальных классов (например, класса круга) также добавляется соответствующая реализация абстрактных методов.

Поскольку один или более методов подобного класса не имеет реализации, невозможно создать объект этого класса. Однако ссылка на абстрактный класс может указывать на любой объект дочернего типа:

```
Figure figure = new Figure(0, 0); // некорректно, абстрактный класс не может иметь объектов
Figure rectangle = new Rectangle(0, 0, 10, 20); // корректно
```

Интерфейс является частным случаем абстрактного класса и может содержать только абстрактные методы и свойства. Преимуществом интерфейсов является то, что они поддерживают множественное наследование (имплементацию).

```
interface Drawable
{
    void Draw();
}

// класс прямоугольника наследует и класс, и интерфейс
class Rectangle : Figure, Drawable
{
    // ширина и высота прямоугольника
    private double width;
    private double height;
```

```
    // конструктор с обращением к конструктору базового класса
    public Rectangle(double x, double y, double width, double height) : base(x, y)
    {
        this.width = width;
        this.height = height;
    }

    // реализация метода получения периметра прямоугольника
    public override double getPerimeter()
    {
        return 2 * (width + height);
    }

    // реализация метода получения площади прямоугольника
    public override double getArea()
    {
        return width * height;
    }

    // реализация метода из интерфейса Drawable
    public void Draw()
    {
        // TODO : action
    }
}
```

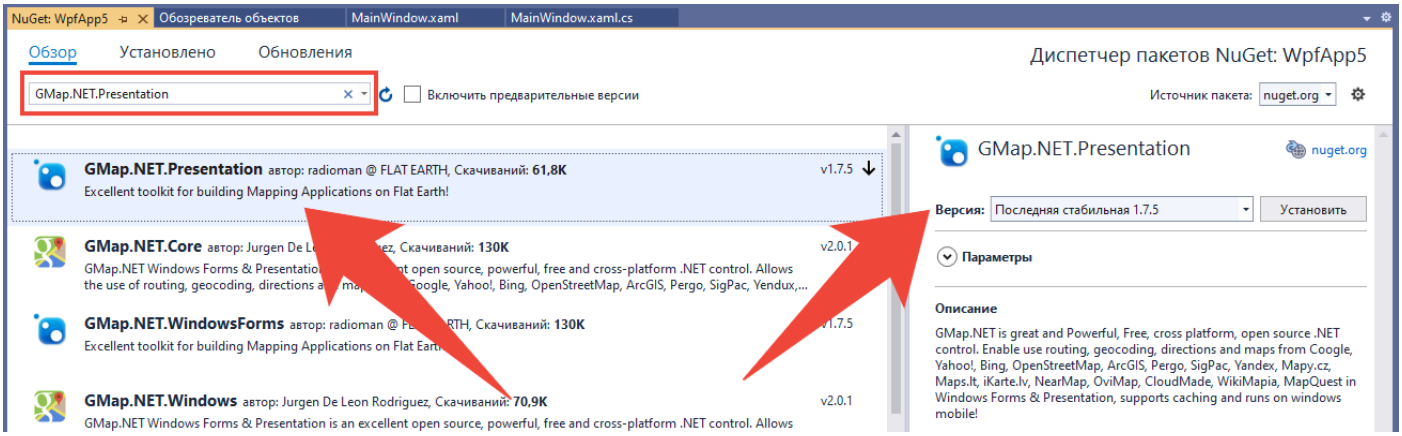
Использование абстрактных классов и интерфейсов гарантирует то, что при добавлении нового класса в уже существующую иерархию он будет поддерживать минимально необходимый для работы программного средства функционал.

Подключение геоинформационных библиотек к проекту и инициализация карты

Библиотека GMap.NET содержит богатый функционал для работы с картами:

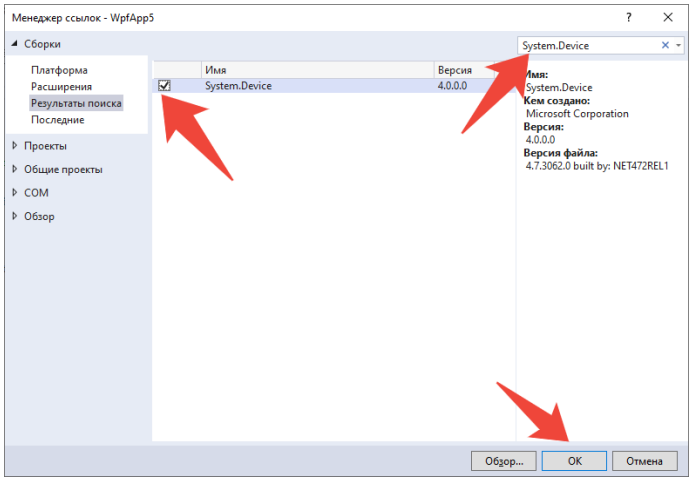
- загрузка карт из различных источников;
- визуализация карт;
- геокодирование, построение маршрутов и др.

GMap.NET подключается к проекту через диспетчер пакетов NuGet:



Пространство имен System.Device.Location позволяет разработчикам приложений удобным образом получать местоположение компьютера из различных источников (GPS, триангуляция WI-FI и базовых станций сотовых сетей), а также производить вычисления с географическими координатами.

Пространство имен System.Device.Location является частью библиотеки System.Device и входит в состав .NET Framework 4 и выше. Для подключения System.Device воспользуйтесь менеджером ссылок (Главное меню -> Проект -> Добавить ссылку…):



Отображение карты и взаимодействие с ней осуществляется с помощью элемента управления GMapControl. Для добавления GMapControl в панель элементов (по умолчанию он там отсутствует) необходимо выполнить следующие действия:

1. в контекстном меню панели элементов выбрать пункт «Выбрать элементы»;
2. в появившемся диалоговом окне (во вкладке «Компоненты WPF») нажать на кнопку «Обзор»;
3. в диалоговом окне найти файл GMap.NET.WindowsPresentation.dll, в котором находится элемент управления GMapControl. Как правило, он находится в следующей директории: путь_к_проекту\packages\GMap.NET.Presentation.1.7.5\lib\net40;
4. подтвердить выбор, нажав кнопку «ОК».

После добавления элемента управления GMapControl в разметку окна необходимо определить следующие атрибуты: Name (имя элемента управления для обращения в коде) и Loaded (обработчик события готовности элемента управления):

```
<WindowsPresentation:GMapControl Name="Map" Loaded="MapLoaded"/>
```

Для работы с картой используются следующие пространства имен:

```
using GMap.NET;
using GMap.NET.MapProviders;
using GMap.NET.WindowsPresentation;
using System.Device.Location;
```

В классе окна необходимо реализовать обработчик события готовности карты и добавить в него код для настройки карты:

```
private void MapLoaded(object sender, RoutedEventArgs e)
{
    // настройка доступа к данным
    GMaps.Instance.Mode = AccessMode.ServerAndCache;

    // установка провайдера карт
    Map.MapProvider = BingMapProvider.Instance;

    // установка зума карты
    Map.MinZoom = 2;
    Map.MaxZoom = 17;
    Map.Zoom = 15;

    // установка фокуса карты
    Map.Position = new PointLatLng(55.012823, 82.950359);

    // настройка взаимодействия с картой
    Map.MouseWheelZoomType = MouseWheelZoomType.MousePositionAndCenter;
    Map.CanDragMap = true;
    Map.DragButton = MouseButton.Left;
}
```

Инициализация маркеров

На карте, помимо встроенных объектов, могут отображаться пользовательские объекты – так называемые маркеры. Маркером можно обозначить определенное местоположение, область или движущийся объект.

Как правило, положение маркеров на карте задается с помощью одной или нескольких точек, заданных в географических координатах. Для работы с географическими координатами библиотека GMap.NET предоставляет класс `PointLatLng`. Конструктор класса принимает два значения – широту и долготу:

```
PointLatLng point = new PointLatLng(55.016511, 82.946152);
```

После определения координат объекта, необходимо создать объект-маркер и задать его визуальные параметры. Например, маркер для автомобиля можно определить следующим образом:

```
GMapMarker marker = new GMapMarker(point)
{
    Shape = new Image
    {
        Width = 32, // ширина маркера
        Height = 32, // высота маркера
        ToolTip = "Honda CR-V", // всплывающая подсказка
        Source = new BitmapImage(new Uri("pack://application:,,,/Resources/car.png")) // картинка
    }
};
```

Отображение области на карте можно выполнить в виде полигона. Для этого необходимо определить список координат полигона, а так же цвет обводки и заливки:

```
// координаты точек замкнутой области (полигона)
List<PointLatLng> points = new PointLatLng[] {
                new PointLatLng(55.016351, 82.950650),
                new PointLatLng(55.017021, 82.951484),
                new PointLatLng(55.015795, 82.954526),
                new PointLatLng(55.015129, 82.953586) }.ToList();

GMapMarker marker = new GMapPolygon(points)
{
    Shape = new Path
    {
        Stroke = Brushes.Black, // стиль обводки
        Fill = Brushes. Violet, // стиль заливки
        Opacity = 0.7 // прозрачность
    }
};
```

Похожим образом можно создать маркер для некоторого маршрута в виде линии:

```
// координаты точек маршрута
List<PointLatLng> points = new PointLatLng[] {
                new PointLatLng(55.010637, 82.938550),
                new PointLatLng(55.012421, 82.940781),
                new PointLatLng(55.014613, 82.943497),
                new PointLatLng(55.016214, 82.945469) }.ToList();

GMapMarker Marker = new GMapRoute(points)
{
    Shape = new Path()
    {
        Stroke = Brushes.DarkBlue, // цвет обводки
        Fill = Brushes.DarkBlue, // цвет заливки
        StrokeThickness = 4 // толщина обводки
    }
};
```

Добавление маркеров на карту осуществляется следующим образом:

```
Map.Markers.Add(marker);
```

Пример карты с маркерами:

Взаимодействие с картой

Помимо всего прочего, пользователь может взаимодействовать с картой различными способами: менять зум и фокус карты, измерять расстояние между точками на карте, искать информацию о различных объектах и т.д. Типичная задача – определение географических координат точки, на которую нажал пользователь. Библиотека GMap.NET предоставляет обработчик нажатия левой кнопки мыши на карту MouseLeftButtonDown. Зарегистрировать его можно следующим образом:

```
<WindowsPresentation:GMapControl Name="Map" Loaded="MapLoaded" MouseLeftButtonDown="Map_MouseLeftButtonDown"/>
```

а затем добавить код обработчика в классе окна для вычисления географических координат:

```
private void Map_MouseLeftButtonDown (object sender, MouseButtonEventArgs e)
{
    PointLatLng point = Map.FromLocalToLatLng((int)e.GetPosition(Map).X, (int)e.GetPosition(Map).Y);
}
```

Для решения аналогичных задач можно воспользоваться следующими обработчиками, работающими по тому же принципу:

- MouseRightButtonDown (нажатие правой кнопкой мыши);
- MouseDoubleClick (двойное нажатие мыши);
- MouseMove (перемещение курсора);
- MouseEnter (наведение курсора на карту);
- и др.

Часто требуется получить расстояние между двумя точками. Ниже приведен пример вычисления расстояния между двумя точками по их координатам:

```
// точки в формате GMap.NET
PointLatLng p1 = new PointLatLng(55.015104, 82.948034);
PointLatLng p2 = new PointLatLng(55.018812, 82.940049);
// точки в формате System.Device.Location
GeoCoordinate c1 = new GeoCoordinate(p1.Lat, p2.Lng);
GeoCoordinate c2 = new GeoCoordinate(p2.Lat, p2.Lng);

// вычисление расстояния между точками в метрах
double distance = c1.GetDistanceTo(c2);
```

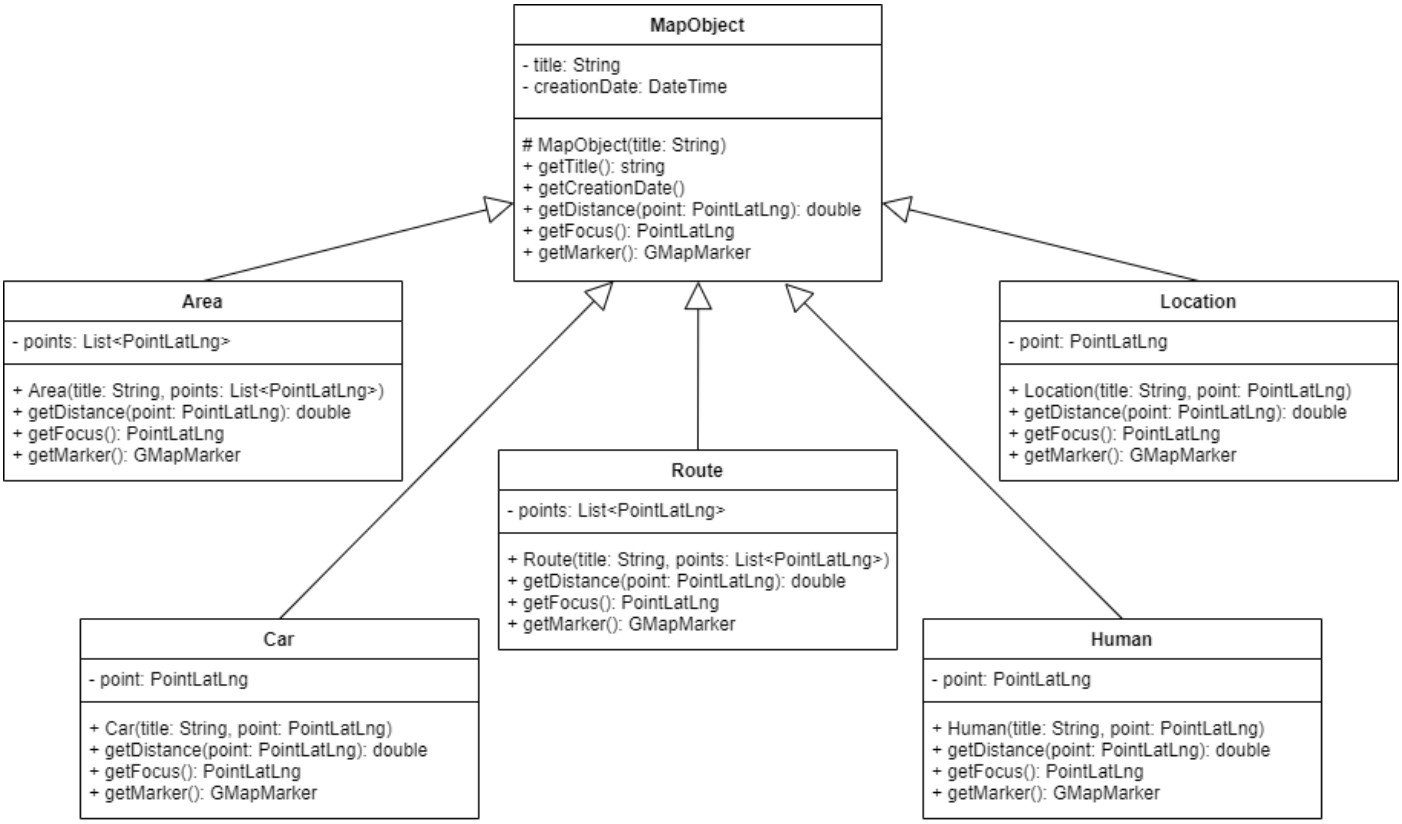
Минимальное расстояние от точки до объекта, состоящего из множества точек (маршрута, области и т.п.) можно получить различными способами, например:

– приближенно, рассчитав расстояние до каждой точки отдельно и определить среди них минимальное значение;
– с помощью формулы Haversine (пример реализации на языке C#).

Задание:

Разработать геоинформационное приложение на базе библиотеки GMap.NET, а также:

1. Реализовать отображение пользовательских объектов в виде маркеров на карте: местоположение, область, маршрут, человек, автомобиль. Для каждого вида объектов необходимо создать отдельный класс, реализующий создание маркера и его отображение. Иерархия и структура классов представлена на UML-диаграмме:



Все экземпляры данных классов должны храниться в одной коллекции (в списке или словаре) и использовать общие методы, определенные в базовом классе MapObject.

2. Реализовать 2 режима работы курсора:
   - режим создания объектов (позволяет сохранять координаты точек при нажатии на карту, см. пункт 3);
   - режим поиска ближайших объектов (при нажатии на карту выводит список объектов, отсортированных по расстоянию, см. пункт 5).
3. Реализовать возможность добавления объектов на карту по точкам, заданным пользователем на карте (для местоположения, автомобиля и человека достаточно одной точки, для маршрута – минимум две точки, для области – минимум три).
4. Реализовать возможность поиска объектов по названию (если подходит несколько объектов, выводить их списком, а при выделении – показывать в фокусе).
5. Для каждого из объектов реализовать метод для определения расстояния до произвольной точки. С помощью этого метода реализовать поиск ближайших объектов.

Ниже приведет один из вариантов пользовательского интерфейса приложения.

Контрольные вопросы:

1) Что такое наследование? Как осуществляется наследование в языке C#?
2) Что такое полиморфизм? Как осуществляется полиморфизм подтипов в языке C#?
3) Назовите главные особенности абстрактных классов.
4) Какие методы можно переопределять?
5) Что такое маркер? Покажите пример отображения маркера на карте.
6) Опишите класс `PointLatLng`, приведите примеры использования.
7) С помощью какого метода можно получить географические координаты из координат курсора?

Список литературы:

1) Герберт Шилдт "C# 4.0: полное руководство"
2) Эндрю Троелсен "Язык программирования C# 5.0 и платформа .NET 4.5"
3) Руководство по программированию на C# https://docs.microsoft.com/ruru/dotnet/csharp/programming-guide/index
4) Полное руководство по языку программирования C# 7.0 и платформе .NET 4.7 https://metanit.com/sharp/tutorial/
5) Руководство по WPF https://metanit.com/sharp/wpf/
6) C# 5.0 и платформа .NET 4.5 http://professorweb.ru/my/csharp/charp_theory/level1/infocsharp.php
7) https://github.com/Microsoft/WPF-Samples