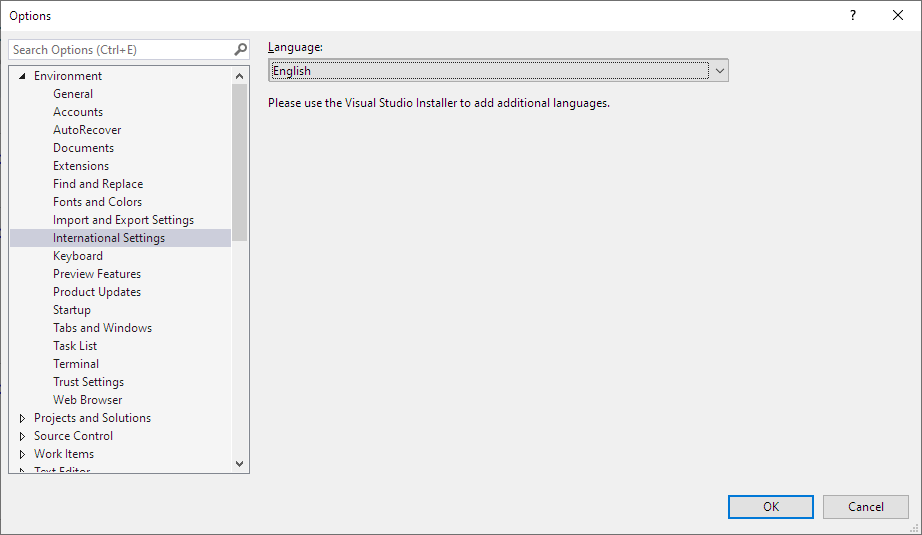
# Сериализация в .NET

**Цель лабораторной работы:** Получение практических навыков хранения набора объектов, межязыковой интеграции, использование нескольких сборок и различных форматов социализации объектов на платформе .NET.

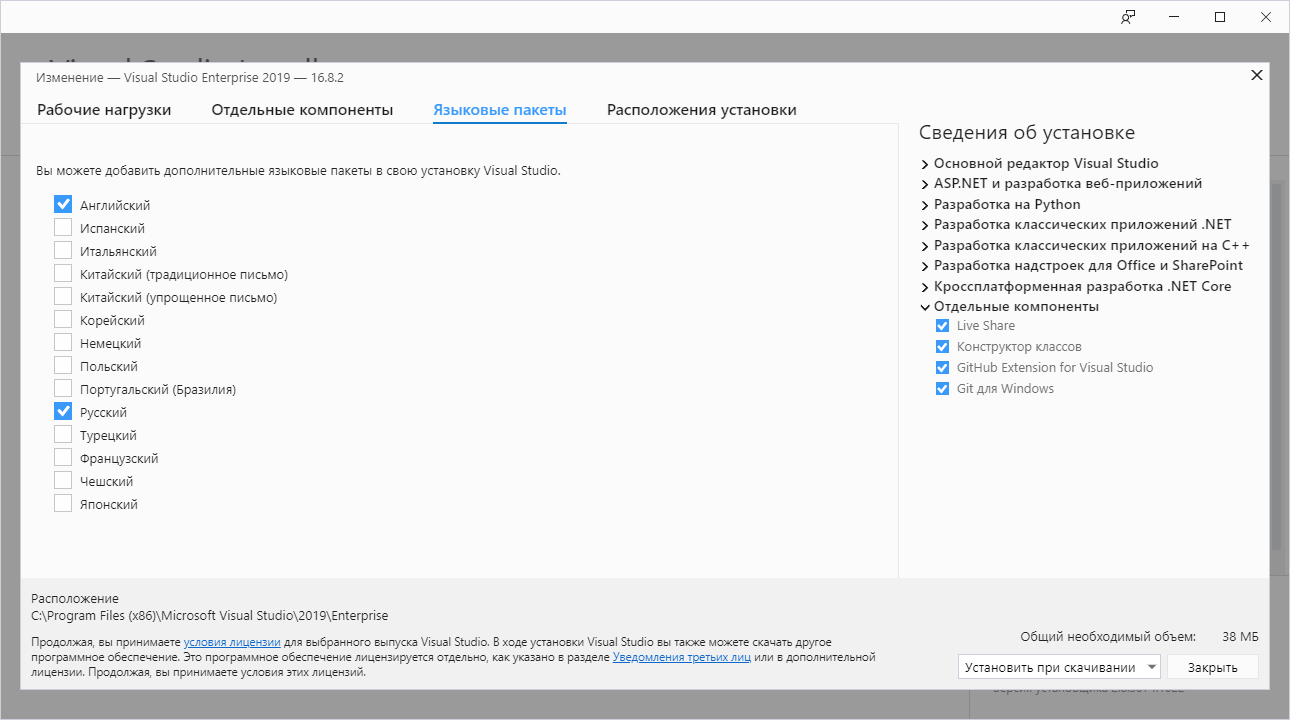
**Постановка задачи:** Разработать на платформе .NET приложение, позволяющее выполнять сериализацию и десериализацию набора объектов классов, представляющих точки в двумерном и трехмерном пространстве. Класс точки должен предоставить методы вычисления расстояния от точки до начала координат, строкового представления объекта и возможность сравнения точек между собой.

**Необходимое программное обеспечение:** Visual Studio 2019 Community

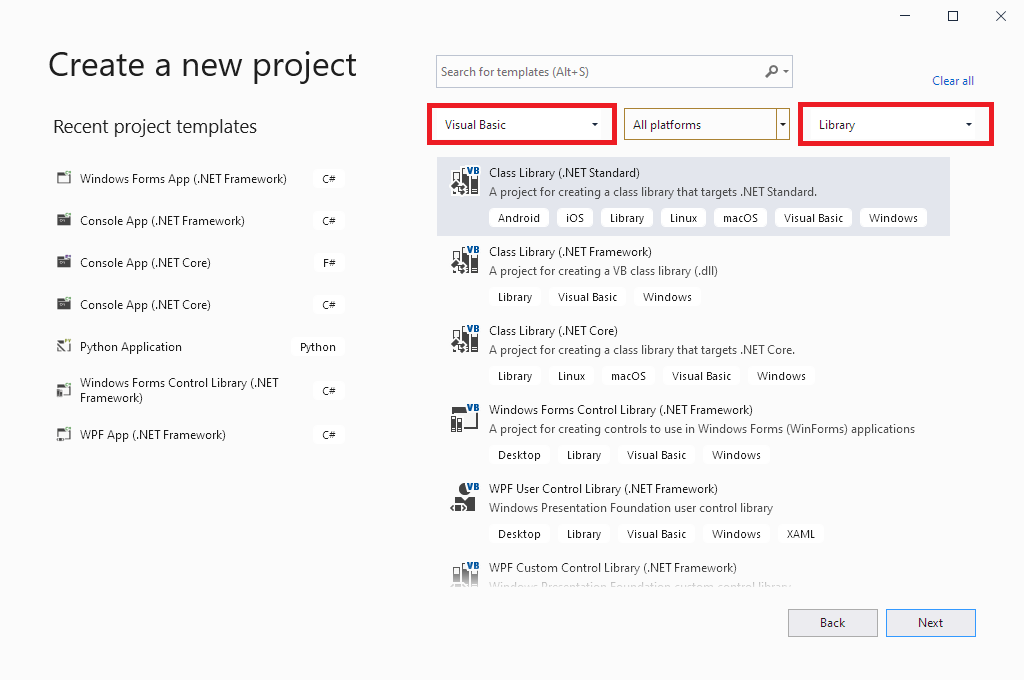
Рекомендуется переключить интерфейс среды на английский язык с помощью команды **Tools|Options…** раздел **Environment\International Settings**. При отсутствии английского языка в списке необходимо запустить программу установку «Visual Studio Installer» и на вкладке **Языкове пакеты** выбрать необходимый язык.



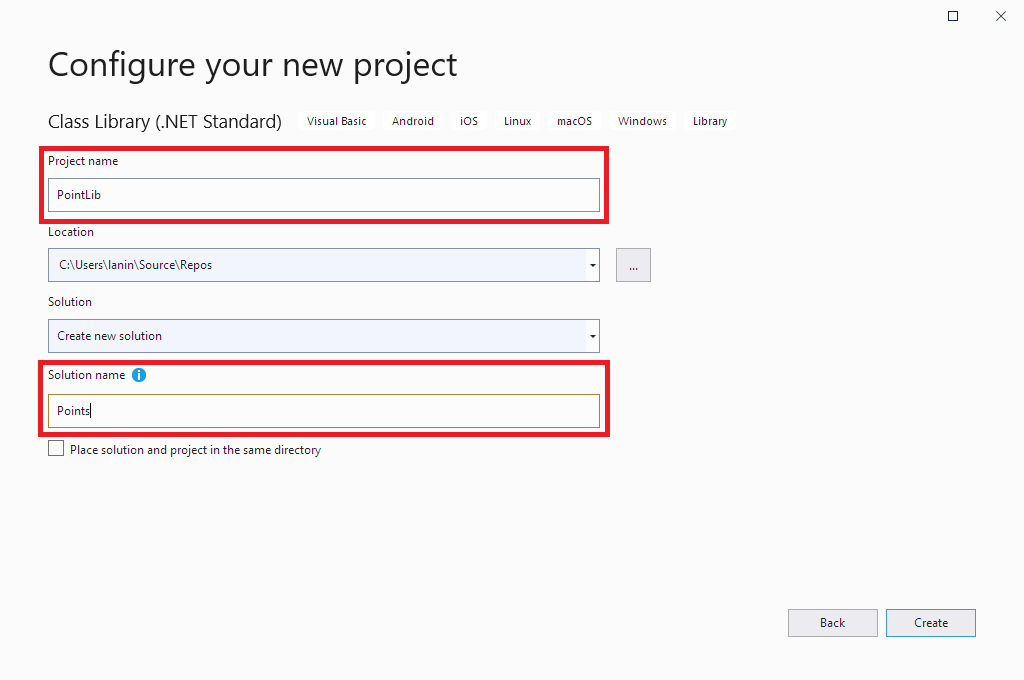
1. **Настройка языка интерфейса среды разработки Visual Studio**



1. **Выбор доступных языков интерфейса среды Visual Studio**
2. Разработка сборки на VB.NET   
   для создания класса
3. Запустите среду разработки Visual Studio. Создайте новый проект на языке Visual Basic, для этого выберите команду *File|New|Project*., в качестве типа проекта укажите *Class Library (.NET Standard)* (рис. 1). Назовите новый проект *PointLib,* а в поле **Solution Name** введите имя *Lab* (рис. 2).



1. **Диалоговое окно добавления нового проекта**



1. **Диалоговое окно добавления нового проекта: выбор названия проекта и решения**
2. В новом проекте автоматически создается исходный файл *Сlass1.vb*. Измените в этом файле имя класса *Class1* на *Point*.
3. Добавьте в определение класса свойства *X* и *Y* с типом *Integer*, и также два конструктора.

Public Class Point

Public Property X As Integer

Public Property Y As Integer

Public Sub New()

X = 0

Y = 0

End Sub

Public Sub New(ByVal x As Integer, ByVal y As Integer)

Me.X = x

Me.Y = y

End Sub

End Class

1. Добавьте метод *Metric()***,** возвращающий расстояние от точки до центра координат. Пометьте метод как *Public Overridable*– таким образом, его можно будет переопределить в классе, порожденном от *Point*.Для вычисления квадратного корня используйте статический метод *Sqrt*объекта *System.Math*.

Public Overridable Function Metric() As Double

Metric = Math.Sqrt((X ^ 2 + Y ^ 2))

End Function

1. Любой класс в среде .NETнеявно унаследован от базового класса *Object*.В этом классе определен метод *ТoString()*, возвращающий текстовую строку, содержащую имя класса. Метод *ТoString()*используется многими стандартными функциями (например, *Console.WriteLine()*) для получения строкового представления объекта. Переопределите этот метод так, чтобы он возвращал строку, содержащую координаты точки.

Public Overrides Function ToString() As String

Return String.Format("({0}, {1})", X, Y)

End Function

1. Допустим, существует необходимость сравнивать между собой объекты типа *Point*. Для этих целей существует специальный интерфейс *IComparable*. Поместите его в список наследования класса *Point*:

Public Class Point

Implements IComparable

...

End Class

1. В интерфейсе *IComparable*объявлен всего один метод *CompareTo*,который принимает объект для сравнения. Определите этот метод представленным ниже образом.

Private Function CompareTo(obj As Object) As Integer Implements IComparable.CompareTo

Dim p As Point = CType(obj, Point)

Return Metric() - p.Metric()

End Function

1. Итоговое описание класса представлено ниже.

Public Class Point

Implements IComparable

Public Property X As Integer

Public Property Y As Integer

Protected Shared rnd As New Random()

Public Sub New()

X = rnd.Next(10)

Y = rnd.Next(10)

End Sub

Public Sub New(x As Integer, y As Integer)

Me.X = x

Me.Y = y

End Sub

Public Overridable Function Metric() As Double

Metric = Math.Sqrt((X ^ 2 + Y ^ 2))

End Function

Public Overrides Function ToString() As String

Return String.Format("({0}, {1})", X, Y)

End Function

Private Function CompareTo(obj As Object) As Integer Implements IComparable.CompareTo

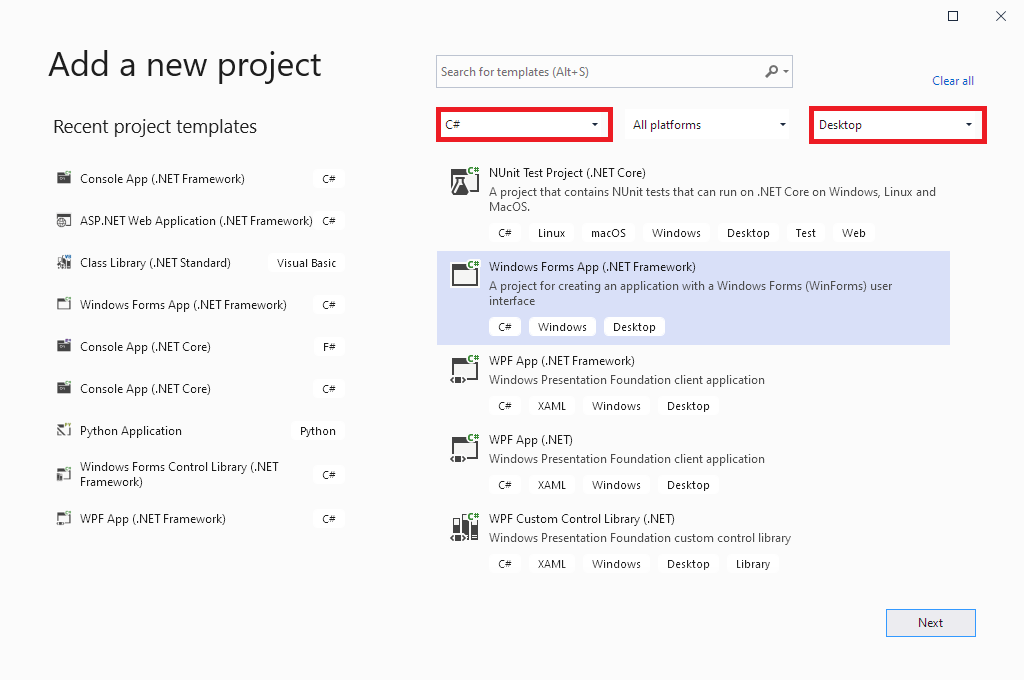
Dim p As Point = CType(obj, Point)

Return Metric() - p.Metric()

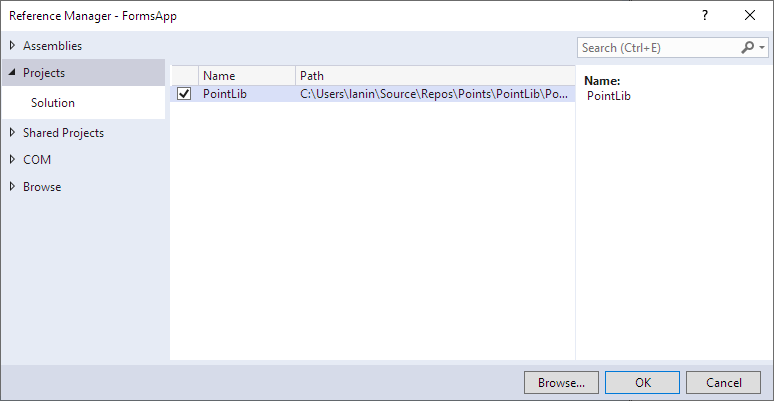
End Function

End Class

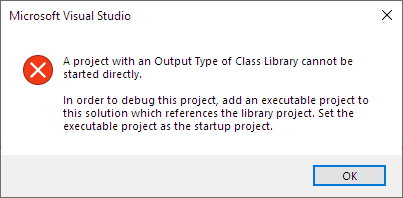
1. Выполните компиляцию проекта (**Build|Build Solution**) и убедитесь в отсутствии ошибок.
2. Использование класса Point   
   в С#-приложении
3. Добавьте в решение новый проект на языке C#, для этого выполните команду главного меню **File|Add|New project…**. Установите тип проекта **Windows Forms App (.NET Framework)**. На следующем шаге создания проекта введите его название (поле *Project Name*), например, «FormsApp».



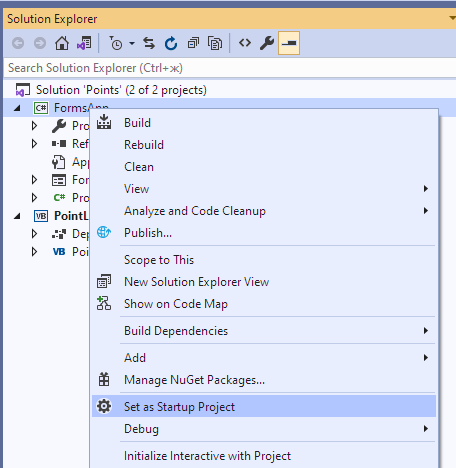
1. **Диалоговое окно создания проекта Windows Forms**
2. Для того, чтобы описанный во внешней библиотеке (сборке в терминологии .NET) класс Point был доступен для использования, необходимо установить ссылку на эту библиотеку. В главном меню выберите команду **Project|Add Reference**, перейдите в раздел Projects. Установите ссылку на проект с классом Point.



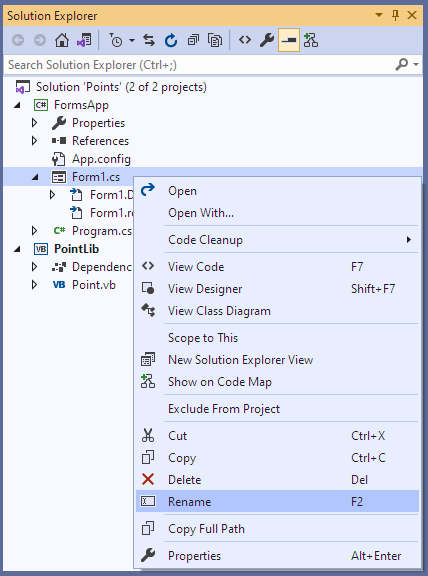
1. **Диалоговое окно добавления ссылки на проект, входящий в решение**
2. При работе с решением, содержащим несколько проектов, один из проектов является «стартовым», т.е. тем, с которого начинается выполнение программы. В нашем случае таким проектом является библиотека, которая не может быть запущена на выполнение непосредственно. При попытке запуска на выполнение (**Debug|Start Debugging** или **F5**) созданного решения будет выведено сообщение (Рис. 7).



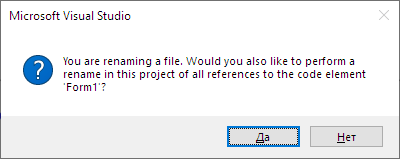
1. **Диалоговое окно с сообщением о невозможности непосредственного исполнения проекта библиотеки**
2. В окне Solution Explorer в контекстном меню проекта FormsApp выберите команду **Set as Startup Project**.



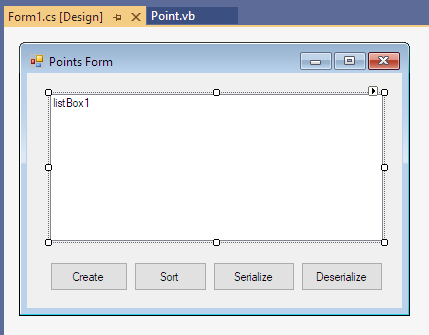
1. **Контекстное меню проекта в окне обозревателя решения**
2. Переименуйте созданную по умолчанию экранную форму *Form1* в *PointForm*. Для этого вызовите контекстное меню у файла формы и выполните команду **Rename** (рис. ). Следует также согласиться с переименованием класса формы (Рис. 9)



1. **Контекстное меню проекта в окне обозревателя решения**



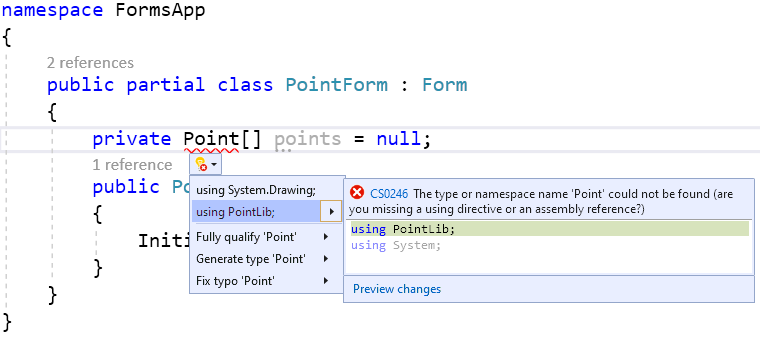
1. **Диалоговое окно подтверждения переименования класса**
2. Разместите на экранной форме командные кнопки (Button) и список (ListBox), как показано на Рис. 11



1. Экранная форма программы
2. С помощью окна **Properties** установите кнопкам надписи, задав свойство Text, переименуйте кнопки (установите свойство *Name*) *btnCreate*, *btnSort*, *btnSerialize* и *btnDeserialize* соответственно. Списку дайте имя *listBox*.
3. Откройте исходный программный код формы (**View|Code** или клавишу **F7**)ив начале файла удалите директиву *using System.Drawing;* (в данном пространстве имен также объявлен класс Point, использование которого не планируется).
4. Добавьте поле класса формы point – массив объектов *Point*.

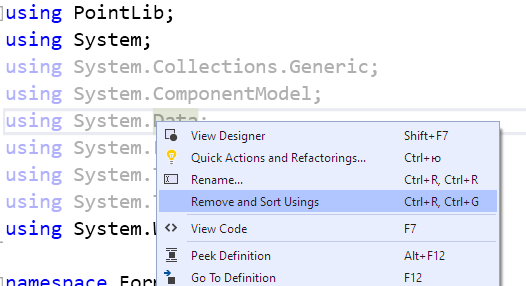
private Point[] points = null;

1. Для того, чтобы система распознала класс Point, описанный в пространстве имен PointLib, необходимо указывать полное имя класса PointLib.Point. Значительно удобнее добавить соответствующую директиву *using PointLib;* (вручную или с использованием раскрывающегося списка Рис. 12). После добавления соответствующей директивы ссылаться на тип PointLib.Point можно будет, используя короткую запись – Point, в пределах всего файла.



1. Команда вставки пространства имен

Неиспользуемые директивы лучше вовсе удалить (Рис. 13).



1. Выбор команды вставки неиспользуемых пространств имен
2. Добавьте обработчик нажатия кнопки **Create**, дважды щелкнув по ней мышью. Вставьте в созданный метод представленный ниже программный код:

private void btnCreate\_Click(object sender, EventArgs e)

{

points = new Point[5];

for (int i = 0; i < points.Length; i++)

points[i] = new Point();

listBox.DataSource = points;

}

1. Добавьте обработчик нажатия кнопки **Sort**, в котором вызовите статический метод *Sort*класса *Array*:

private void btnSort\_Click(object sender, EventArgs e)

{

if (points == null)

return;

Array.Sort(points);

listBox.DataSource = null;

listBox.DataSource = points;

}

Заметьте, что в программе несколько раз явно запрашивается выделение памяти под создаваемые объекты (вызовы функции *new*).Однако ни разу явно память не освобождается. Память автоматически освобождается системным процессом – Garbage Collector(Сборщиком Мусора).

1. Откомпилируйте, запустите и протестируйте приложение.
2. Создание класса Point3D, производного   
   от класса Point
3. Добавьте в проект *FormApp* новый класс с именем *Point3D* (**Project|Add class…**), базовым для которого будет ранее созданный класс *Point*.

using PointLib;

namespace FormsApp

{

public class Point3D:Point

{

}

}

1. Добавьте в тело класса поле *z* и два конструктора:

using PointLib;

namespace FormsApp

{

public class Point3D:Point

{

public int Z { get; set; }

public Point3D() : base()

{

Z = rnd.Next(10);

}

public Point3D(int x, int y, int z) : base(x, y)

{

Z = z;

}

}

}

1. Переопределите методы *Metric()* и *ToString()*для нового класса:

using PointLib;

using System;

namespace FormsApp

{

public class Point3D:Point

{

public int Z { get; set; }

public Point3D() : base()

{

Z = rnd.Next(10);

}

public Point3D(int x, int y, int z) : base(x, y)

{

Z = z;

}

public override double Metric()

{

return Math.Sqrt(X \* X + Y \* Y + Z \* Z);

}

public override string ToString()

{

return string.Format($"({X} , {Y}, {Z})");

}

}

}

1. Модифицируем код по созданию массива. Обратите внимание на использование тернарного оператора.

private void btnCreate\_Click(object sender, EventArgs e)

{

points = new Point[5];

var rnd = new Random();

for (int i = 0; i < points.Length; i++)

points[i] = rnd.Next(3) % 2 == 0 ? new Point(): new Point3D();

listBox.DataSource = points;

}

1. Откомпилируйте и запустите приложение. Обратите внимание, что сортирующий код и код по выводу элементов массива на экран не изменился.
2. Сохранение и восстановление   
   объектов

Объектно-ориентированное программирование вряд ли получило бы столь широкое признание, если бы программист не мог сохранить объект в текущем состоянии и восстановить его позднее. Запись объекта в поток данных называется *сериализацией* (serialization), а обратный процесс называется *десериализацией* (deserialization). Следует заметить, что сохранение и восстановление состояния объекта – более сложная и тонкая проблема, чем кажется на первый взгляд. Одна из причин этого заключается в том, что объект может содержать другие объекты. Следовательно, процесс сохранения должен поддерживать рекурсивное сохранение внутренних объектов. Более того, при этом необходимо позаботиться об отсутствии дублирования.

В .NET сохранение объектов не требует особых усилий со стороны программиста. Объекты можно сохранять в форматах XML, SOAP (Simple Object Access Protocol), основанном на языке XML, JSON и в компактном двоичном формате.

1. Для добавления возможности сохранения массива точек в файл создайте обработчик кнопки *btnSerialize* с отображением диалогового окна выбора имени файла и формата сериализации.

private void btnSerialize\_Click(object sender, EventArgs e)

{

var dlg = new SaveFileDialog();

dlg.Filter = "SOAP|\*.soap|XML|\*.xml|JSON|\*.json|Binary|\*.bin";

if (dlg.ShowDialog() == DialogResult.OK)

{

MessageBox.Show(dlg.FileName);

}

}

private void btnDeserialize\_Click(object sender, EventArgs e)

{

var dlg = new OpenFileDialog();

dlg.Filter = "SOAP|\*.soap|XML|\*.xml|JSON|\*.json|Binary|\*.bin";

if (dlg.ShowDialog() == DialogResult.OK)

{

MessageBox.Show(dlg.FileName);

}

}

1. Убедитесь работоспособности добавленного кода.
2. Реализация сериализации в бинарный формат
3. Добавьте атрибут *Serializable()* классам *Point* и *Point3D*.

<Serializable> Public Class Point

...

End Class

using PointLib;

using System;

namespace FormsApp

{

[Serializable]

public class Point3D:Point

{

...

}

}

1. Измените код кнопок сериализации и десериализации.

private void btnSerialize\_Click(object sender, EventArgs e)

{

var dlg = new SaveFileDialog();

dlg.Filter = "SOAP|\*.soap|XML|\*.xml|JSON|\*.json|Binary|\*.bin";

if (dlg.ShowDialog() != DialogResult.OK)

return;

using (var fs =

new FileStream(dlg.FileName, FileMode.Create, FileAccess.Write))

{

switch (Path.GetExtension(dlg.FileName))

{

case ".bin":

var bf = new BinaryFormatter();

bf.Serialize(fs, points);

break;

}

}

}

private void btnDeserialize\_Click(object sender, EventArgs e)

{

var dlg = new OpenFileDialog();

dlg.Filter = "SOAP|\*.soap|XML|\*.xml|JSON|\*.json|Binary|\*.bin";

if (dlg.ShowDialog() != DialogResult.OK)

return;

using (var fs =

new FileStream(dlg.FileName, FileMode.Open, FileAccess.Read))

{

switch (Path.GetExtension(dlg.FileName))

{

case ".bin":

var bf = new BinaryFormatter();

points = (Point[])bf.Deserialize(fs);

break;

}

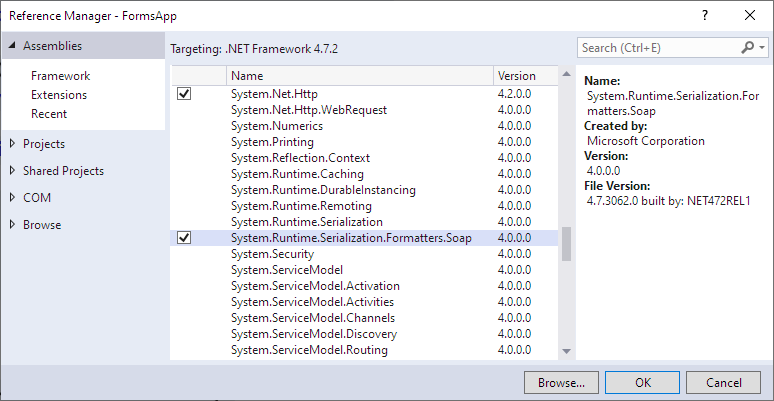
}

listBox.DataSource = null;

listBox.DataSource = points;

}

1. Проверьте работоспособность кода. Откройте полученный файл в текстовом редакторе.
2. Реализация сериализации в формат SOAP
3. Установите ссылку на сборку *System.Runtime.Serialization.Formatters.Soap* (**Project|Add reference…**).



1. Окно добавления ссылки на системную сборку
2. Измените код кнопок сериализации и десериализации как указано ниже. Проверьте работоспособность кода. Откройте полученный файл в текстовом редакторе.

private void btnSerialize\_Click(object sender, EventArgs e)

{

var dlg = new SaveFileDialog();

dlg.Filter = "SOAP|\*.soap|XML|\*.xml|JSON|\*.json|Binary|\*.bin";

if (dlg.ShowDialog() != DialogResult.OK)

return;

using (var fs =

new FileStream(dlg.FileName, FileMode.Create, FileAccess.Write))

{

switch (Path.GetExtension(dlg.FileName))

{

case ".bin":

var bf = new BinaryFormatter();

bf.Serialize(fs, points);

break;

case ".soap":

var sf = new SoapFormatter();

sf.Serialize(fs, points);

break;

}

}

}

private void btnDeserialize\_Click(object sender, EventArgs e)

{

var dlg = new OpenFileDialog();

dlg.Filter = "SOAP|\*.soap|XML|\*.xml|JSON|\*.json|Binary|\*.bin";

if (dlg.ShowDialog() != DialogResult.OK)

return;

using (var fs =

new FileStream(dlg.FileName, FileMode.Open, FileAccess.Read))

{

switch (Path.GetExtension(dlg.FileName))

{

case ".bin":

var bf = new BinaryFormatter();

points = (Point[])bf.Deserialize(fs);

break;

case ".soap":

var sf = new SoapFormatter();

points = (Point[])sf.Deserialize(fs);

break;

}

}

listBox.DataSource = null;

listBox.DataSource = points;

}

1. Реализация сериализации коллекции в XML
2. Измените код кнопок сериализации и десериализации как указано ниже. Проверьте работоспособность кода. Откройте полученный файл в текстовом редакторе.

private void btnSerialize\_Click(object sender, EventArgs e)

{

var dlg = new SaveFileDialog();

dlg.Filter = "SOAP|\*.soap|XML|\*.xml|JSON|\*.json|Binary|\*.bin";

if (dlg.ShowDialog() != DialogResult.OK)

return;

using (var fs =

new FileStream(dlg.FileName, FileMode.Create, FileAccess.Write))

{

switch (Path.GetExtension(dlg.FileName))

{

case ".bin":

var bf = new BinaryFormatter();

bf.Serialize(fs, points);

break;

case ".soap":

var sf = new SoapFormatter();

sf.Serialize(fs, points);

break;

case ".xml":

var xf = new XmlSerializer(typeof(Point[]), new[] {typeof(Point3D)});

xf.Serialize(fs, points);

break;

}

}

}

private void btnDeserialize\_Click(object sender, EventArgs e)

{

var dlg = new OpenFileDialog();

dlg.Filter = "SOAP|\*.soap|XML|\*.xml|JSON|\*.json|Binary|\*.bin";

if (dlg.ShowDialog() != DialogResult.OK)

return;

using (var fs =

new FileStream(dlg.FileName, FileMode.Open, FileAccess.Read))

{

switch (Path.GetExtension(dlg.FileName))

{

case ".bin":

var bf = new BinaryFormatter();

points = (Point[])bf.Deserialize(fs);

break;

case ".soap":

var sf = new SoapFormatter();

points = (Point[])sf.Deserialize(fs);

break;

case ".xml":

var xf = new XmlSerializer(typeof(Point[]), new[] {typeof(Point3D)});

points = (Point[])xf.Deserialize(fs);

break;

}

}

listBox.DataSource = null;

listBox.DataSource = points;

}

1. Реализация сериализации коллекции в JSON
2. Измените код кнопок сериализации и десериализации как указано ниже.

private void btnSerialize\_Click(object sender, EventArgs e)

{

var dlg = new SaveFileDialog();

dlg.Filter = "SOAP|\*.soap|XML|\*.xml|JSON|\*.json|Binary|\*.bin";

if (dlg.ShowDialog() != DialogResult.OK)

return;

using (var fs =

new FileStream(dlg.FileName, FileMode.Create, FileAccess.Write))

{

switch (Path.GetExtension(dlg.FileName))

{

case ".bin":

var bf = new BinaryFormatter();

bf.Serialize(fs, points);

break;

case ".soap":

var sf = new SoapFormatter();

sf.Serialize(fs, points);

break;

case ".xml":

var xf = new XmlSerializer(typeof(Point[]), new[] { typeof(Point3D) });

xf.Serialize(fs, points);

break;

case ".json":

var jf = new JsonSerializer();

using (var w = new StreamWriter(fs))

jf.Serialize(w, points);

break;

}

}

}

private void btnDeserialize\_Click(object sender, EventArgs e)

{

var dlg = new OpenFileDialog();

dlg.Filter = "SOAP|\*.soap|XML|\*.xml|JSON|\*.json|Binary|\*.bin";

if (dlg.ShowDialog() != DialogResult.OK)

return;

using (var fs =

new FileStream(dlg.FileName, FileMode.Open, FileAccess.Read))

{

switch (Path.GetExtension(dlg.FileName))

{

case ".bin":

var bf = new BinaryFormatter();

points = (Point[])bf.Deserialize(fs);

break;

case ".soap":

var sf = new SoapFormatter();

points = (Point[])sf.Deserialize(fs);

break;

case ".xml":

var xf = new XmlSerializer(typeof(Point[]), new[] { typeof(Point3D) });

points = (Point[])xf.Deserialize(fs);

break;

case ".json":

var jf = new JsonSerializer();

using (var r = new StreamReader(fs))

points = (Point[])jf.Deserialize(r, typeof(Point[]));

break;

}

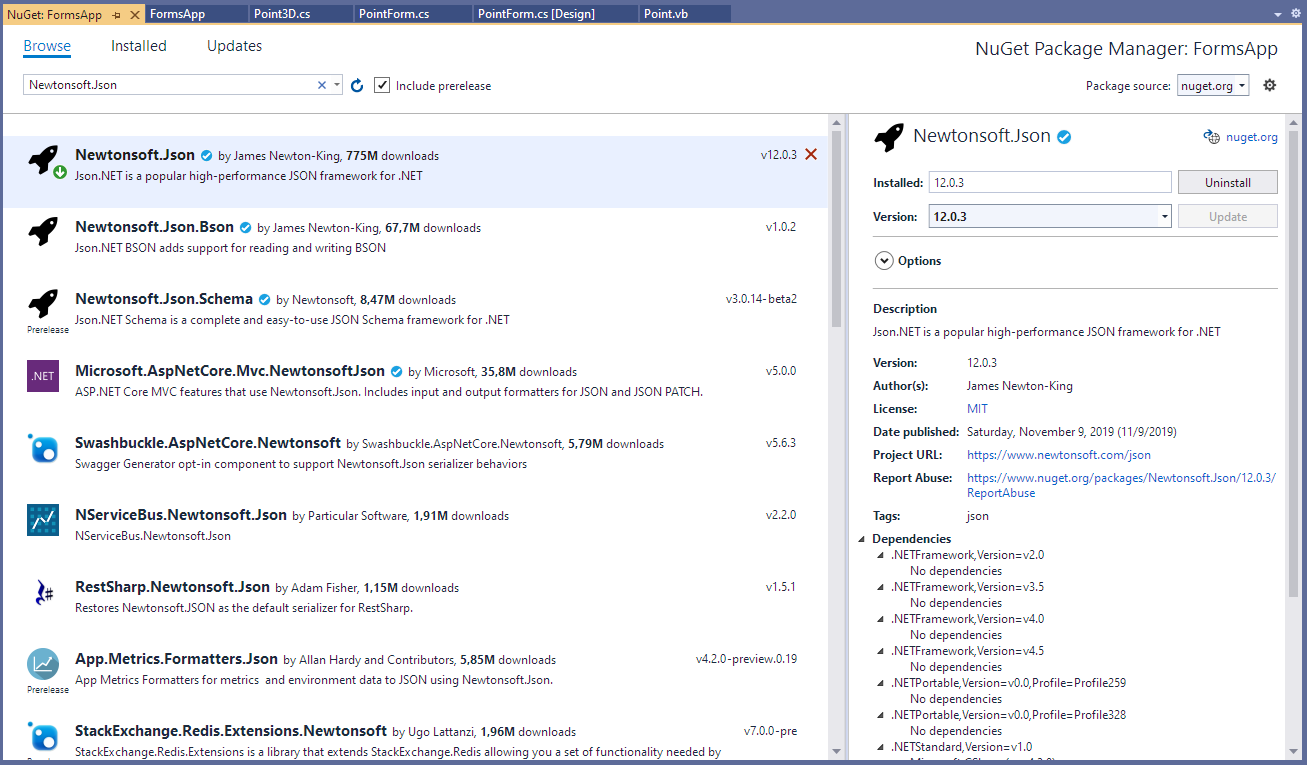
}

listBox.DataSource = null;

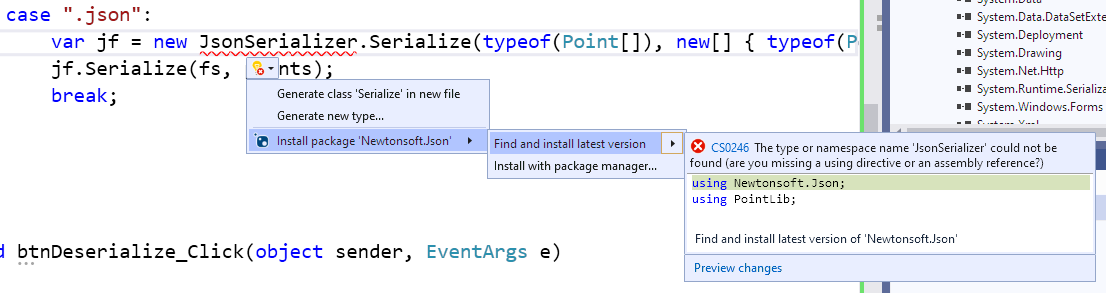
listBox.DataSource = points;

}

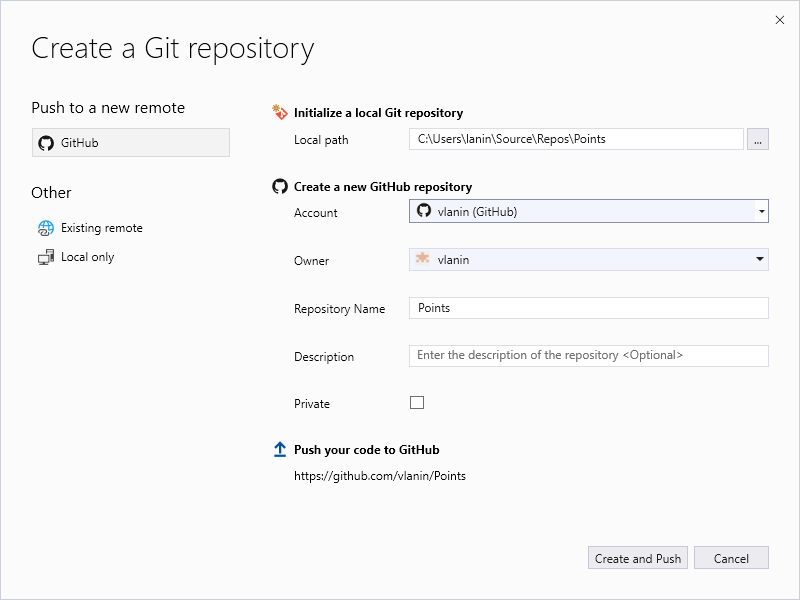
1. Установите NuGet-пакет *Newtonsoft.Json*, используя команду главного меню **Project|Manage NuGet packages…** (Рис. 15) или команду в контекстом меню кода (Рис. 16).



1. Окно добавления ссылки на пакет NuGet



1. Добавления ссылки на пакет NuGet из окна редактирования кода
2. Проверьте работоспособность сериализации. Откройте полученный файл в текстовом редакторе.
3. Проверьте работоспособность десериализации. Правильно ли происходит процесс? Попробуйте исправить ошибку.
4. Публикация проекта в репозитории GitHub
5. Создайте учетную запись на <https://github.com/>
6. Выполните команду главного меню **Git|Create Git Repository…**
7. В окне **Create Git Repository ()** задайте учетную запись GitHub, имя репозитория, снимите флажок с поля Private.
8. Нажмите кнопку Create and Push для публикации проекта в репозитории.



1. Окно создания репозитория Git
2. Зайдите на сайт <https://github.com/> и убедитесь в успешной публикации репозитория.