МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РФ
ВОЛГОГРАДСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

На правах рукописи

А.Е. Андреев, М.А. Кузнецов,
В.Л. Абдрахманов

Технологии программирования и инструментальные средства разработки систем искусственного интеллекта

Учебно-методическое пособие

Волгоград
2021

УДК 004.4’2:004.41:004.8

Рецензенты:

кафедра информационных систем и компьютерного моделирования
ФГАОУ ВО «Волгоградский государственный университет»,
зав. каф. проф., д.ф.-м. наук *А.В. Хоперсков*;

Технический директор ООО «Ай-Теклабс»
А.П. Бугров

Печатается по решению редакционно-издательского совета
Волгоградского государственного технического университета

Андреев А.Е.
Технологии программирования и инструментальные средства разработки систем искусственного интеллекта / Андреев А.Е., М.А.. Кузнецов, В.Л. Абдрахманов; ВолгГТУ. – Волгоград, 2021. – 100 с.
ISBN 978-5-9948-0000-0

В учебно-методическом пособии рассмотрены технологии и инструментальные средства создания объектно-ориентированных приложений, включая распределенные, используемые в том числе при создании систем искусственного интеллекта (СИИ). Учебное пособие предназначено для магистров, обучающихся по программам магистратуры по профилю «Искусственный интеллект» по направлениям 09.04.01 «Информатика и вычислительная техника», 09.04.03 «Прикладная информатика», 09.04.02 «Информационные системы и технологии». Учебное пособие выполнено в рамках реализации гранта на разработку программ бакалавриата и программ магистратуры по профилю «Искусственный интеллект», а также на повышение квалификации педагогических работников образовательных организаций высшего образования в сфере искусственного интеллекта (конкурс 2021-ИИ-01 от 10.06.2021).

Ил. 42. Табл. 3. Библиогр.: 16 назв.

ISBN 978-5-9948-0000-0

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Технология программирования изучает технологические процессы разработки программного обеспечения, а также порядок их применения на разных этапах разработки, определяемые технологическими подходами. В рамках данного пособия больший акцент делается на технологических процессах, характерных для гибких адаптивных (agile) подходов к разработке, к таким процессам относятся, в частности: рефакторинг, модульное тестирование, использование библиотек подставных объектов (mock), широкое использование паттернов проектирования, в том числе при осуществлении рефакторинга. Также уделяется внимание архитектурным каркасам, объединяющим в себе многие из перечисленных технологий и подходов, в частности, рассматриваются каркасы MVC и родственные им. Отдельное внимание уделено библиотекам языка Python, применяемым в машинном обучении, а также в системах искусственного интеллекта в целом.

1 ТЕСТИРОВАНИЕ ПРИЛОЖЕНИЙ, ПЕРЕРАБОТКА КОДА, АРХИТЕКТУРНЫЕ ПАТТЕРНЫ И УПАКОВКА ПРОЕКТОВ

Используемые в данном практикуме подходы и технологические процессы рассмотрены в многочисленных литературных источниках, в том числе в пособии [1]. В частности, в пособии [1] рассмотрено модульное тестирование и рефакторинг (переработка кода), использование паттернов проектирования классов. Однако ряд вопросов в пособии [1] не были отражены из-за его ограниченного объема, поэтому перед рассмотрением практикума кратко остановимся на них.

1.1 Модульное тестирование и использование подставных объектов (мокирование)

В пособии [1] рассмотрено модульное (юнит) тестирование как технологический процесс не только собственно тестирования, но и разработки через тесты (Test-Driven Development), рассмотрены шаблоны (приемы) тестирования, библиотека модульного тестирования, приведены примеры. Однако необходимо подробнее остановиться на концепции подставных объектов (mock objects) при модульном тестировании (мокировании).

Подставные объекты (mock, заглушки, stub) позволяют изолировать тестируемый объект (модуль) и гарантировать, что возможные ошибки не содержатся в тех объектах, с которыми взаимодействует тестируемый объект (модуль). Помимо собственно тестопригодности такой подход повышает гибкость кода и всего проекта за счет введения зависимостей от интерфейсов, а не от конкретных их реализаций. Это свидетельствует о пользе модульного тестирования не только с точки зрения повышения надежности программного обеспечения, но и с точки зрения его гибкости, адаптивности к изименениям и качества проектных решений в целом.

Рассмотрим, например, иллюстрацию, приведенную в [2].

Необходимо провести тестирование объекта класса Платежная ведомость (Payroll) в сильносвязанной системе из нескольких объектов, показанной на рисунке 1.

При тестировании Payroll мы не можем гарантировать, что причины возможных ошибок кроются в самом классе Payroll, они также могут быть вызваны ассоциированными с ним классами.

Выход состоит в выделении интерфейсов, в которых нуждается Payroll, а затем в реализации этих интерфейсов как исходными классами Employee, Employee Database и CheckWriter, так и с помощью подставных классов MockEmployee, MockEmployeeDataBase (рисунок 2). И если под-

ставной сотрудник MockEmployee еще может вызывать вопрос (как правило, такие классы-сущности Entities не мокируются), то два остальных введенных интерфейса вполне полезны не только для самого тестирования, но и для повышения гибкости и адаптируемости всего проектного решения, так как варианты и генератора чеков, и базы данных впоследствии могут быть самыми различными.

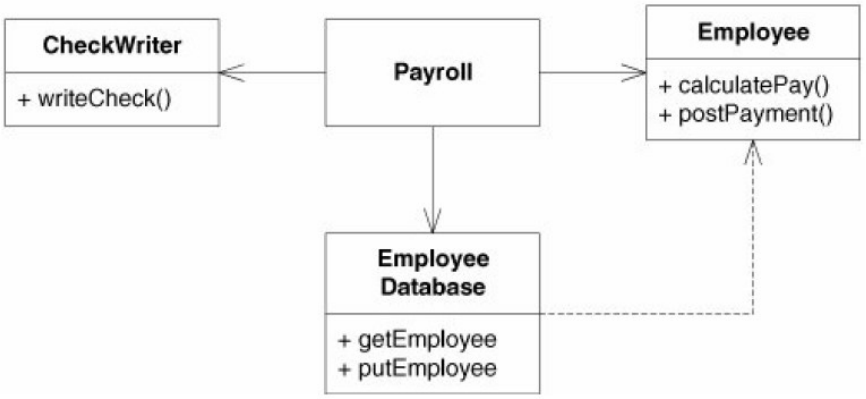


Рис.1 Исходная диаграмма классов сильносвязанной системы

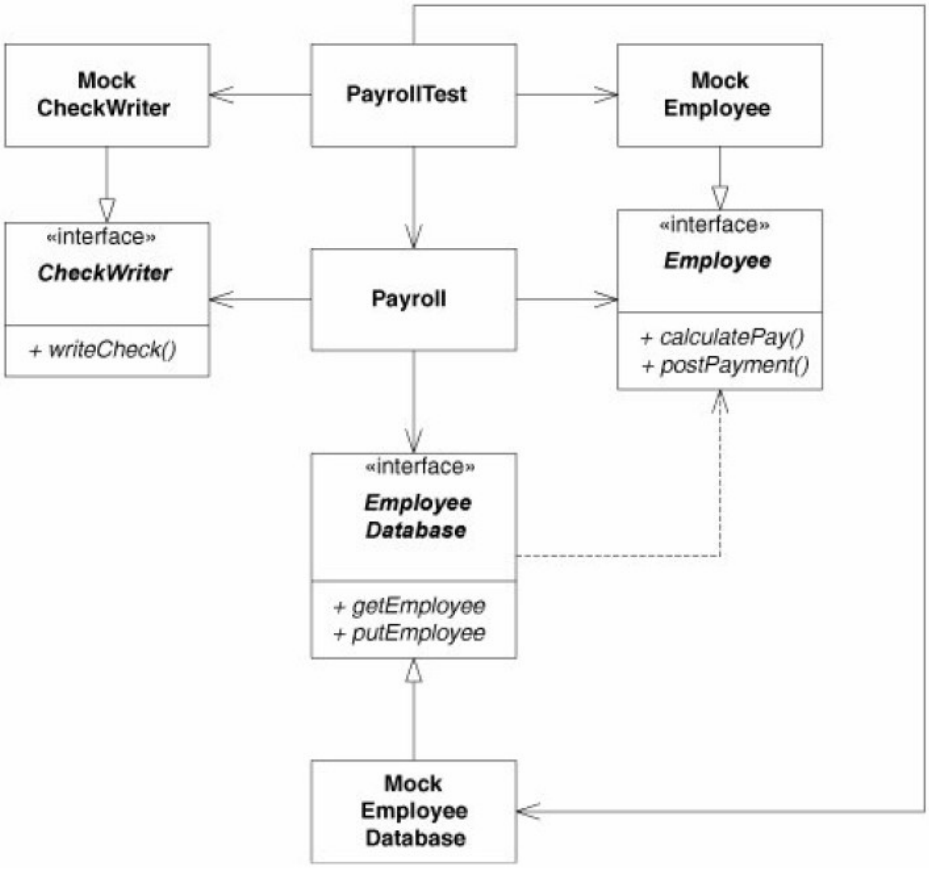


Рис. 2 Диаграмма классов с подставными объектами

Данный пример показывает, как модульное тестирование помогает принимать более удачные проектные решения.

Реализовать подставные объекты можно и самостоятельно, используя генерацию нужных объектов в памяти с помощью констант и коллекций констант, а можно воспользоваться mock-библиотеками.

1.2 Архитектурные шаблоны

Из многочисленных архитектурных шаблонов в рамках данного практикума наибольшее внимание уделяется концепции (и паттерну) MVC и родственными ему MVP и MVVM.

Паттерн MVC (model — view — controller) подразумевает взаимодействие трех компонентов: контроллера (controller), модели (model) и представления (view) – рисунок 3.

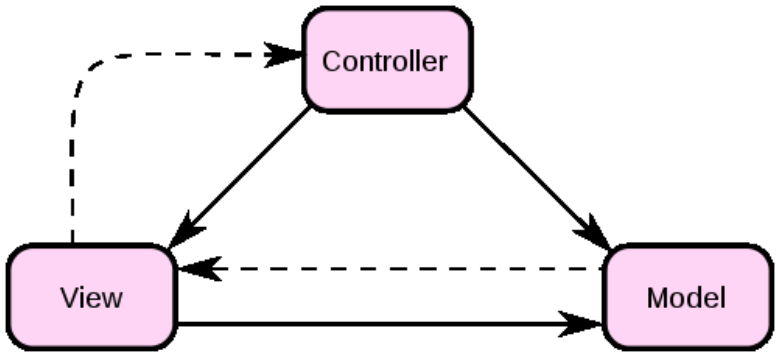


Рис.3. Концепция MVC – основные компоненты

Контроллер (controller) — это класс, с которого начинается работа приложения. Этот класс обеспечивает связь между моделью и представлением. Получая вводимые пользователем данные, контроллер исходя из внутренней логики при необходимости обращается к модели и генерирует соответствующее представление.

Представление (view) — это визуальная часть или пользовательский интерфейс приложения, как правило html-страница, через которую пользователь, зашедший на сайт, взаимодействует с веб-приложением.

Модель (model) — это набор классов, описывающих логику используемых данных.

Паттерн MVP (Model-View Presenter) отличается тем, что в нем Контроллер становится Презентером в том смысле, что он явно (или через выделенный интерфейс IView) управляет View и скрывает от него наличие модели полностью или частично (рисунок 4). При наличии IView появляется возможность использовать различные View с общим IView. Роль Презентера в данной системе, как правило, более активна, чем контроллера в MVC. Так, обычно контроллер предоставляет методы для вызова View, а вот Презентер сам вызывает методы View через интерфейс IView, получая от View только информацию о событиях.

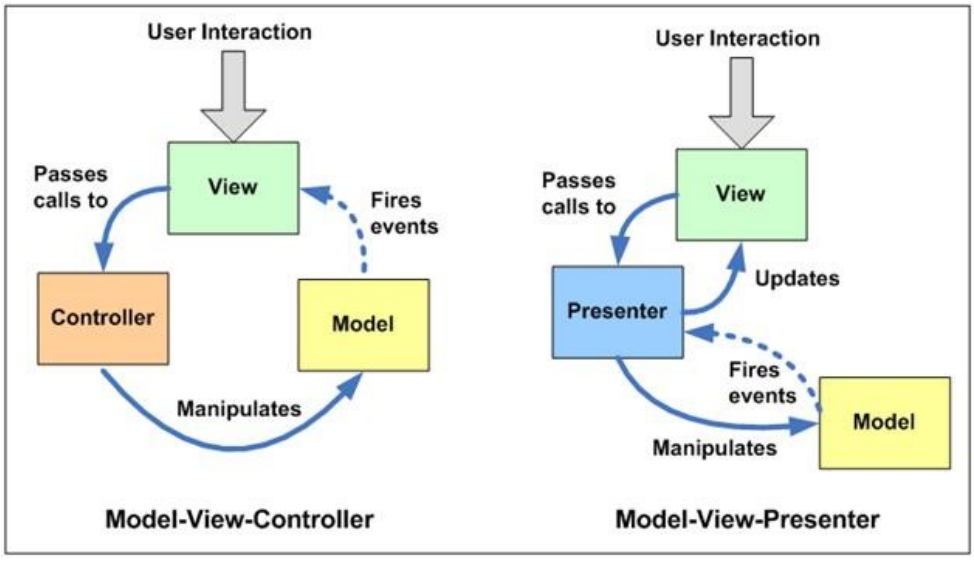


Рис.4. Сравнение реализуемых MVC и MVP
(из Niraj Bhatt, http://nirajrules.wordpress.com)

Наконец, паттерн MVVM (Model - View – ViewModel) еще в большей степени, чем в MVP с IView уменьшает роль View в реализации логики ин-

терфейса пользователя. Зачастую View в MVVM сводится к формам, реализуемым с помощью языков разметки, в которых выполняется привязка (binding) к данным и обработчикам событий (через команды по паттерну Команда), реализуемым в ViewModel (рисунок 5).

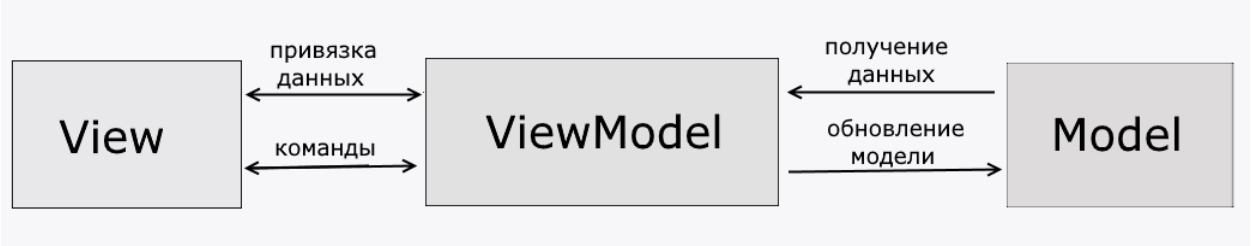


Рис. 5. Паттерн MVVM

1.3 Принципы и метрики упаковки проектов

Задача упаковки состоит в распределении классов (модулей) по более крупным подсистемам приложения (решения, проекта) – пакетам. В качестве пакетов на начальном этапе выступают логические подсистемы – пространства имен (namespaces), а затем пакеты становятся физическими подсистемами, компонентами решения (исполняемые модули, библиотеки и др.). При распределении необходимо учитывать такие общие принципы объектного проектирования как Сильное зацепление – слабое связывание (методы и данные класса должны быть сцеплены друг с другом, но слабо связаны с внешним миром), а также принципы Единичной ответственности (класс отвечает за одну группу функций, должна быть одна группа причин для его изменения), Инверсии зависимостей DIP (абстракциям не следует зависеть от деталей, такие зависимости необходимо инвертировать с помощью выделения интерфейсов) и другие принципы SOLID [2], которые в данном случае распространяются с классов на пакеты. Кроме того, упаковка должна вестись с учетом повторного использования кода (code reuse).

Принципы упаковки рассматриваются в [2].

Принципы формирования пакетов :

1. Принцип эквивалентности повторного использования и выпусков (Reuse – Release Equivalence Principle - REP).

Либо все классы, входящие в состав пакета, являются повторно используемыми, либо ни один не является таковым.

2. Принцип совместного повторного использования (Common Reuse Principle - CRP).

Классы, входящие в состав пакета, могут повторно использоваться совместно.

3. Принцип совместного закрытия (Common Closure Principle - CCP).

Классы, входящие в состав пакета, должны быть закрыты по отношению к одним и тем же изменениям. Изменение, влияющее на пакет, затрагивает все классы, входящие в него.

Принципы связывания пакетов :

4. Принцип ацикличных зависимостей (Acyclic Dependencies Principle - ADP). Не допускается наличие циклов в графе, описывающем зависимости пакета (рисунок 6).

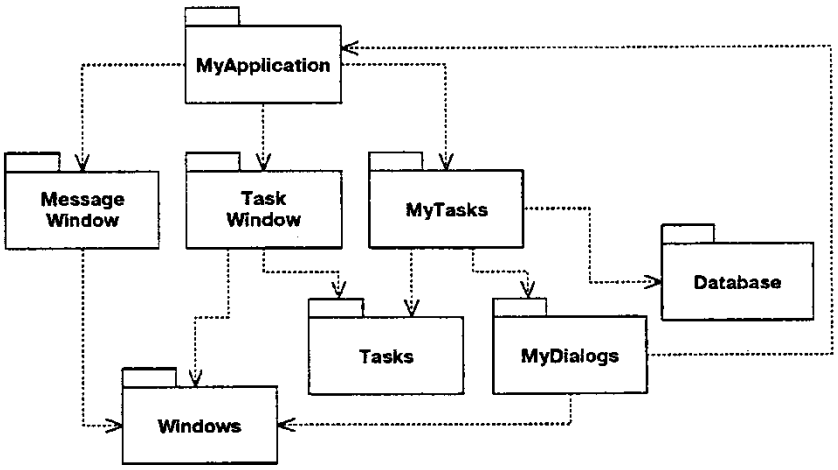


Рис. 6. Циклическая зависимость на диаграмме пакетов

5. Принцип устойчивых зависимостей (Stable Dependencies Principle – SDP).

Степень зависимости от пакета должна быть пропорциональна его устойчивости (или стабильности). Устойчивость прямо пропорциональна количеству зависимостей от пакета и обратно пропорциональна его собственным зависимостям (рисунок 7).

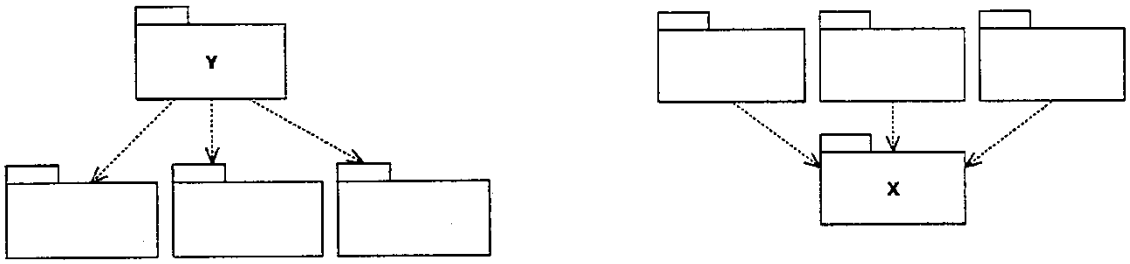


Рис. 7. К понятию устойчивости – пакет Y является неустойчивым, пакет X – устойчивым, пакет X не должен зависеть от Y

6. Принцип устойчивости абстракций (Stable Abstractions Principle - SAP).

Пакет должен быть устойчивым в той же степени, в которой он абстрактен.

Способы соблюдения принципов упаковки :

1. Перенос классов из пакета в пакет.
2. Выделение и встраивание пакетов.
3. Применение принципа DIP.
4. Использование паттернов GoF и других, в частности – Абстрактной Фабрики
5. Изменение степени абстрактности пакетов добавлением / удалением интерфейсов и др.

Так, на рисунке 8 приведена иллюстрация применения принципа DIP для выполнения принципа SDP – класс U из устойчивого пакета зависит от класса C из неустойчивого (гибкого) пакета. Мы выделяем интерфейс IU, от которого зависит U, в отдельный пакет UInterface, теперь от него зави-

сят как устойчивый пакет, так и неустойчивый, но так как сам пакет UInterface также является устойчивым, утойчивость исходного пакета с классом U практически не снижается.

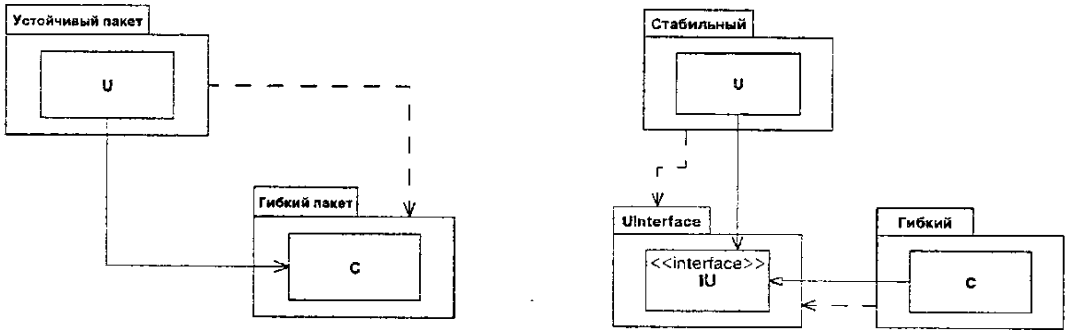


Рис. 8. Соблюдение принципа SDP за счет выделения интерфейса

Для количественной оценки степени зацепления, связывания, устойчивости, абстрактности пакетов используют количественные характеристики – метрики упаковки [2]:

1) I – неустойчивость, I = Ce / (Ce + Ca);

2) A – абстрактность, A = NA / N;

где N – количество классов / модулей в пакете;

NA – количество абстрактных классов (интерфейсов) в пакете;

Ca – центростремительное связывание (сколько классов из других пакетов зависят от классов этого пакета);

Ce – центробежное связывание (от скольких классов из других пакетов зависят классы этого пакета).

3) H – относительное сцепление, H = (R + 1) / N;

где R – количество связей внутри пакета.

Обычно метрики I и A отображают на графике, в котором по оси X откладывают I, а по оси Y – A. Тогда главной последовательностью называется диагональ квадрата, в котором могут размещаться значения метрик, проходящая через точки (0, 1) и (1, 0) - рисунок 9.

4) D – расстояние до главной последовательности,

D = | A + I – 1 | / 20,5 ;

D' – нормализованное расстояние, D' = | A + I – 1|.

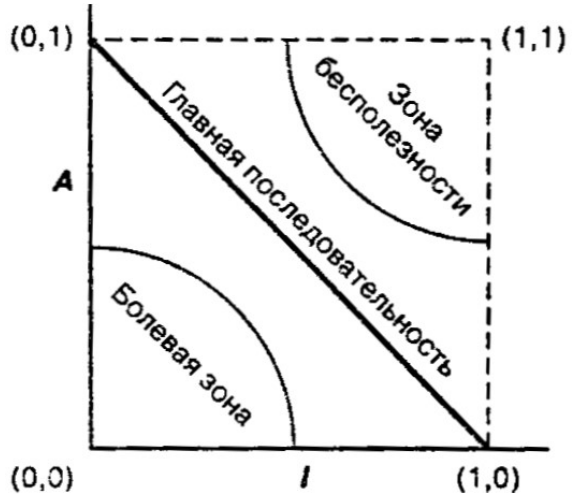


Рис. 9. Графическое отображение метрик упаковки

2 ЛАБОРАТОРНЫЙ ПРАКТИКУМ

2.1 Лабораторная работа № 1 Разработка адаптируемых многослойных приложений с применением технологий гибкой разработки (рефакторинг, модульные тесты, паттерны проектирования)

2.1.1 Цель работы

Целью данной работы является первоначальное знакомство с рядом полезных практик разработки приложений: приемами рефакторинга, концепцией расслоения приложений с использованием объектного подхода, принципами объектного проектирования, паттернами проектирования.

2.1.2 Краткое описание работы

Работу можно рассматривать как быстрое неформальное введение в инструменты и методологию построения современных адаптируемых многослойных систем с использованием гибкого объектного подхода. Рассмотренные в работе практики рассматриваются и развиваются более подробно в последующих работах: изучение паттернов проектирования классов, пользовательского интерфейса и взаимодействия с базами данных (БД), тестирование, декомпозиция и упаковка проектов. В качестве базовых примеров, на которых демонстрируются упомянутые практики, рассматривается небольшое приложение, взаимодействующее с БД.

Один пример разработан для платформы .NET на языке C# с использованием библиотек Windows Forms и ADO.NET, в нем рассматривается GUI приложение для учета сотрудников организации с расчетом небольшой статистики.

Рассматривается программа на C# с GUI для обработки данных из одной таблицы, включая типичный CRUD (Create, Read, Update, Delete), выполняемый с помощью SQL – запросов (соответственно Insert, Select, Update, Delete). В качестве библиотеки доступа к данным используется ADO.NET. Предусмотрена проверка вводимых данных, а также расчет статистики средствами языка (без использования SQL).

Имеется таблица, описывающая людей (предположим, сотрудников предприятия) :

Persons (Id_Person PK, Lastname, FirstName, MiddleName,
Gender, BirthDate, WorksFrom)

Нужно создать форму со списком людей (добавление, удаление, редактирование – рисунок 10) с краткой информацией о стаже (рисунок 11а), а также – форму для вывода статистики (рисунок 11б).

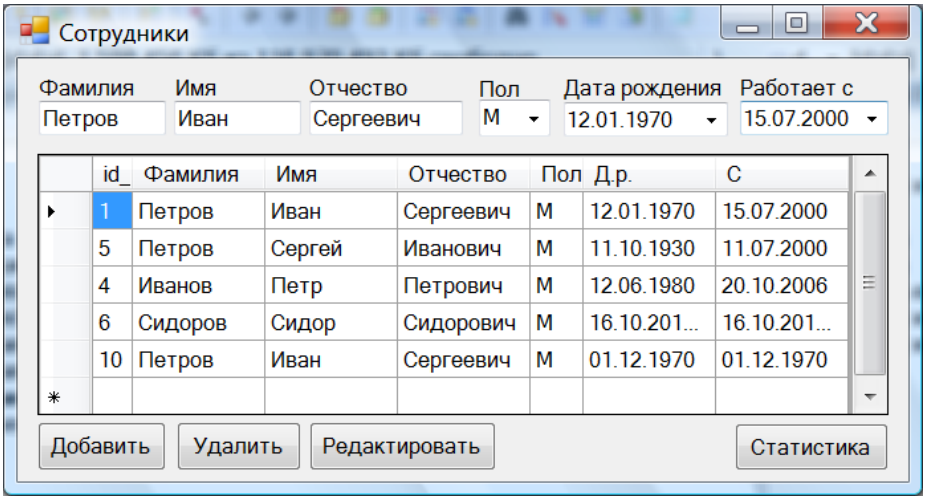
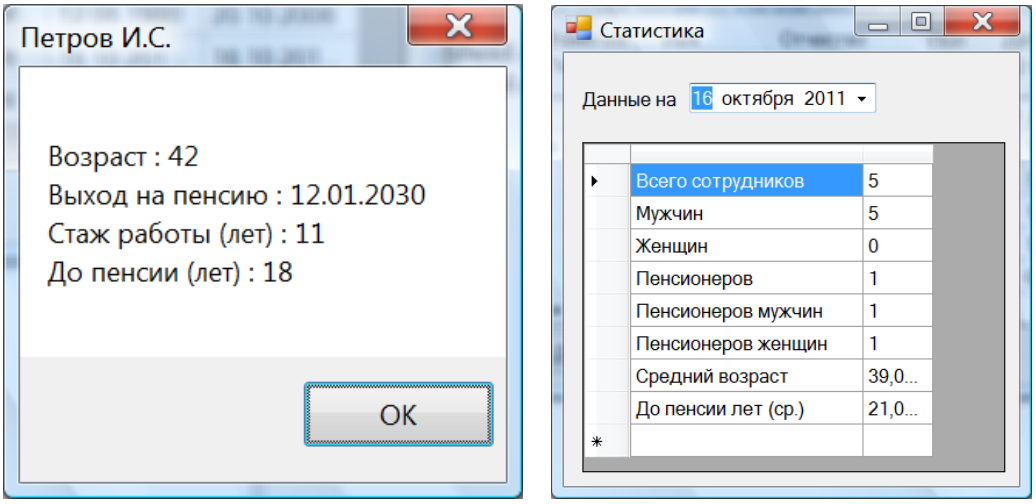


Рис.10. Форма с данными о сотрудниках



(а) (б)

Рис. 11. Информация о стаже работника (а) и общая статистика (б)

Сначала рассматривается так называемый «спагетти» - код в методах обработки событий обработки нажатия кнопок, в которых совместно ведется работа с элементами интерфейса пользователя, обработка логики (бинес-правил), работа с базой данных, например :

```
private void button3_Click(object sender, EventArgs e)
 {
     // Проверка правильности дат – работа с календарем
```

```
TimeSpan ts = dateTimePicker2.Value –
     dateTimePicker1.Value;
// Бизнес-правила
if ((ts.TotalDays / 365) < MIN_AGE ||
     dateTimePicker1.Value.Year < MIN_YEAR
    || dateTimePicker2.Value > DateTime.Now)
{
      // Диалоговое окно
      MessageBox.Show("Неправильно введены даты !");
      return;
}
// Подключение к БД
SqlCeConnection cnn = new SqlCeConnection(
      "Data Source=Persons.sdf");
// ...
```

Также в коде встречаются многочисленные повторы как результат копирования-вставки похожих фрагментов кода, например, при работе с базой данных часто встечается фрагмент перезагрузки данных и вывода их в табличный элемент управления DataGridView :

```
SqlCeDataAdapter da = new SqlCeDataAdapter(
       "select * from Persons", cnn);
DataSet ds = new DataSet();
da.Fill(ds);
cnn.Close();
dataGridView1.DataSource = ds.Tables[0];
```

Далее рассматривается последовательное преобразование кода с помощью ряда рефакторингов (переименование, замена алгоритма, выделение метода, выделение класса, выделение интерфейса), которое приводит в итоге к реализации крупного рефакторинга «Отделение предметной области от представления».

Так, например, происходит выделение методов перезагрузки Reload() и настройки таблицы InitGrid(), а также переименование элемента управления dataGridView1 в gridPersList:

```
public frmPersons()  {
   InitializeComponent();
   Reload();
   InitGrid();
}

private void Reload()  {
    SqlCeConnection cnn = new SqlCeConnection(
          "Data Source=Persons.sdf");
    cnn.Open();
    SqlCeDataAdapter da = new SqlCeDataAdapter(
           "select * from Persons", cnn);
    DataSet ds = new DataSet();
    da.Fill(ds);
    cnn.Close();
    gridPersList.DataSource = ds.Tables[0];
}

private void initGrid() {
   gridPersList.Columns[0].Width = 30;
   gridPersList.Columns[1].Width = 100;
   // ...
   gridPersList.Columns[6].HeaderText = "C";
}
```

Далее выделяется отдельный класс для работы с SQL базой данных, а потом и интерфейс такой БД, не привязанный к конкретной реализации :

```
public interface IDB {
    DataTable QueryPersons();
    void execCommand(string sql);
}
```

За счет этого появляется возможность легко реализовывать различные источники данных, в том числе тестовый (подставной – mock) :

```
  public class TestDB : IDB  {
          public DataTable QueryPersons() {
              DataColumn dc = new
                    DataColumn("id_person");
              DataTable dt = new DataTable();
              dt.Columns.Add(dc);
              dc = new DataColumn("lastname");
              dt.Columns.Add(dc);
              DataRow dr = dt.NewRow();
              //…
           }
            public void execCommand(string sql) {  }
  }
```

Затем интерфейс и само приложение видоизменяется за счет выделения класса предметной области Person :

```
public class Person {
    public int Id { get; set; }
    public String LastName { get; set; }
    public String FirstName { get; set; }
    public String MiddleName { get; set; }
    public DateTime BirthDate { get; set; }
    public DateTime WorksFrom { get; set; }
    public Char Gender { get; set; }
```

```
    public DateTime getPensDate() {
       DateTime toPens;
       if (Gender == 'M')
            toPens = BirthDate.AddYears(MAX_M);
       else
            toPens = BirthDate.AddYears(MAX_W);
        return toPens;
    }
    // ...

public interface IDB {
    DataTable QueryPersons();
    List<Person> QueryPersonsList()
    void execCommand(string sql);
}
```

Вводятся также классы для работы с отчетом

```
public class RepItem {
     public String ItemName { get; set; }
     public String ItemValue { get; set; }
}

public class EmployeeReport {
   private List<Person> pl;
   public EmployeeReport(List<Person> _pl) {
     pl = _pl;
   }

   public List<RepItem> getReport(DateTime now) {
        // Алгоритм расчета статистики теперь здесь !
        // ...
```

Возникает возможность тестирования классов предметной области с помощью механизма модульных тестов, подробно рассматриваемого в третьей лабораторной работе :

```
using NUnit.Framework;

namespace Employees {
      public class TestPerson {
         [Test]
         public void testPension() {
            Person p = new Person();
             p.BirthDate = new DateTime(1970, 10, 1);
             p.Gender = 'M';
            Assert.AreEqual(p.BirthDate.AddYears(MAX_M),
                  p.getPensDate());
            p.Gender = 'Ж';
             Assert.AreEqual(p.BirthDate.AddYears(MAX_W),
                  p.getPensDate());
         }
      }
}
```

Выделяется также класс Фасада (паттерн проектирования) для предметной области EmployeesDomain, «закрывающий» многие детали кода от форм (например, наличие источника данных как такового).

В результате монолитное приложение преобразуется в объектно-ориентированное, в котором предметная область отделена от представления (название крупного рефакторинга) – рисунок 12.

Другой пример небольшого приложения реализован на языке Python с использованием библиотек SQLAlchemy и requests, в нем рассматривается обработка информации о погоде. Исходный вариант приложения, а также

промежуточные версии его переработки предоставляются преподавателем [3].

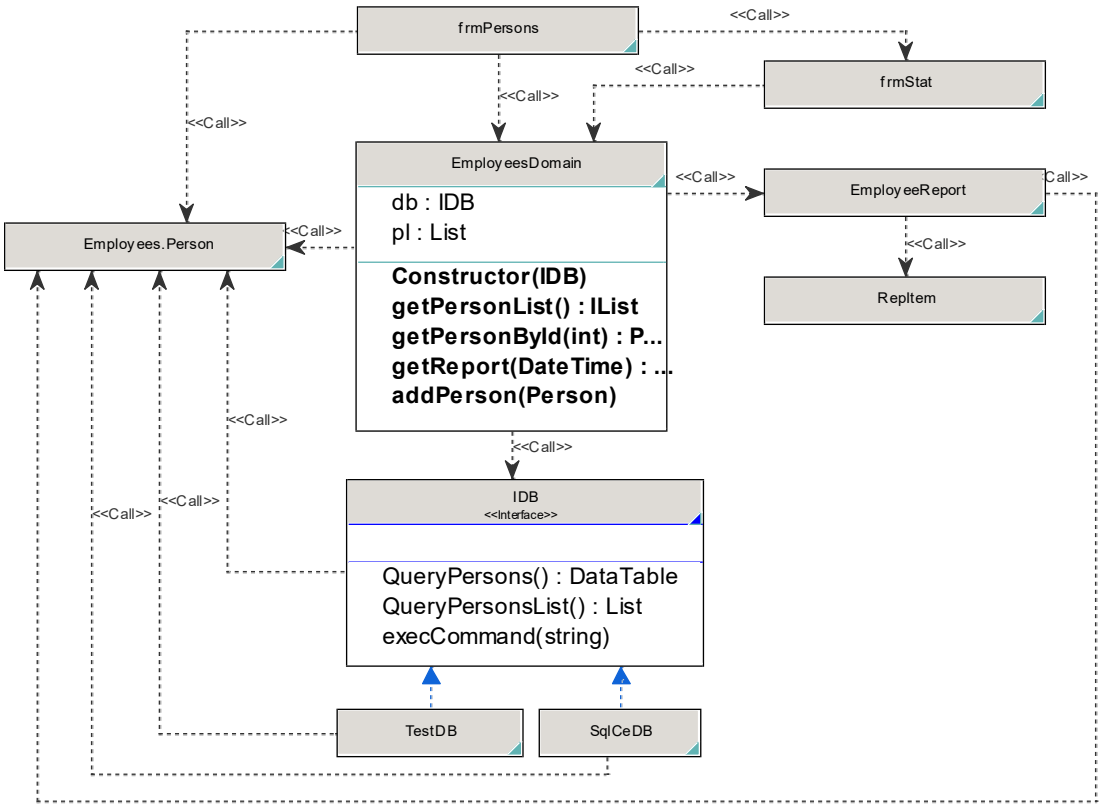


Рис.12. Диаграмма классов после рефакторинга

Исходная версия представляет собой совсем небольшой скрипт, однако и в нем можно выполнить декомпозицию на подсистемы, выполнив рефакторинги и выделив классы.

```
import requests
import sqlite3

url =
'https://weather.visualcrossing.com/VisualCrossingWebServices/rest/services/weatherdata/history'
params = {
    'aggregateHours':24,
```

```
        'startDateTime':'2020-09-29T00:0:00',
        'endDateTime':'2020-09-29T23:59:59',
        'unitGroup':'metric',
        'location': 'Volgograd,Russia',
        'key': 'I3D60I88UB6KPSDAVGK38HNP5',
        'contentType':'json'
    }
    data = requests.get(url, params).json()

    with sqlite3.connect('weather.sqlite3') as connection:
        c = connection.cursor()
        c.execute('create table if not exists weather (date text,
             mint real, maxt real, location text, humidity real)')
        for row in data['locations']['Volgograd,Russia']['values']:
            c.execute('insert into weather values (?,?,?,?,?);',
            (row['datetimeStr'][:10],  row['mint'],  row['maxt'],
            'Volgograd,Russia', row['humidity']))

    for row in c.execute('select * from weather;'):
        print(row)
```

В ходе первого шага рефакторинга добавляется поддержка ORM-библиотеку sqlalchemy для ухода от SQL запросов.

```
import requests
  from sqlalchemy import create_engine, Table, Column, String,
  Float, MetaData
from sqlalchemy.sql import select
# ...
engine = create_engine('sqlite:///weather.sqlite3')
metadata = MetaData()
```

```
    weather = Table(
        'weather', metadata,
        Column('date', String),
        Column('mint', Float),
        Column('maxt', Float),
        Column('location', String),
        Column('humidity', Float),
    )
    metadata.create_all(engine)

c = engine.connect()
for row in data['locations']['Volgograd,Russia']['values']:
    c.execute(
        weather.insert(),
        [
            {
                'date': row['datetimeStr'][:10],
                'mint': row['mint'],
                'maxt': row['maxt'],
                'location': 'Volgograd,Russia',
                'humidity': row['humidity'],
            }
          for row in data['locations']['Volgograd,Russia']['values']
        ],
    )
for row in c.execute(select([weather])):
    print(row)
```

На следующем шаге выделяется класс для работы с погодой :

```
    class WeatherProvider:
        def __init__(self, key):
            self.key = key
```

```
        def get_data(self, location, start_date, end_date):
            url =
            # см. выше
            params = {
                'aggregateHours': 24,
                # ... см. выше
            }
            data = requests.get(url, params).json()
            return [
                {
                    'date': row['datetimeStr'][:10],
                     # ... см. выше
                }
                for row in data['locations'][location]['values']
            ]
```

Основная программа :

```
engine = create_engine('sqlite:///weather.sqlite3')
metadata = MetaData()
weather = Table(
    'weather', metadata, Column('date', String),
    # ... см. выше
)
metadata.create_all(engine)

c = engine.connect()

provider = WeatherProvider('I3D60I88UB6KPSDAVGK38HNP5')
c.execute(weather.insert(), provider.get_data('Volgograd,Russia',
'2020-09-20', '2020-09-29'))
```

Следующие шаги рефакторинга студентам предлагается проделать самостоятельно.

2.1.3 Порядок выполнения работы

1) Изучение учебных примеров или одного из примеров по выбору.

Необходимо рассмотреть и выполнить отдельные этапы преобразования проекта. Перед выполнением следующего пункта переходить к очередной версии кода в системе контроля версий.

1.1) Загрузить первую версию проекта, ознакомиться с исходным кодом, запустить программу, протестировать ее работу с помощью интерфейса командной строки.

1.2) Выполнить 1-2 переименования методов, выделить 2-3 метода. Выделить класс для работы с БД.

1.3) Выделить интерфейс для работы с БД.

1.4) Выделить интерфейс для работы с источником (провайдером) данных для примера с данными о погоде.

1.5) Выделить класс – контроллер (фасад предметной области) для примера со статистикой о сотрудниках.

2) Выполнить индивидуальное задание.

2.1.4 Вопросы и задания

1. Повторить и закрепить информацию об итеративных адаптивных подходах к разработке и используемых в них технологических процессах.

2. Что такое рефакторинг ?

3. Что такое модульные тесты ? Для чего они служат ?

4. Как выполняется крупный рефакторинг ?

5. Что такое паттерны проектирования ? Их назначение ?

2.2 Лабораторная работа № 2 Изучение каркасов MVC на примере ASP.NET Core MVC и Django.

2.2.1 Цель работы

Цель работы — изучение основных приемов использования ASP.NET Core MVC и Django для создания простых веб-приложений.

2.2.2 Краткое описание работы

В работе рассматривается реализация небольшого интернет-магазина на двух каркасах : ASP.NET Core MVC и Django.

ASP.NET Core – новая версия библиотеки ASP.NET создания веб-приложений для платформы .NET Core, более переносимой и кросс-платформенной (с точки зрения операционных систем) реализации .NET. В рамках ASP.NET Core можно как реализовывать веб-сервисы, поддерживающие WebApi в стиле REST, так и создавать собственно приложения, построенные по шаблону MVC и использующие соответствующий каркас.

Для создания шаблона нового приложения можно воспользоваться Visual Studio (причем подойдет версия Community), для этого выбираем соответствующую платформу и тип приложения в мастере (рис. 13-16).

Наряду с Visual Studio Community для разработки можно воспользоваться такой кросс-платформенной средой, как Visual Studio Code. Правда, в этом случае часть действий придется делать вручную, например создание проекта (результат и настройка отображены на рисунках 17 - 20) :

```
dotnet new –t web
dotnet restore
```

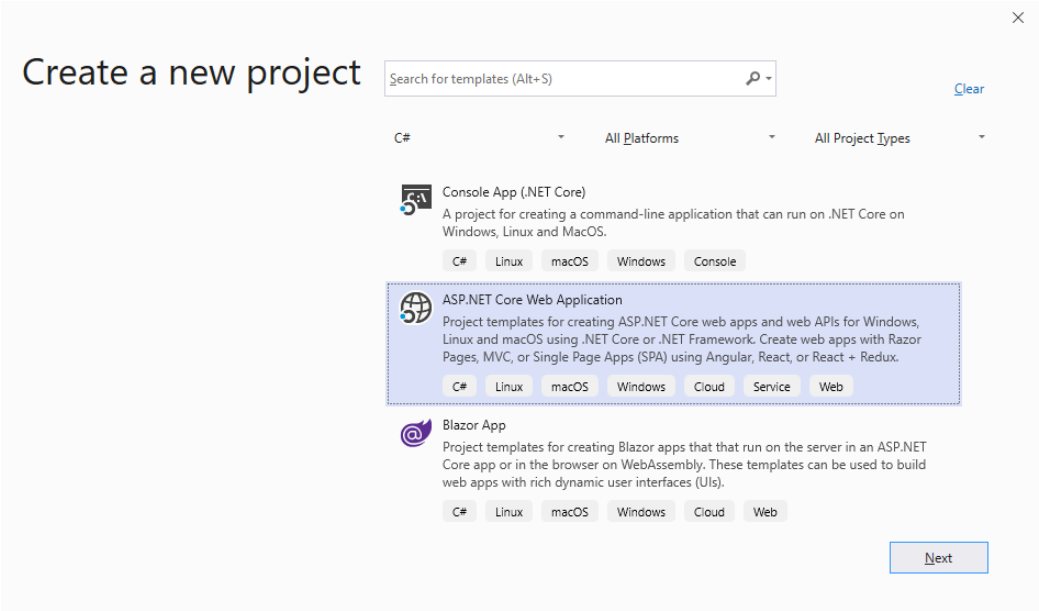


Рис. 13. Создание проекта ASP.NET Core в Visual Studio

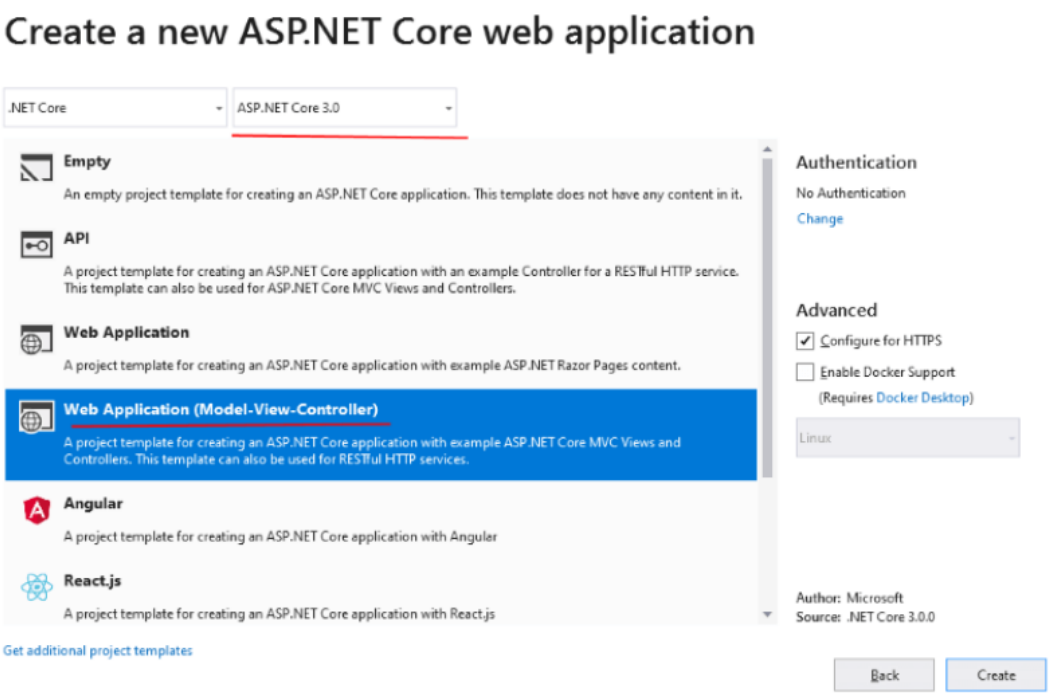


Рис. 14. Создание проекта ASP.NET Core MVC в Visual Studio

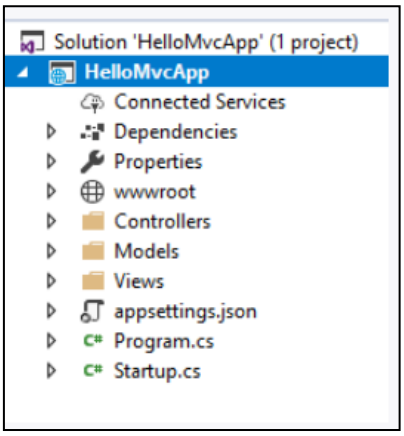


Рис. 15. Структура проекта ASP.NET Core MVC

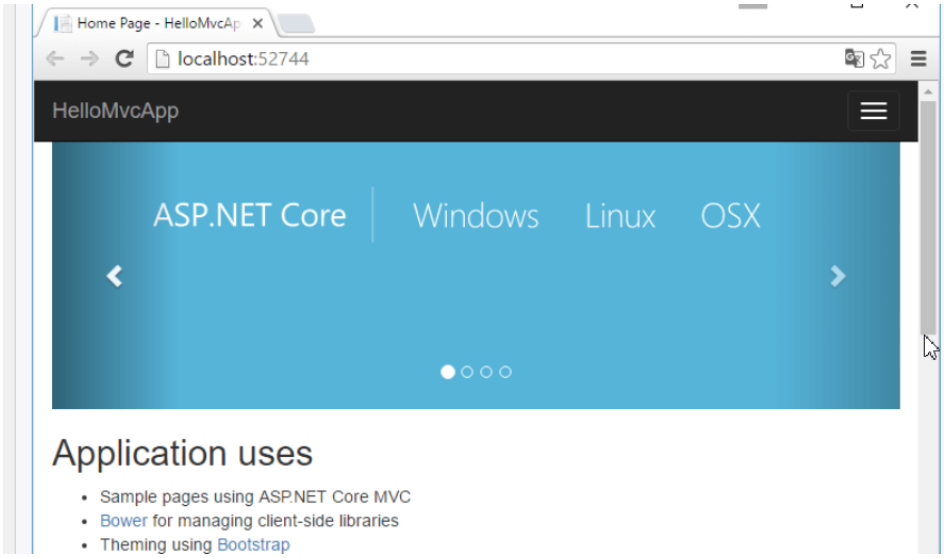


Рис. 16. Внешний вид запущенного приложения MVC в браузере

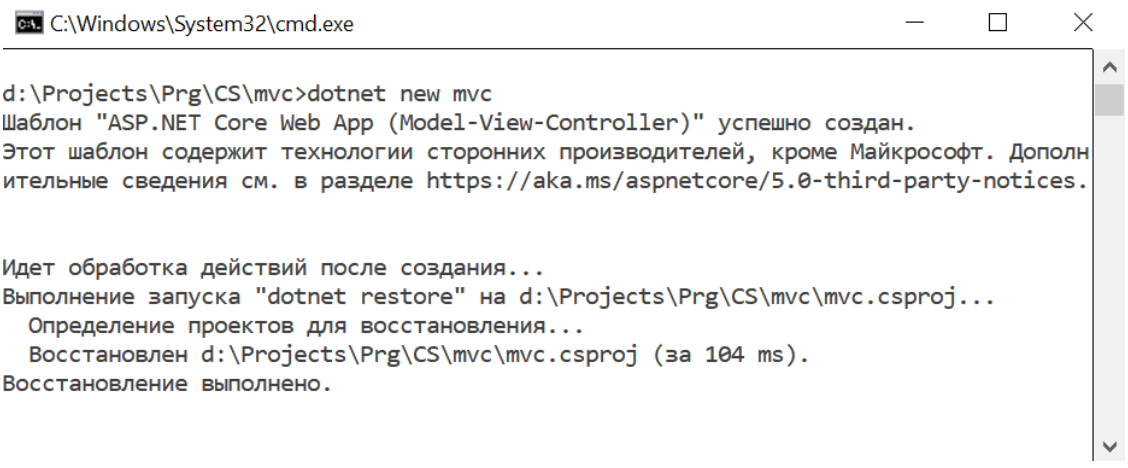


Рис 17. Создание проекта ASP.NET MVC Core в командной строке

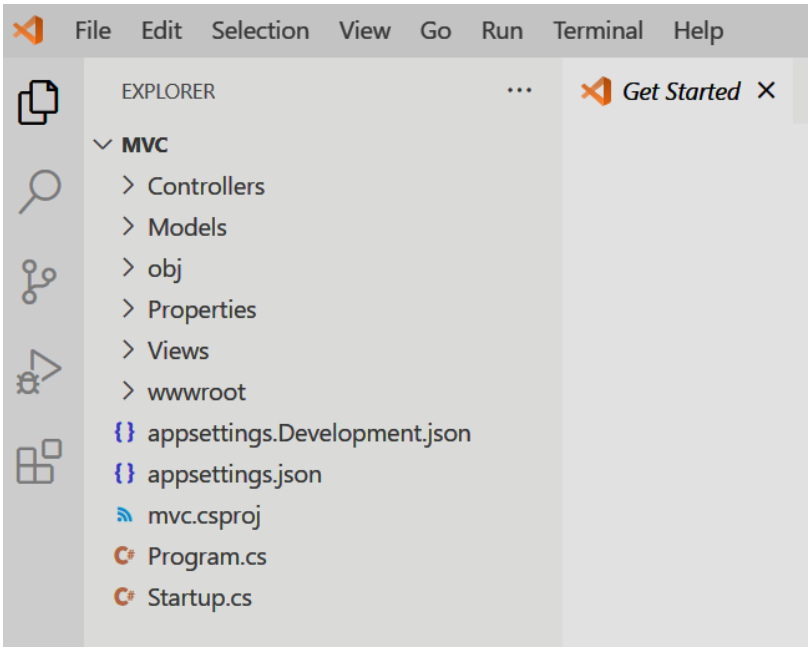


Рис 18 Структура проекта ASP.NET MVC Core в окне обозревателя проектов Visual Studio Code

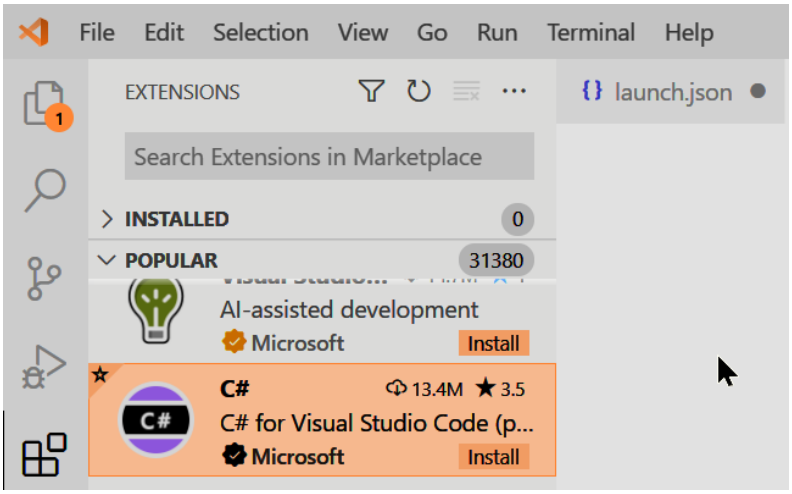


Рис. 19. Установка расширения (плагина) для C# в VS Code

```
"version": "0.2.0",
"configurations": [
    {
        "name": ".NET Core Launch (web)",
        "type" :"coreclr",
        "request": "launch",
        "preLaunchTask": "build",
        "program": "${workspaceFolder}/bin/Debug/<target-framework>/<project-name.dll>",
```

Рис. 20 Конфигурация запуска приложения в VS Code (нужно указать папку, целевую платформу и имя библиотеки для ключа «program»)

Основные фрагменты кода для примера с интернет-магазином приведены ниже.

Модель :

```
namespace example.Models {
    public class Product  {
        public int ID { get; set; }
        public string Name { get; set; }
        public int Price { get; set; }
    }

    public class Purchase     {
        public int ID { get; set; }
        public int ProductID { get; set; }
        public string Person { get; set; }
        public string Address { get; set; }
        public DateTime Date { get; set; }
    }
}
```

Один из контроллеров :

```
public class HomeController : Controller {
     private ShopContext db = new ShopContext();
     public ActionResult Index() {
         IEnumerable<Product> products = db.Products;
         ViewBag.Products = products;
         return View();
     }

     [HttpGet]
     public ActionResult Buy(int id)  {
         ViewBag.ProductId = id;
         return View();
```

```
    }

    [HttpPost]
    public string Buy(Purchase purchase) {
        purchase.Date = DateTime.Now;
        db.Purchases.Add(purchase);
        db.SaveChanges();
        return "Спасибо за покупку, " + purchase.Person + "!";
    }
}
```

Сама разметка Вида (view) в виде файла cshtml :

```
@{
    Layout = null;
}
<!DOCTYPE html>
<html>
<head>
    <meta name="viewport" content="width=device-width" />
    <title>Товары</title>
</head>
<body>
    <div>
        <h3>Список</h3>
        <table>
            <tr>
                <td><p>Наименование</p></td>
                <td><p>Цена</p></td>
                <td></td>
            </tr>
            @foreach (var p in ViewBag.Products)
            {
```

```
                <tr>
                    <td><p>@p.Name</p></td>
                    <td><p>@p.Price</p></td>
                    <td><p>
                  <a href="/Home/Buy/@p.ID">Купить</a></p></td>
                </tr>
            }
        </table>
    </div>
</body>
</html>
```

Также имеет смысл обратить внимание на использование в примере ORM Entity Framework (рассматривается подробнее при рассмотрении курсовой работы) :

```
namespace example.DAL {
    public class ShopContext: DbContext     {
        public DbSet<Product> Products { get; set; }
        public DbSet<Purchase> Purchases { get; set; }
    }
}
```

Настройка маршрутизации запросов (routing) также осуществляется в отдельном классе :

```
namespace example {
    public class RouteConfig  {
       public static void RegisterRoutes(RouteCollection routes) {
            routes.IgnoreRoute("{resource}.axd/{*pathInfo}");
            routes.MapRoute(
                name: "Default", url: "{controller}/{action}/{id}",
                defaults: new { controller = "Home",
                action = "Index", id = UrlParameter.Optional }
            );
        }
    }
```

```
}
```

Каркас Django представляет собой аналогичный MVC каркас для языка Python. Он, как ASP.NET Core MVC, поддерживает ORM, разные варианты шаблонизаторов и саму концепцию MVC для веб-приложений, но в нем используется немного другая терминология, в частности, контроллеры в нем называются View, а виды - Template :

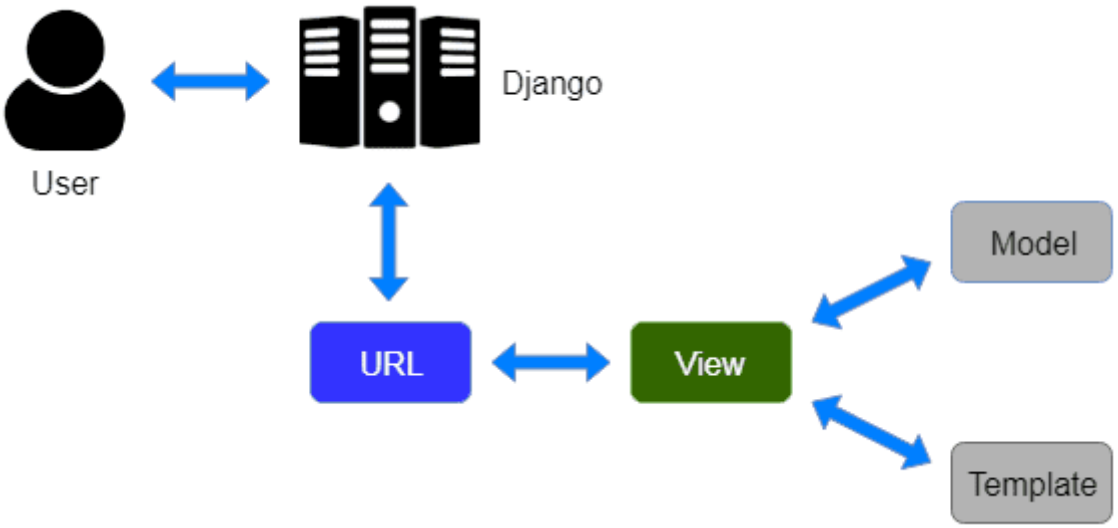


Рис. 21. Структура приложения в MVT-каркасе Django

Модель магазина :

```
from django.db import models

# Create your models here.
class Product(models.Model):
    name = models.CharField(max_length=200)
    price = models.PositiveIntegerField()

class Purchase(models.Model):
    product = models.ForeignKey(Product, on_delete=models.CASCADE)
    person = models.CharField(max_length=200)
    address = models.CharField(max_length=200)
    date = models.DateTimeField(auto_now_add=True)
```

Собственно View (контроллер в терминологии MVC) :

```
from django.shortcuts import render
from django.http import HttpResponse
from django.views.generic.edit import CreateView
from rest_framework import viewsets

from .models import Product, Purchase
from .serializers import ProductSerializer
# Create your views here.
def index(request):
    products = Product.objects.all()
    context = {'products': products}
    return render(request, 'shop/index.html', context)

class PurchaseCreate(CreateView):
    model = Purchase
    fields = ['product', 'person', 'address']

    def form_valid(self, form):
        self.object = form.save()
        return HttpResponse(f'Спасибо за покупку,
          {self.object.person}!')

class ProductViewSet(viewsets.ModelViewSet):
    serializer_class = ProductSerializer
    queryset = Product.objects.all()
```

В качестве вида (view) выступает шаблон на базе html / css (код и данные, подставляемые в шаблон, размещаются в фигурных скобках):

```
<!DOCTYPE html>
<html>
<head>
    <meta name="viewport" content="width=device-width" />
    <title>Товары</title>
</head>
<body>
    <div>
        <h3>Список</h3>
        <table>
            <tr>
                <td><p>Наименование</p></td>
                <td><p>Цена</p></td>
                <td></td>
            </tr>
            {% for p in products %}
                <tr>
                    <td><p>{{ p.name }}</p></td>
                    <td><p>{{ p.price }}</p></td>
                    <td><p>
                      <a href="/buy/{{ p.id }}">Купить</a></p></td>
                </tr>
            {% endfor %}
        </table>
    </div>
</body>
</html>
```

Маршрутизация запросов настраивается в url.py :

```
from django.urls import path
```

```
from rest_framework.routers import DefaultRouter

from . import views

router = DefaultRouter()
router.register(r'api/products', views.ProductViewSet,
       basename='api-product')

urlpatterns = [
    path('', views.index, name='index'),
    path('buy/<int:product_id>/',views.PurchaseCreate.as_view(),
       name='buy'), *router.urls
]
```

2.2.3 Порядок выполнения работы

1 Ознакомиться с примером приложения интернет-магазина (ASP.NET или Django, либо с обоими по выбору студента). Проделать основные шаги, удостовериться в работоспособности примера.

2 Добавить страницу с покупками.

3 Добавить регистрацию и авторизацию.

4 Добавить страницу "Мои покупки".

5 Выполнить индивидуальное задание.

2.2.4 Вопросы и задания

1 Опишите особенности каркаса ASP.NET MVC

2 Что нового в реализации ASP.NET Core MVC ?

3 Опишите особенности реализации MVC в каркасе Django.

4 Какие еще каркасы для создания веб-служб и приложений на Python помимо Django Вы знаете ?

5 Опишите концепцию / паттерн MVC

6 Опишите особенности паттернов MVP, MVVM. В чем их отличия от MVC ? Где применяется каждый из паттернов ?

7. Повторить и закрепить информацию о паттернах проектирования и о каркасах MVC.

2.3 Лабораторная работа № 3 Применение библиотек модульного тестирования и подставных объектов.

2.3.1 Цель работы

Цель работы — изучение основных приемов и методов модульного тестирования с использованием библиотек модульного тестирования и mock-объектов.

2.3.2 Краткое описание работы

При тестировании важной задачей является изоляция тестируемого компонента от всех прочих. Если этого не сделать, то результат теста будет зависеть не только от правильности реализации тестируемого компонента, но и от его зависимостей. Самый простой способ изоляции — использование входных параметров.

Однако не всегда зависимости являются настолько простыми. Чаще зависимость может быть представлена некоторым интерфейсом. При этом существует класс, возможно даже не один, реализующий этот интерфейс, который и используется в процессе работы приложения. Но как быть при тестировании? Можно вручную создавать специальные классы-заглушки, которые будут реализовывать соответствующий интерфейс и необходимую для тестирования функциональность. Более удобным способом является использование так называемых mock-объектов.

В работе рассматривается модульное тестирование на ряде примеров, а также на примере интернет-магазина из предыдущей лабораторной работы с использованием библиотек модульного тестирования и mock-объектов.

Сам процесс тестирования с использованием библиотеки NUnit и Visual Studio рассмотрим на примере простого теста для поиска максимума в массиве целых чисел :

```
using NUnit.Framework;
using Largest2;
namespace TestProject1  {
    public class Tests  {
        [SetUp]
        public void Setup() {}
        [Test]
        public void LargestOf3()  {
            Assert.AreEqual(9,
                Cmp.Largest(new int[] { 8, 9, 7 }));
        }
        [Test]
        public void LargestOf1()     {
            Assert.AreEqual(9,  Cmp.Largest(new int[] { 9 }));
        }
    }
}
namespace Largest2 {
    public class Cmp    {
        public static int Largest(int[] list)   {
            int index, max = Int32.MinValue;
            for (index = 0; index < list.Length; index++) {
                if (list[index] > max)
                    max = list[index];
            }
            return max;
        }
    }
}
```

При использовании Visual Studio сначала создадим проект библиотеки классов .NET Core, а потом добавим в него проект модульного теста для NUnit (рисунок 22). Затем нужно добавить ссылку в зависимостях проекта TestProject1 на тестируемый проект, чтобы получилась структура решения, показанная на рисунке 23.

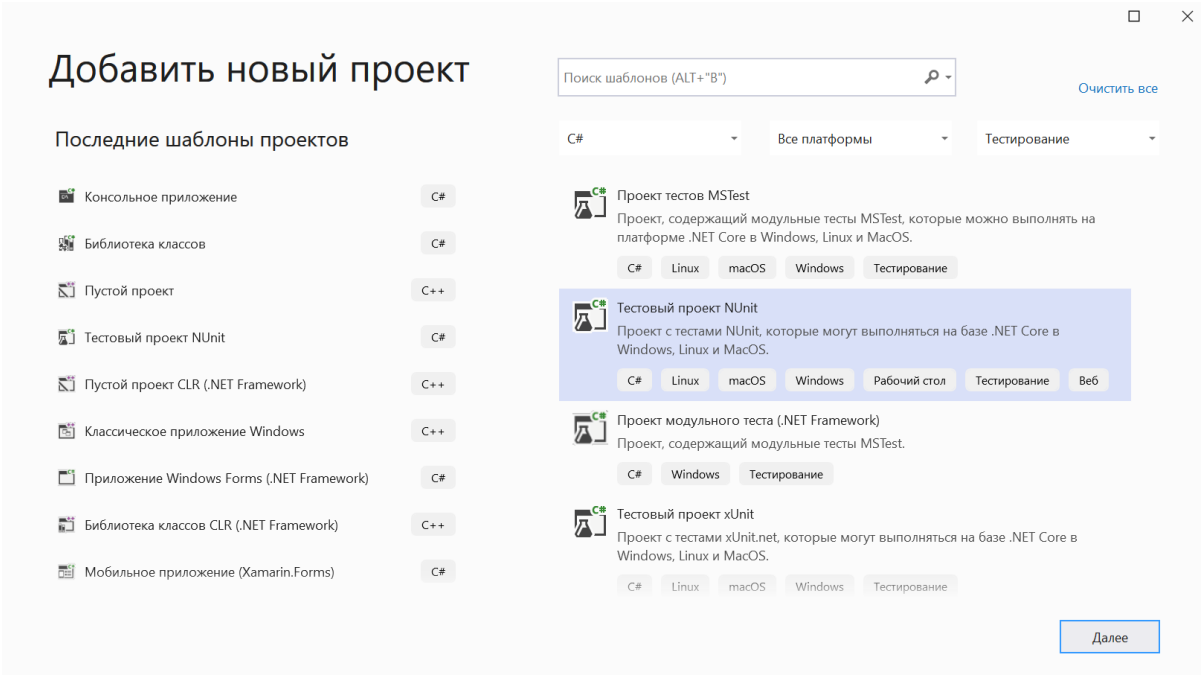


Рис. 22. Создание тестового проекта NUnit и добавление его в решение

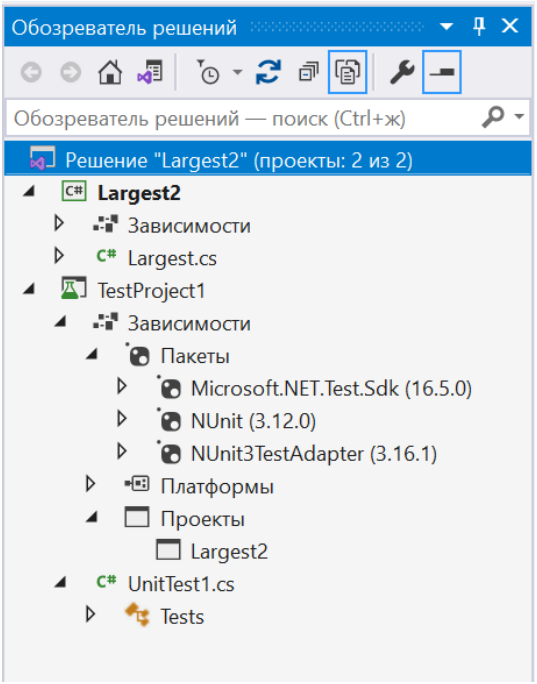


Рис. 23. Два проекта – основной и тестовый – в решении VS

После этого мы можем запустить тестирование, используя меню Тест (рисунок 24) и наблюдать результаты с помощью Обозревателя тестов (рисунки 25 – 26). На рисунке 26 показан результат, когда тест не пройден. Подробную сводку о тесте можно получить, если в окне справа кликнуть на непройденный тест.

На рисунке 37 показан результат запуска тестов в среде VS Code.

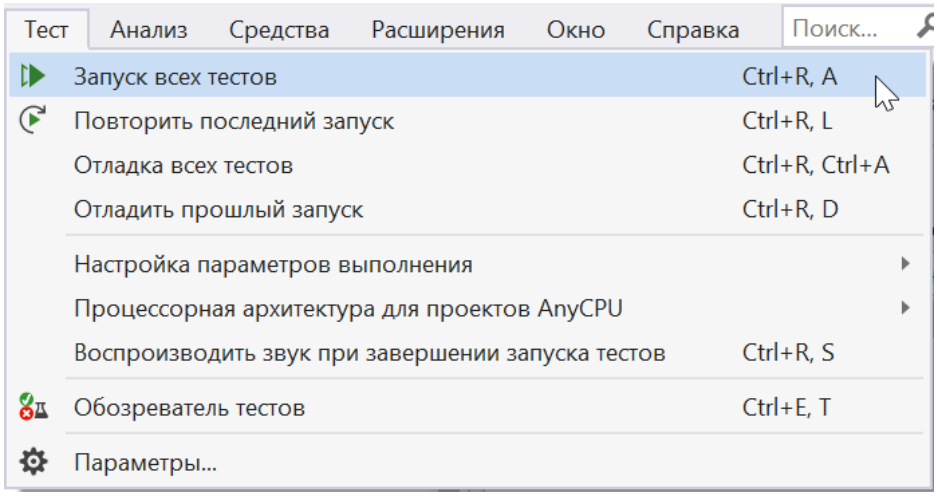


Рис. 24 Запуск всех тестов

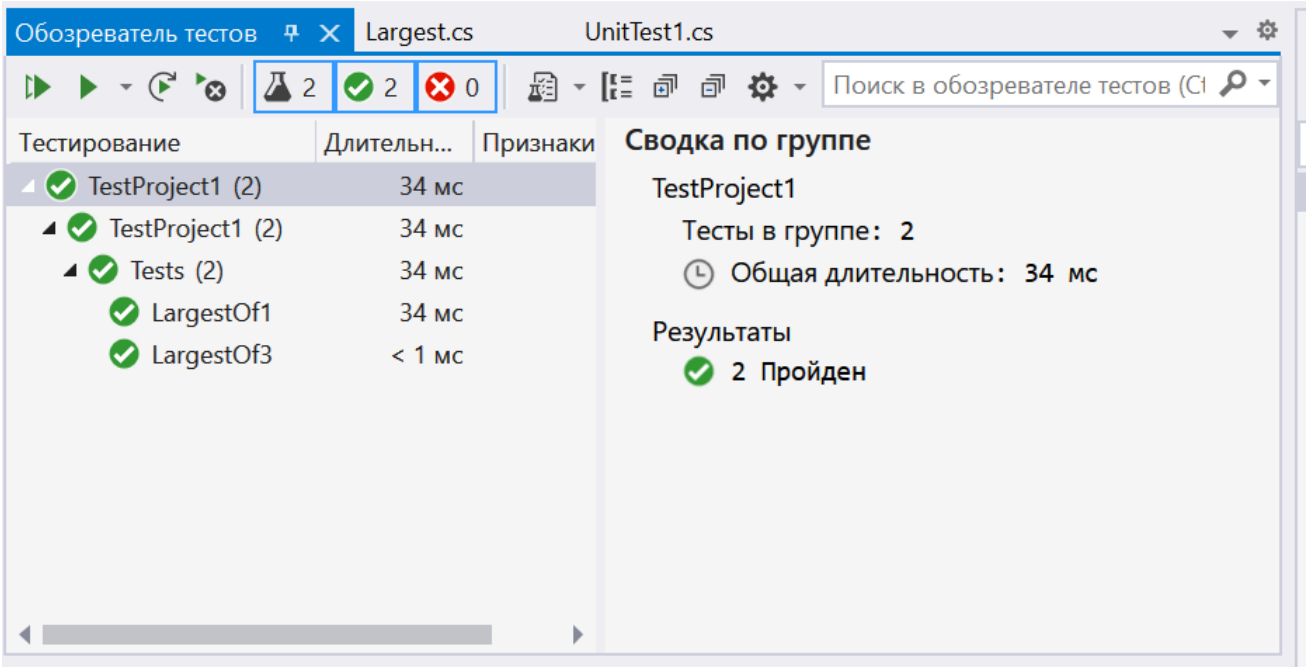


Рис. 25. Результаты тестирования (оба теста пройдены)

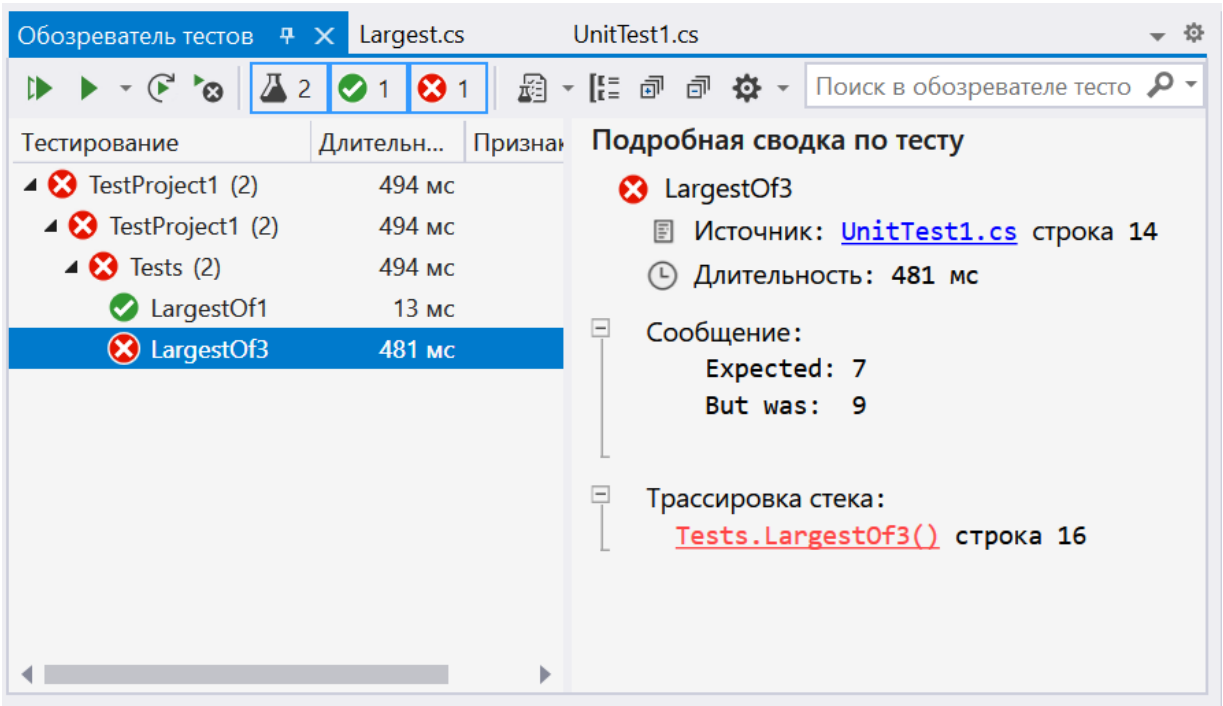


Рис. 26. Результаты тестирования (один тест не пройден)

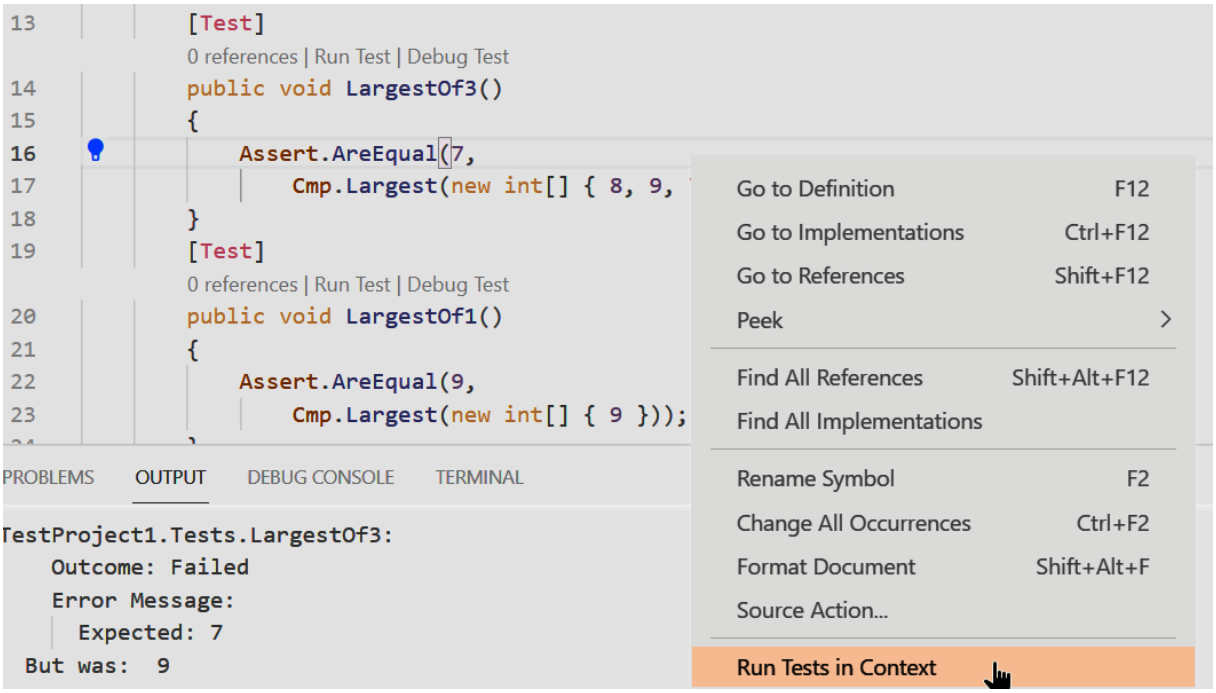


Рис. 27. Результаты тестирования (один тест не пройден) в окне VS Code

Для запуска можно выбрать конкретный тест и запустить его, либо запустить все тесты. На рисунке 28 показан подробный отчет о непройденном тесте.

Рис. 28. Результаты тестирования (один тест не пройден) в окне VS Code

При создании приложения shop в лабораторной работе № 2 django среди прочих сгенерировала файл shop/tests.py со следующим содержимым:

```
from django.test import TestCase
# Create your tests here.
```

Именно в нём предполагается размещать тесты. Давайте напишем простейший тест:

```
from django.test import TestCase
from django.urls import reverse
from .models import Product

# Create your tests here.
class IndexViewTests(TestCase):
    def test_no_products(self):
        response = self.client.get(reverse('index'))
        self.assertEqual(response.status_code, 200)
        self.assertQuerysetEqual(response.context['products'],
        [])
```

Тесты организуются в виде методов класса-наследника TestCase. django.test представляет интерфейс для написания тестов, совместимый с модулем стандартной библиотеки unittest и расширяющий его. В

представленном тесте мы проверяем, что если при пустой базе совершить запрос к url /, помеченному у нас как index в urls.py, то код ответа будет 200 и контекстная переменная products будет пустым queryset.

Для запуска тестов используется manage.py:

```
python manage.py test
```

После запуска мы увидим следующее:

```
Creating test database for alias 'default'...
System check identified no issues (0 silenced).
.
----------------------------------------------------------------------
-------
Ran 1 test in 0.012s
OK
Destroying test database for alias 'default'...
```

Как мы видим, тест прошёл успешно. Ещё можно заметить, что django создал тестовую базу данных, запустил тест и затем удалил её.

2.3.3 Порядок выполнения работы

1. Ознакомиться с созданием модульных (юнит) тестов для .NET и python с использованием выбранных библиотек модульного тестирования и среды разработки.
2. Ознакомиться с созданием юнит-тестов и использованием моков на примере приложения интернет-магазина из лабораторной работы
3. Проделать основные шаги, удостовериться в работоспособности тестов.
4. Покрыть большую часть приложение юнит-тестами.
5. Применить mock-объекты при тестировании.
6. Применить библиотеку mock-объектов.
7. Выполнить индивидуальное задание.

2.3.4 Вопросы и задания

1 Повторить и закрепить информацию о модульном тестировании и разработке, управляемой тестами, шаблонах тестирования.

2 Какие шаблоны модульного тестирования Вы знаете ?

3 Как связаны процессы рефакторинга и модульного тестирования ?

4 Приведите пример необходимости применения mock-объектов.

5 Какие библиотеки модульного тестирования Вы использовали ?

6 Какие библиотеки подставных объектов Вы использовали ?

7 В чем особенность модульного тестирования кода, взаимодействующего с базой данных ? С интерфейсом пользователя ?

2.4 Лабораторная работа № 4. Обзор библиотек Python на примере задач машинного обучения.

2.4.1 Цель работы

Цель работы — знакомство со средствами анализа данных в Python с использованием интерактивных блокнотов Jupyter.

2.4.2 Подготовка к выполнению работы

Для выполнения работы понадобятся Python 3, Jupyter и библиотеки pandas, matplotlib и scikit-learn. Есть несколько способов подготовить рабочее окружение, некоторые из них перечислены ниже.

1) Самый простой способ – использовать Google Colab, работающий в браузере [4]. Просто переходите на сайт и сразу можете приступать к написанию кода.

2) Использование Anaconda [5].

Anaconda — это всё, что вам нужно и даже больше, собранное в один установочный пакет. После установки в системе появится приложение

Jupyter Notebooks, после запуска которого у вас появится создать возможность блокнот в открывшемся окне браузера.

3) Способ для экономии дискового пространства : Miniconda [6].

Miniconda не загружает все пакеты, но потребует некоторых действий:

```
conda activate base
conda install jupyter pandas matplotlib scikit-learn
jupyter notebook.
```

После запуска Jupyter, например, из командной строки Windows командой jupyter notebook, откроется браузер с панелью выбора файлов интерактивных записных книг (рисунок 29). Содержимое файла записной книги для лабораторной работы приведено в Приложении А.

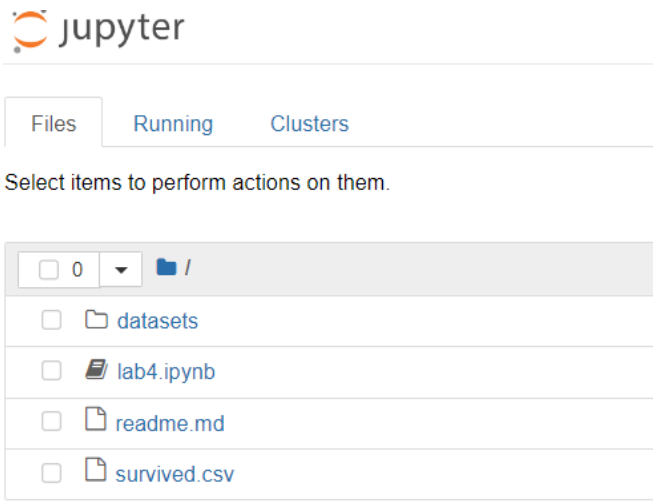


Рис. 29. Выбор файла интерактивных записных книг Jupyter

2.4.3 Порядок выполнения работы

1 Открыть интерактивный блокнот lab4.ipynb в Jupyter и ознакомиться с материалами работы [3].

2 Определить долю выживших и определите 2 наиболее важных признака для выживания при помощи DecisionTreeClassifier на основе данных из datasets/titanic.csv.

3 Для задачи о классификации вина методом ближайших соседей произведите масштабирование признаков с помощью функции

sklearn.preprocessing.scale. Снова найдите оптимальное k на кросс-валидации. Помогло ли масштабирование?

4 Определить качество линейной регрессии в примере с зарплатами при помощи метрики r2_score.

5 Выполнить индивидуальное задание.

2.4.4 Вопросы и задания

1. Что такое Jupyter Notebook ? Как его установить ? Как с ним работать ? Его назначение ?

2. Назначение и характеристика библиотеки pandas

3. Назначение и характеристики библиотеки matplotlib

4. Назначение и характеристики библиотеки NumPy

5. Назначение и особенности библиотеки scikit-learn. Какие задачи машинного обучения можно решать с ее помощью ?

6. Назначение и особенности TenzorFlow ?

7. Какова роль каждой из рассмотренных библиотек при решении задач машинного обучения и анализа данных ?

8. При подготовке к лабораторной работе необходимо познакомиться с задачами машинного обучения и методами их решения, используя в том числе литературу из приведенного списка.

3 ВЫПОЛНЕНИЕ КУРСОВОЙ РАБОТЫ

3.1. Задание на курсовую работу

На курсовую работу студенту выдается индивидуальное задание (по вариантам), заключающееся в итеративной разработке по варианту простого прикладного программного приложения бизнес-аналитики либо машинного обучения / анализа данных с использованием изучаемых в дисци-

плине технологий (итеративное планирование, прецедентный анализ, UML, модульные тесты, рефакторинг, паттерны проектирования, каркасы MVC).

3.2. Примерное содержание курсовой работы

1. Титульный лист.
2. Формулировка варианта задания.
3. Основная часть, включающая (для варианта бизнес-аналитики):

1) описание требований к приложению (состав функций);

2) описание используемых средств разработки, технологий, библиотечных функций и классов;

3) итеративный план разработки;

4) прецедентный анализ;

5) результаты моделирования предметной области;

6) структуру базы данных приложения;

7) архитектуру приложения и используемые архитектурные шаблоны;

8) диаграммы классов, диаграммы взаимодействия (если есть);

9) модульные тесты;

10) описание примененных паттернов проектирования;

11) описание переработки (реинжиниринга) кода с использованием и без использования паттернов проектирования;

12) экранные формы работы приложения и результаты его работы;

13) список использованных источников;

14) коды программы (в приложении).

Для варианта с приложением, использующим методы машинного обучения или анализа данных, допускается не приводить часть перечисленных пунктов, в частности, 3, 5, 6, а пункты, связанные с рефакторином, паттер-

нами и тестами приводить для служебных и интерфейсных компонентов приложения.

3.3. Примерные варианты заданий курсовой работы

Ниже приведены возможные варианты заданий – предметная область и небольшие уточнения по обрабатываемым данным и их типам для проектирования иерархии классов (для бизнес-аналитики) :

1. Комплектующие ПК : конфигурации, цена, мощность (разные типы деталей).

2. Успеваемость студентов: баллы по лисциплинам и средний балл (разные типы дисциплин).

3. Заказы в интернет-магазине : среднее количество и цена (разные типы товаров).

4. Зарплаты списка сотрудников за год (должности сотрудников / варианты оплаты, типы начислений).

5. Количество постов по аккаунтам в соцсетях с датами, лайками и дизлайками.

6. Составы семей (взрослые / дети / пенсионеры / работающие / учащиеся) с данными о доходах и расходах (семейный бюджет).

7. Модели и комплектации автомобилей (тип автомобиля, модель, двигатели, коробка, опции, мощность, стоимость).

8. Расчет турпутевок (направления, длительность, транспорт, проживание, стоимость).

9. Публикации сотрудников вуза (тип, дата, объем, индексация в наукометрических базах).

10. Штатное расписание кафедры (должность, степень, звание, ставка, заработная плата).

11. Мероприятия, проводимые вузом (тип, количество участников,

дата, место, прочее).

12. Соревнования спортивные (тип, участники, результаты).

13. Чемпионаты спортивные (вид спорта, участники, результаты, расписание, таблица).

14. Турниры по олимпийской системе (вид спорта, участники, турнирная сетка).

15. Командировки (кто, куда, срок, список расходов, типы расходов).

16. Населенные пункты, районы, округа (название, тип пункта, подчинение, количество жителей, бюджет, глава).

По каждому варианту предусмотреть расчет статистики.

Также возможны варианты построения клиент-серверной системы, решающей задачи из области машинного обучения и анализа данных :

17. Задача машинного обучения в качестве серверной компоненты.

18. Задача применения ранее обученной модели в качестве серверной компоненты.

19. Задача анализа данных в качестве серверной компоненты.

20. Задача подготовки / фильтрации / преобразования данных в качестве серверной компоненты.

3.4 Примеры выполнения курсовой работы

3.4.1 Примеры выполнения на платформе .NET / .NET Core

В данном примере (в рамках которого приведены разные варианты реализации) рассматривается небольшое приложение на платформе .NET / .NET Core для работы со списком объектов из иерархии простых двумерных фигур. Приложение работает с разными источниками данных, включая текстовый файл, базу данных SQL Server Express LocalDB, а также константный источник (который можно рассматривать

как заглушку - мок). Технологии реализации интерфейса пользователя: Windows Forms, WPF, консоль, работы с БД - EntityFramework.

Используется иерархия интерфейс – базовый абстрактный класс - наследники (в C#), несколько простых паттернов GoF (Адаптер, Стратегия/Полиморфизм, Наблюдатель), элементы многослойности (архитектурный шаблон Слои), модульные тесты, также реализованные самостоятельно (без использования готовых каркасов) паттерны MVP, MVC и паттерн MVVM, используемый в WPF.

3.4.1.1 Постановка задачи, варианты использования и планирование

Необходимо разработать небольшое приложение с GUI (а также консольным интерфейсом и в перспективе – веб сервис и веб-приложение) для обработки списка фигур с определением суммарной площади и суммарной длины периметра фигур в группе.

Приложение должно поддерживать разные источники данных, в том числе БД SQL Server и текстовый файл.

Варианты использования (прецеденты / истории) :

П1 Добавление фигуры в список на GUI-форме.

П2 Расчет общей площади и длины периметра фигур в списке.

П3 Загрузка списка фигур из текстового файла.

П4 Загрузка тестового списка фигур.

П5 Загрузка большого тестового списка фигур.

П6 Реализация консольного интерфейса.

П7 Загрузка списка фигур из базы данных.

П8 Расчет параметров фигуры / списка фигур в веб-приложении.

При загрузке списка фигур необходима обработка в зависимости от его размера – для небольших списков он выводится целиком, для средних программа запрашивает пользователя о необходимости загрузки, большие списки не выводятся, а только обрабатываются.

Таблица 1 – План версий и итераций

История	Сложность (идеальных дней)	№ итерации	№ версии
П1	1	1	1
П2	1	1	1
П3	0,5	2	1
П4	0,5	2	1
П5	1	2	1
П6	1,5	3	1
П7	2	4	2
П8	2	5	3

3.4.1.2 Первая версия проекта: GUI на Windows Forms и консольное приложение с использованием паттерна MVP

3.4.1.2.1 Общая архитектура и паттерны

В данном случае мы стараемся использовать выделенные классы Модели, Вида и Презентер, который управляет Видом через интерфейс Вида. В результате мы получаем возможность реализовать два разных приложения (консольное и графическое) на базе одного и того же набора классов, а также получаем возможность тестирования не только модели, но и логики Презентера. По ходу разработки используется рефакторинг, а в самом проекте помимо паттерна MVP применяются такие паттерны GoF, как Адаптер, Полиморфизм (Стратегия) и Наблюдатель.

Базовая иерархия классов модели показана на рисунке 30.

В этой иерархии выделяется интерфейс IShape, базовый класс с основными атрибутами (их 2 : a и b), конструктором, методом вывода в строку, а также две конкретных фигуры. Также реализован класс списка фигур

ShapeList, собирающий статистику по фигурам (общая площадь и суммарная длина периметров), не зная об их типах. Это уже можно считать примером применения паттерна Полиморфизм (Стратегия), где стратегией является определение площади и периметра фигуры.

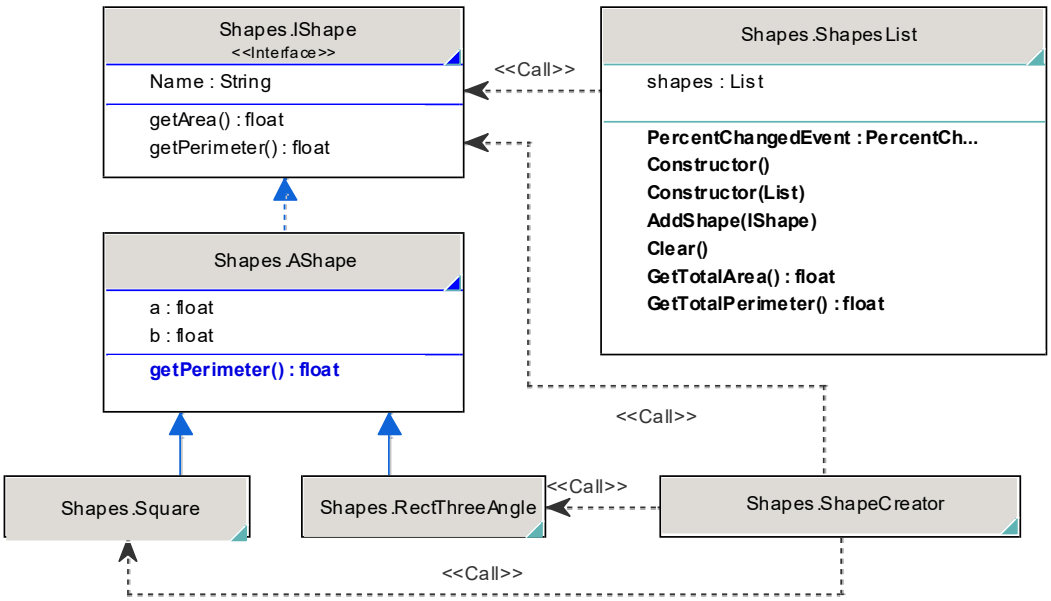


Рис. 30. Базовая иерархия классов фигур, списка фигур и создателя фигур

Дополнительно реализован прототип фабрики – ShapeCreator, создающий фигуры с интерфейсом IShape из строк. Этот класс в будущем можно снабдить интерфейсом и получить паттерн Абстрактная Фабрика. Также здесь реализуется интерфейс для Наблюдателя в виде события изменения процента выполненной работы на один процент, на который подписываются делегаты в наблюдателях (в данном случае это форма, наблюдатель используется для отображения прогресс-бара в форме).

Диаграмма классов, связанных с Презентером, показана на Рисунке 31. Тут выделен Presenter, который управляет видами через интерфейс IView. Презентер управляет видом, также он работает с источниками данных IDataSource. При этом он может использовать любой IDataSource, а также знает о трех конкретных IDataSource, реализованных в программе (кон-

стантный, источник большого списка для тестирования Наблюдателя и прогресс-бара и источник, читающий список фигур из CSV-файла). Это тоже своего рода Стратегия (или Абстрактный сервер) – Рисунок 32.

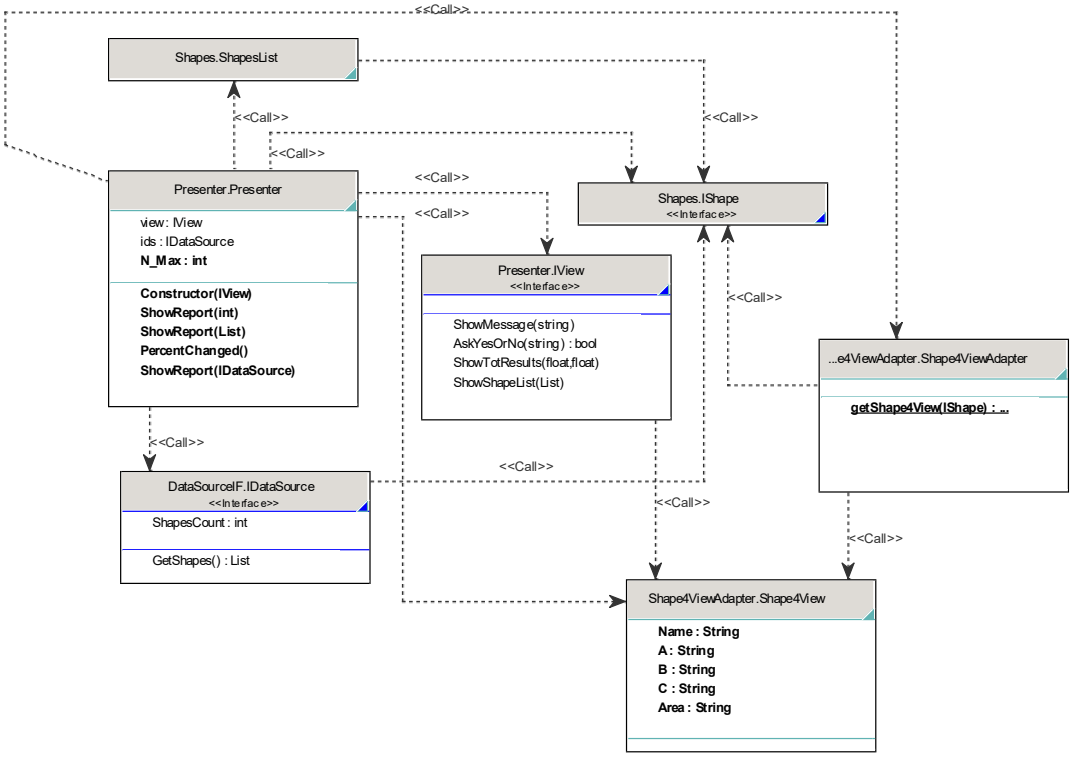


Рис. 31. Презентер и связанные с ним классы

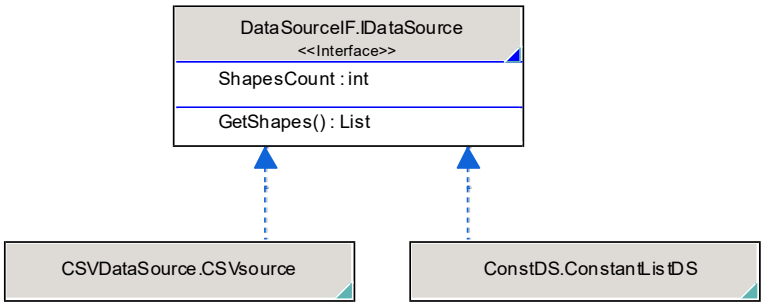


Рис. 32. Реализации источника данных

Помимо этого Презентер использует Адаптер Shape4ViewAdapter для конвертирования фигур в представление Shape4View, в котором нуждается

форма (GUI), что можно в дальнейшем развить во что-то, похожее на паттерн MVVM.

Исходный код примера (без кода формы) приведен в Приложении Б.

В данном примере не показан итеративный характер разработки в первых итерациях согласно плану, приведенному выше, но в самой работе необходимо это сделать, показав очередность прохождения итераций, в частности – создание и тестирование классов предметной области, применение паттернов при реализации очередной новой функции, рефакторинг в процессе добавления новых функций и так далее.

3.4.1.2.2 Модульные тесты

Модульные тесты позволяют тестировать как классы модели, так и класс презентера с помощью подставных вида и источника данных, в данном случае реализованных вручную (используется библиотека MSTest).

Тест для базовых фигур :

```
[TestClass]
public class ShapeTests {
    [TestMethod]
    public void TestSquare()   {
        Square sq = new Square(3, 4);
        Assert.AreEqual(3 * 4, sq.getArea());
    }

    [TestMethod]
    public void TestRectThreeAngle()   {
        RectThreeAngle tri = new RectThreeAngle(3, 4);
        Assert.AreEqual(3 * 4 / 2, tri.getArea());
        Assert.AreEqual(3 + 4 + tri.C, tri.getPerimeter());
    }
```

Тест для списка форм :

```
[TestMethod]
public void TestShapeList() {
    RectThreeAngle tri = new RectThreeAngle(3, 4);
    Square sq = new Square(2, 2);
    ShapesList sh = new ShapesList();
    sh.AddShape(tri);
    sh.AddShape(sq);
    Assert.AreEqual(tri.getArea() + sq.getArea(),
                h.GetTotalArea());
    Assert.AreEqual(tri.getPerimeter() +
          q.getPerimeter(), sh.GetTotalPerimeter());
}
```

Подставной вид (MockView) использует шаблон тестирования Строка журнала в виде строки Log :

```
class MockView : IView {
    public String Log { get; set; }
    public bool Yes { get; set; }
    public List<String> Shapes { get; set; }

    public MockView()  {
        Log = ""; Yes = true;
        Shapes = new List<string>();
    }

    public bool AskYesOrNo(string quest) {
        Log += "AskYesNo(" + quest + ");";
        return Yes;
    }

    public void ShowMessage(string msg)  {
        Log += "ShowMessage(" + msg + ");"; }
```

```
    public void ShowShapeList(List<Shape4View> shapes)  {
        int cnt = shapes == null ? 0 : shapes.Count;
        Log += "ShowShapeList() - " + cnt + " items;";
        foreach(Shape4View shape in shapes) {
          Shapes.Add(shape.Name + ":" + shape.A + " " + shape.B
           + " " + shape.C + " area = " + shape.Area + " per = "
           + shape.Per);
        }
    }

    public void ShowTotResults(float area, float perimeter) {
        Log += "ShowTotResults(" + area + "," + perimeter + ");";
    }

    public void OneMorePercent() { }
 }
```

Подставной источник данных :

```
class MockDataSource : IDataSource {
    List<IShape> shapes;
    public MockDataSource(List<IShape> shl)  {
        shapes = shl;
    }
    public int ShapesCount {
        get { return shapes.Count; } set { }
    }
    public List<IShape> GetShapes() {
        return shapes;
    }
    public List<String> GetShapesStr() {
        return new List<string>();
    }
}
```

Тест для Презентера (проверятся логика его работы) :

```
[TestClass]
public class PresenterUnitTest {
   [TestMethod]
   public void TestPresenter() {
         List<IShape> shl = new List<IShape>();
         shl.Add(new Square(1, 1));
         shl.Add(new Square(2, 2));
         MockDataSource ds = new MockDataSource(shl);
         MockView mv = new MockView();
         Presenter.Presenter pres = new Presenter.Presenter(mv);
         pres.ShowReport(ds);

         Assert.AreEqual("ShowTotResults(" + (5.0).ToString() +
     "," + (12.0).ToString() + ");ShowShapeList() – 0 items;",
                mv.Log);
   }
}
```

3.4.1.2.3 GUI и консольное приложения

Сама GUI – программа выглядит, как показано на рисунке 33.

В приложении реализуется работа со списком фигур из файла, из большого списка, реализованного в памяти (при работе с ним активируется прогресс-бар, использующий механизм Наблюдателя, реализованный через события и делегаты C#), по умолчанию (небольшой список в памяти), а также – из списка, формируемого вручную в форме (при выборе активируется кнопка Добавить, можно использовать комбо-бокс и поля ввода a и b, и наборот, отключается кнопка работы со списками, формируемыми не в форме, Загрузить и посчитать).

Форма реализует интерфейс IView, но кода в ней мало, т.к. основные функции находятся в коде презентера, источников данных и модели.

В основном методе Main класса программы Program происходит привязка формы к Презентеру:

```
frmShapes frm = new frmShapes();
frm.Pres = new Presenter.Presenter(frm);
Application.Run(frm);
```

Так же можно сделать настройку на источники данных, однако в данном случае это делается в самом Презентере.

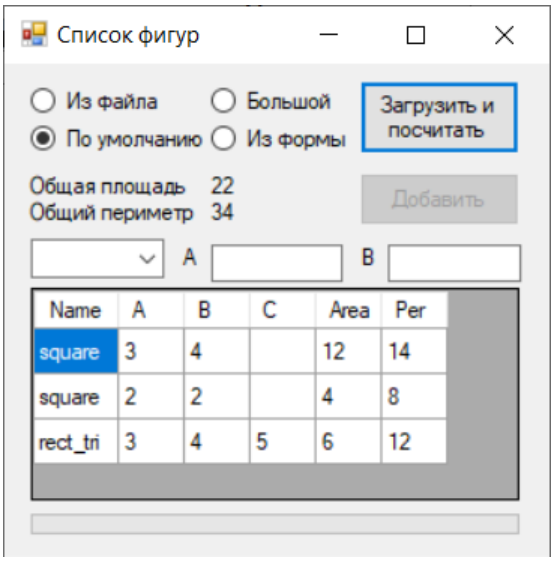


Рис. 33. GUI программа в работе

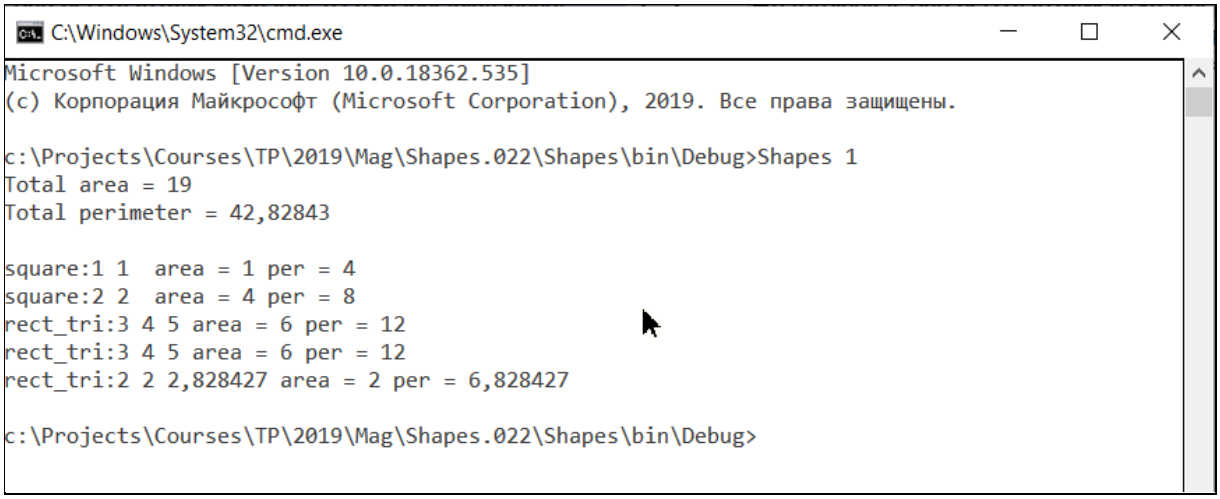


Рис. 34. Консольное приложение в работе

Консольное приложение, использующее те же классы (но с собственной реализацией IView), выглядит в работе как на рисунке 34.

3.4.1.2.4 Рефакторинг в ходе реализации первой версии

В ходе работы выполнялся рефакторинг (улучшение кода без изменения функциональности), в частности, выделение методов, переименование, упрощение условных выражений и т.д. При описании рефакторинга в работе необходимо придерживаться стиля, приведенного в [1], то есть использовать таблицы, явно указывая, какой рефакторинг выполнялся, приводить фрагменты кода до и после рефакторинга, например, как показано в таблице 2.

Таблица 2 – Примеры описания рефакторинга

<table>
<tr><td colspan="2">Выделение метода</td></tr>
<tr><td>До</td><td>После (в данном случае выделен
метод – свойство C)</td></tr>
<tr><td><pre>public override float getPerimeter()
{
   return a + b +
    (float)Math.Sqrt(a * a + b * b);
}</pre></td><td><pre>public override float getPerimeter()
{
    return a + b + C;
}

public float C {
  get {
      return (float)Math.Sqrt(a * a
              + b * b);
  }
}</pre></td></tr>
<tr><td colspan="2">Переименование</td></tr>
<tr><td>До</td><td>После</td></tr>
<tr><td><pre>Square</pre></td><td><pre>Rectangle</pre></td></tr>
</table>

3.4.1.3 Пример реализации с использованием паттерна MVC

Паттерн MVC отличается от паттерна MVP в основном тем, что в MVP Presenter явно управляет видом через интерфейс IView, вся информация в вид из модели поступает только от Presenter. В MVC возможны разные варианты, но в целом поддерживается непосредственное обновление вида при изменении моделей, что реализуется с помощью паттерна Наблюдатель (как один из вариантов). В данном пункте рассмотрена реализация варианта MVC, показанного на рисунке 4.

Для данной реализации Наблюдателя используются механизмы событий и делегатов C#.

Вместо Презентера реализуется Контроллер. Для реализации обновления вида по модели реализуем отдельный класс модели ObservableShapesList, за которым наблюдает форма (вид), в нем описаны события, на которые должна быть подписана форма :

```
// Адаптер для ShapeList, делающий его наблюдаемым объектом ...
public class ObservableShapesList {
   ShapesList sl;
   public bool HandlePercents { get; set; }
   public TotalsHandler TotalsEvent;
   public ListHandler ListEvent;
   public PercentChangedHandler PercentEvent;

   public void PercentChangedHandler()  {
       PercentEvent();
   }

   public ObservableShapesList(ShapesList _sl)  {
       sl = _sl; HandlePercents = false;
   }
```

```
        public void GetTotals() {
            if (HandlePercents)
              sl.PercentChangedEvent += PercentChangedHandler;
            TotalsEvent(sl.GetTotalArea(), sl.GetTotalPerimeter());
        }

        public void GetList() {
            List<Shape4View> l4view = new List<Shape4View>();
            foreach (IShape sh in sl.GetList())  {
              l4view.Add(
Shape4ViewAdapter.Shape4ViewAdapter.getShape4View(sh));
            }
            ListEvent(l4view);
        }
    }
```

В самом контроллере предусмотрены обработчики, которые должна задать форма или головная программа :

```
    public bool LShowBigLists { get; set; }  // Вместо диалога в MVP
    public TotalsHandler TH { get; set; }
    public ListHandler LH { get; set; }
    public PercentChangedHandler PH { get; set; }
```

Привязка к наблюдаемому объекту осуществляетя также в коде контроллера :

```
    // Объект НАБЛЮДАЕМОГО списка
    ObservableShapesList sl =
            new ObservableShapesList(new ShapesList(_sl));
    sl.PercentEvent += PH;
    sl.TotalsEvent += TH;
    sl.ListEvent += LH;
```

Подписывание форм на события осуществляется в данном случае в головной программе (поле формы Ctrl - контроллер):

```
frmShapes frm = new frmShapes();
frm.Ctrl = new ShapesController.Controller();
frm.Ctrl.PH += frm.OneMorePercent;
frm.Ctrl.TH += frm.ShowTotResults;
frm.Ctrl.LH += frm.ShowShapeList;
Application.Run(frm);
```

Внешний вид и поведение программы аналогичны варианту с паттерном MVP.

3.4.1.4 Вторая версия - реализация GUI-приложения на C# (.NET Core) с использованием WPF, ORM каркаса EntityFramework и трех разных паттернов : MVC, MVP и MVVM

Во второй версии рассматривается, во-первых, работа с БД с помощью современной ORM библиотеки (каркаса) Entity Framework, а также более современная технология создания GUI – Windows Presentation Foundation (WPF), основанная на той же по сути разметке XAML, что и проекты на Xamarin. На данном примере также показана реализация паттерна MVVM, характерного для проектов на WPF, а также условная реализация MVC и MVP в контексте этого проекта WPF (аналогичная показанным выше).

3.4.1.4.1 Модификация базовой модели классов предметной области

По сравнению с примером из первой версии в данном случае немного модифицирована сама объектная модель ядра приложения, в частности, выделены другие пространства имен, добавлена абстрактная фабрика, добавлен еще один абстрактный класс для фигур с двумя параметрами и еще

она стандартная фигура – квадрат (Square), а прямоугольник назван Rectangle. Это сделано, в частности, чтобы показать работу с подобными иерархиями в ORM библиотеке. Все подобные изменения при оформлении работы можно отразить в списке рефакторингов по аналогии с таблицей 2.

Базовые интерфейсы и абстрактные классы :

```
public interface IShape {
    int Id { get; set; }
    String Name { get; }
    String Value { get; }
    float getArea();
    float getPerimeter();
}

public abstract class AShape : IShape {
    public int Id { get; set; }
    public abstract String Name { get; }
    public abstract float getArea();
    public abstract float getPerimeter();
    public abstract string ToString2();
    public override string ToString()     {
        return Name + ": " + ToString2() +
            " area = " + getArea().ToString() +
            " perimeter = " + getPerimeter().ToString();
    }

    public String Value     {
        get => ToString2() +
            " area = " + getArea().ToString() +
            " perimeter = " + getPerimeter().ToString();
    }
}
```

```
public abstract class ABShape : AShape {
    protected float a;
    protected float b;
    public ABShape() { }
    public ABShape(float _a, float _b)  {
        a = _a;
        b = _b;
    }
    public float A { get => a; set { a = value; } }
    public float B { get => b; set { b = value; } }
    public override string ToString2() {
        return " a = " + a.ToString() + " b = " + b.ToString();
    }
}

public interface IShapeFactory {
   IShape CreateShapeFromString(String src);
}
```

Базовый класс прямоугольник :

```
public class Rectangle : ABShape {
    public Rectangle()  { }
    public Rectangle(float _a, float _b) : base(_a, _b) { }
    public override String Name { get => "Rectangle"; }
    public override float getArea() {
        return a * b;
    }
    public override float getPerimeter()  {
        return 2 * (a + b);
    }
}
```

Код создания объектов по строковым кодам перенесен в реализацию абстрактной фабрики :

```
namespace BasicShapes {
    public class BasicShapesCSFactory : IShapeFactory {
        public float A { get; set; }
        public float B { get; set; }
        public IShape CreateShapeFromString(String src) {
            IShape shape = null;
            String[] arr = src.Split(';');
            try   {
                if (arr[0] == "0")
                    shape = new Rectangle(float.Parse(arr[1]),
                        float.Parse(arr[2]));
                else if (arr[0] == "1")
                    shape = new RectThreeAngle(
                         float.Parse(arr[1]),
                        float.Parse(arr[2]));
                else if (arr[0] == "2")
                    shape = new Square(float.Parse(arr[1]));
                A = float.Parse(arr[1]);
                B = float.Parse(arr[2]);
            }
            catch { }
            return shape;
        }
    }
}
```

3.4.1.4.2 Реализация GUI с помощью WPF

GUI реализован в виде XAML на WPF. Саму разметку тут приводить не будем, так как она делается с помощью редактора форм, но отметим, что внешний вид приложения немного отличается – см. рисунок 35. В данной версии программы не реализована поддержка ввода данных из формы

(кроме добавления фигуры), вместо этого добавлена поддержка работы с БД через Entity Framework (загрузка данных и добавление объектов).

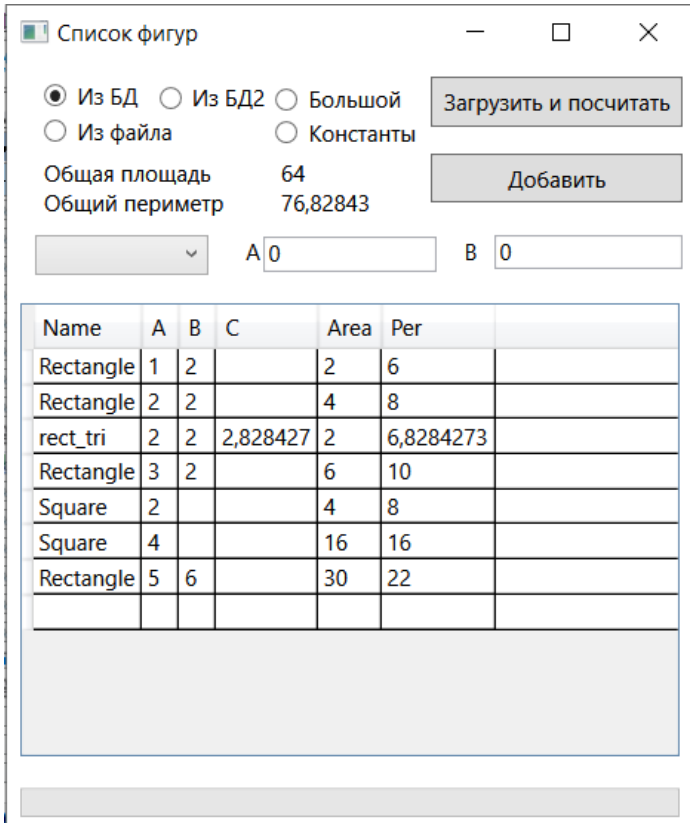


Рис. 35. Внешний вид приложения на WPF

3.4.1.4.3 Реализация еще одного источника данных с помощью ORM библиотеки Entity Framework

Остановимся на реализации источника данных с помощью Entity Framework. В данном случае используется наследник стандартного библиотечного класса DbContext и подход Code First :

```
namespace DbLibEF  {
public class ShapesDbContext : DbContext, IDataSource   {
    DbSet<Rectangle> Rectangles { get; set; }
    DbSet<RectThreeAngle> RectThreeAngles { get; set; }
    DbSet<Square> Squares { get; set; }
```

```
        public ShapesDbContext()  {
            Database.EnsureCreated();
        }

        protected override void OnConfiguring(DbContextOptionsBuilder
optionsBuilder) {
     optionsBuilder.UseSqlServer(
"Server=(localdb)\\mylocaldb;Database=shapesdb;Trusted_Connection=True;");
        }

       protected override void OnModelCreating(ModelBuilder
                modelBuilder) {
            modelBuilder.Entity<Rectangle>().Ignore(b => b.Value);
            modelBuilder.Entity<Square>().Ignore(b => b.Value);
            modelBuilder.Entity<RectThreeAngle>().Ignore(b =>
               b.Value);
            modelBuilder.Entity<Rectangle>().Property(p =>
               p.Id).ValueGeneratedNever();
            modelBuilder.Entity<Square>().Property(p =>
               p.Id).ValueGeneratedNever();
            modelBuilder.Entity<RectThreeAngle>().Property(p =>
               p.Id).ValueGeneratedNever();
        }

        public List<IShape> GetShapes()   {
            List<IShape> shapes = new List<IShape>();
            shapes.AddRange(Rectangles.ToList());
            shapes.AddRange(RectThreeAngles.ToList());
            shapes.AddRange(Squares.ToList());
            return shapes.OrderBy<IShape, int>(x =>
               x.Id).ToList<IShape>();
        }
```

```
        public void AddShape(string par)  {
              BasicShapesCSFactory f = new BasicShapesCSFactory();
              IShape sh = f.CreateShapeFromString(par);
              sh.Id = Rectangles.Count() + Squares.Count() +
                 RectThreeAngles.Count() + 1;
              if (sh.Name == "Rectangle")
                 Rectangles.Add((Rectangle)sh);
              if (sh.Name == "Square") Squares.Add((Square)sh);
              if (sh.Name == "rect_tri")
                  RectThreeAngles.Add((RectThreeAngle)sh);
              SaveChanges();
        }
        public int ShapesCount { get; set; }
        }
    }
```

На что тут необходимо обратить внимание :

1. Использованы 3 репозитория для трех базовых фигур. Несмотря на наследование, т.к. используется базовый абстрактный класс, нельзя применить сохранение всех сущностей в одной таблице, хотя EF это поддерживает (для обычного наследования).

2. В связи с пунктом 1 для уникального id каждого объекта они генерируются исходя из общего количества записей во всех трех репозиториях:

```
sh.Id = Rectangles.Count() + Squares.Count() +

   RectThreeAngles.Count() + 1;
```

и затем вносятся программно (а не генерируются автоматически), для этого предварительно при настройке модели устанаввивается соответствующий признак :

```
modelBuilder.Entity<Rectangle>().Property(p =>
                p.Id).ValueGeneratedNever();
```

3. Для сохранения вызывается библиотечный метод SaveChanges().

4. По умолчанию классы-сущности (Entities) должны удовлетворять некоторым т.н. соглашениям, в частности, должны содержать публичные свойства, которые отоюражаются на поля таблиц БД, содержать пустые конструкторы и т.д. В частности, поэтому такие члены классов добавлены в сущности – фигуры выше.

5. Все дополнительные свойства модели (помимо т.н. соглашений) устанавливаются с помощью метода OnModelCreating() - в нем используется API Fluent, например :

```
modelBuilder.Entity<RectThreeAngle>().Ignore(b => b.Value);
modelBuilder.Entity<Rectangle>().Property(p =>
                p.Id).ValueGeneratedNever();
```

6. В качестве БД используется SQL Server LocalDB – локальная база, использующая тот же движок и библиотеки доступа, что и SQL Server Express. Также она поддерживает и Entity Framework. Она заменила в значительной степени SQL Server Compact. Для использования ее в работе нужно вручную запустить сервер LocalDb из командной строки (!), также можно работать с его базами и таблицами через меню VS Вид – Обозреватель объектов SQL Server.

7. Класс ShapesDbContext помимо наследования от DbContext реализует интерфейс IDataSource, то есть два его метода (запрос списка фигур и сохранение фигуры) и свойство ShapesCount.

Как видно из данного примера, вполне возможно использовать EntityFramework в сочетании не с чисто анемичными классами-сущностями (то есть с классами данных), а с обычными классами предметной области, у которых есть содержательные методы.

3.4.1.4.4 Реализация паттерна MVVM

В данной версии также рассматривается более характерный для WPF вариант паттерна MVVM (рисунок 5).

Фактически главным отличием от MVP или MVC в данном случае является использование DataBinding в коде XAML разметки, в результате чего в форме практически нет никакого кода, а весь код содержится в классе ViewModel, публичные свойства которого (включая команды) непосредственно синхронизируются с XAML, например :

```
<DataGrid ItemsSource="{Binding Shapes}" …
...
<Label Content="{Binding TotArea, UpdateSourceTrigger=PropertyChanged}"
```

Сам код класса ShapesViewModel достаточно громоздкий, так как в нем явно прописаны теперь все публичные свойства, синхронизируемые с разметкой, например :

```
public float TotPerimeter { get {  return totPerimeter; }
  set {
        totPerimeter = value;
        OnPropertyChanged("TotPerimeter");
  }
}
```

Класс наследует публичный интерфейс .NET InotifyPropertyChanged, необходимый для реализации привязки (Binding) в WPF :

```
public event PropertyChangedEventHandler PropertyChanged;
public void OnPropertyChanged([CallerMemberName] string prop = "") {
     if (PropertyChanged != null)
          PropertyChanged(this, new PropertyChangedEventArgs(prop));
}
```

Метод OnPropertyChanged вызывается везде, где необходимо синхронизировать данные (см. выше).

Для управления диалогом ViewModel класс как и раньше использует интерфейс, схожий с IView, который в данном случае называется IDialogService :

```
public interface IDialogService {
     void ShowMessage(string msg);
     bool AskYesOrNo(string quest);
}
```

а для вызова команд (выполнения команд), инициируемых видом (разметкой XAML) он также использует стандартный интерфейс ICommand (из System.Windows.Input) и его реализацию :

```
public class Command : ICommand {
       private readonly Action action;

       public Command(Action action) {
            this.action = action;
       }

       public bool CanExecute(object parameter)   {
            return true;
       }

       public event EventHandler CanExecuteChanged;

       public void Execute(object parameter)      {
                 action();
       }
}
```

Вот так реализуются свойства – команды (шаблон Команда) :

```
public ICommand AddCommand {
    get {
        return new Command(() => AddShape(ShapeType, A, B));
    }
}
```

3.4.1.4.5 Итоговые диаграммы классов, пакетов и оценка упаковки проекта с паттерном MVVM

Итоговая диаграмма классов проекта с паттерном MVVM (без части классов предметной области) приведена на рисунке 36, а диаграмма пакетов – на рисунке 37.

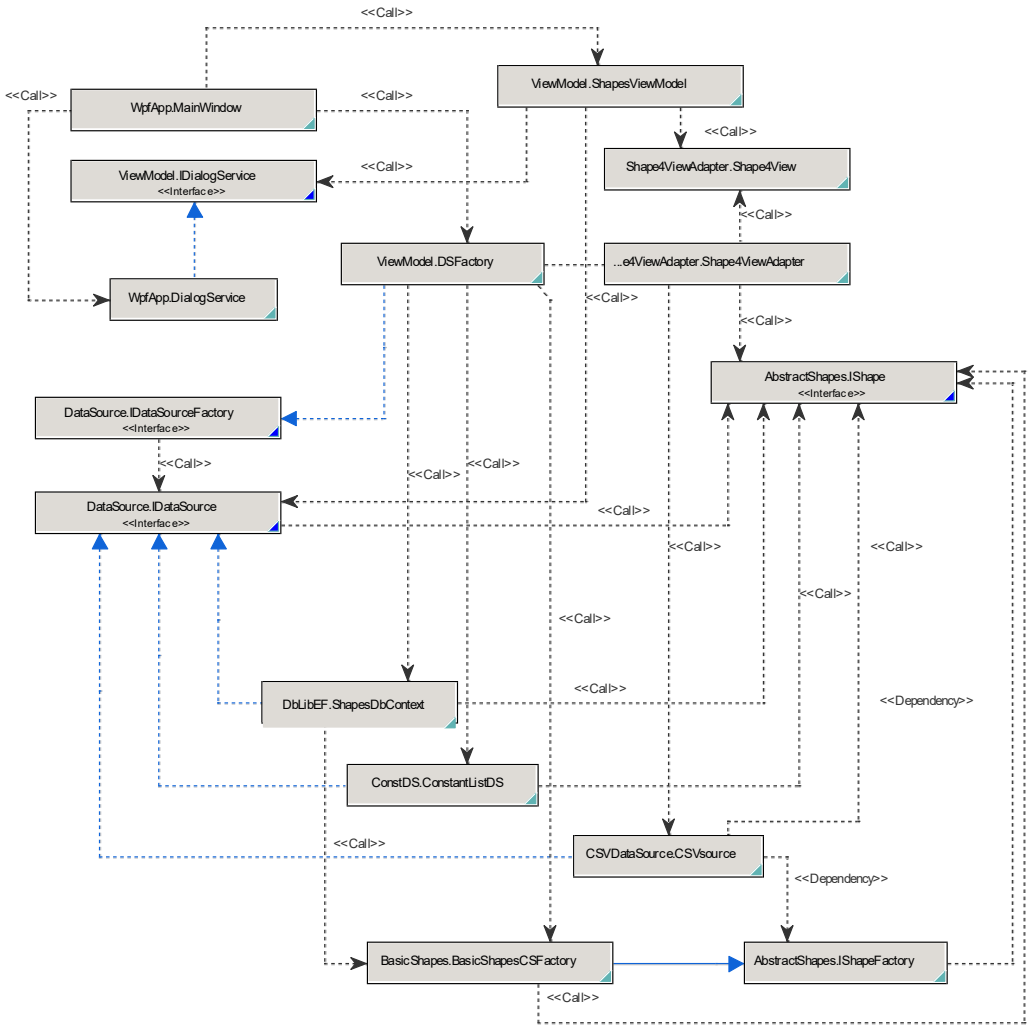


Рис. 36. Основные классы проекта Shapes с MVVM

По диаграммам и коду можно рассчитать метрики упаковки [2], приведенные в таблице 3.

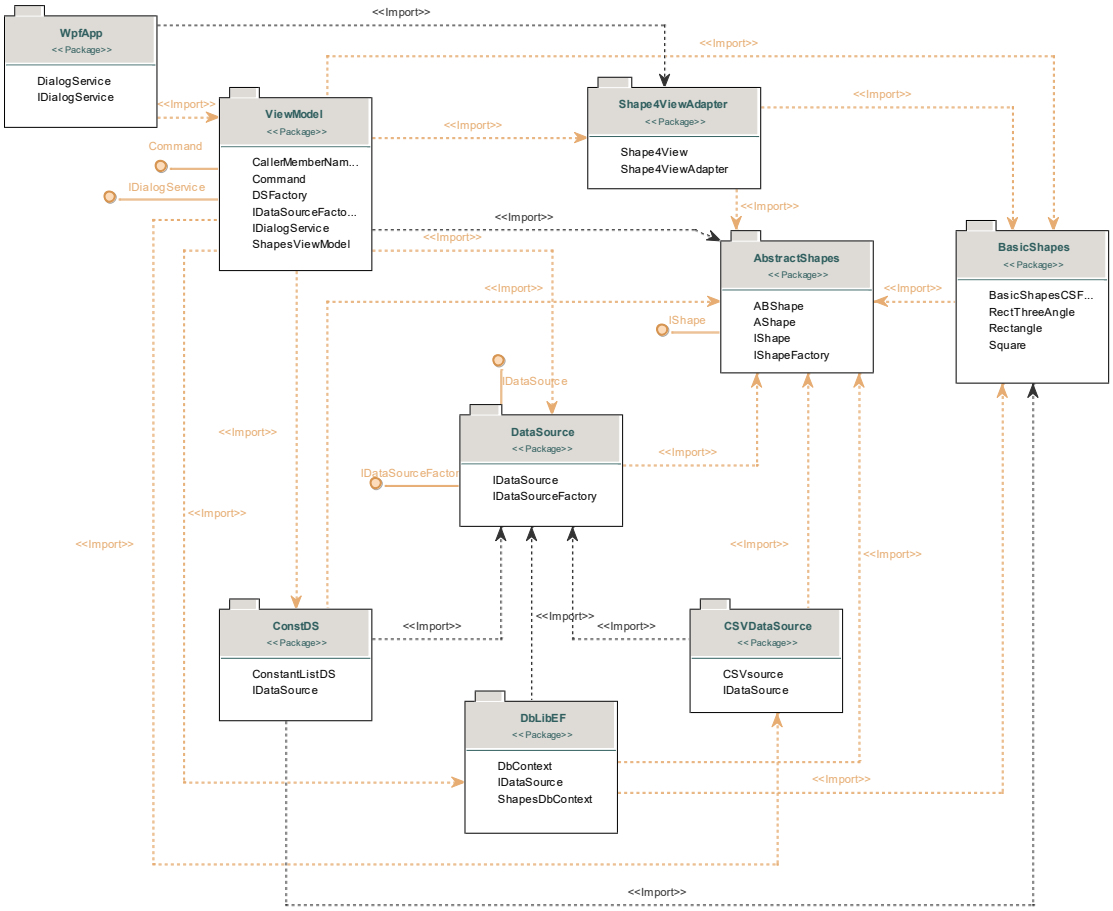


Рис. 37. Диаграмма пакетов проекта Shapes с MVVM

Таблица 3 – Метрики упаковки

Пакет	N	NA	Ca	Ce	R	H	I	A	D	D'
WpfApp	2	0	0	4	1	1	1	0	0	0
ViewModel	4	1	5	6	3	1	0,55	0,25	0,144635	0,2045
Shape4ViewAdapter	2	1	2	2	1	1	0,5	0,5	0	0
AbstractShapes	4	2	6	0	3	1	0	0,5	0,353553	0,5
BasicShapes	4	0	4	4	3	1	0,5	0	0,353553	0,5
DataSource	2	2	4	1	1	1	0,2	1	0,141421	0,2
ConstDS	1	0	1	5	0	1	0,83	0	0,117851	0,1667
CSVDataSource	1	0	1	2	0	1	0,67	0	0,235702	0,3333
DBLibEF	1	0	1	5	0	1	0,83	0	0,117851	0,1667
ShapesHandler	1	0	1	1	0	1	0,5	0	0,353553	0,5

На рисунке 38 метрики изображены на графике со значениями неустойчивости по оси X и абстрактности по оси Y, из графика видно, что

бо́льшая часть пакетов имеет подходящие метрики, так что упаковку можно считать более-менее приемлемой.

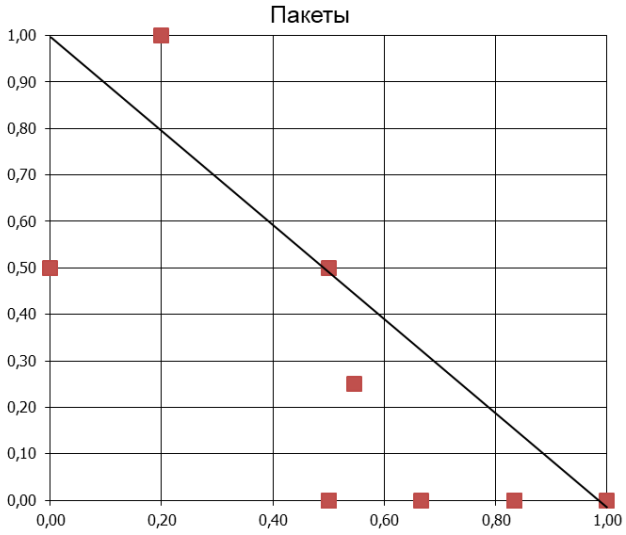


Рис. 38. Метрики упаковки (по оси X – I, по оси Y – A)

3.4.1.4.6 Особенности реализации MVC и MVP для данного варианта проекта

В данном случае MVC и MVP варианты можно реализовать на базе MVVM путем соответствующей модификации ViewModel по аналогии с описанными выше вариантами. Это необходимо в основном для того, чтобы потом реализовать ASP.NET Core WebAPI и MVC проекты на базе имеющегося кода.

3.4.1.5 Реализация веб-интерфейса на базе ранее рассмотренного варианта с использованием WebAPI и ASP.NET Core MVC

Для реализации WebAPI мы просто повторно используем ранее созданные контроллеры и делаем обертку (адаптер), унаследованный от базового библиотечного класса ControllerBase :

```
namespace ShapesWebApi {
    [Route("api/[controller]")]
    [ApiController]
    public class WebController : ControllerBase {
        ShapesController.Controller ctrl;

        public WebController() {
            ctrl = new ShapesController.Controller(new
                DSFactory(),   OneMorePercent);
        }

        public void OneMorePercent() { }

        [HttpGet("{id}")]
        public ActionResult<List<Shape4View>> Get(int id) {
            ctrl.CalcReport(id);
            return new
  ActionResult<List<Shape4View>>(ctrl.GetList().ToList());
        }
    }
}
```

На рисунке 39 показан ответ веб-сервиса на один из запросов, а на рисунке 40 – вид html-формы при обращении к нему.

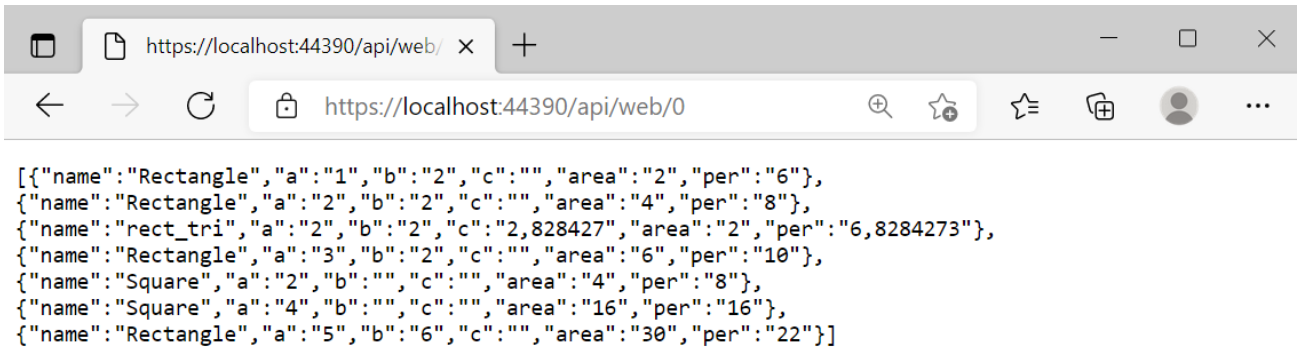


Рис. 39. Пример ответа веб-сервиса

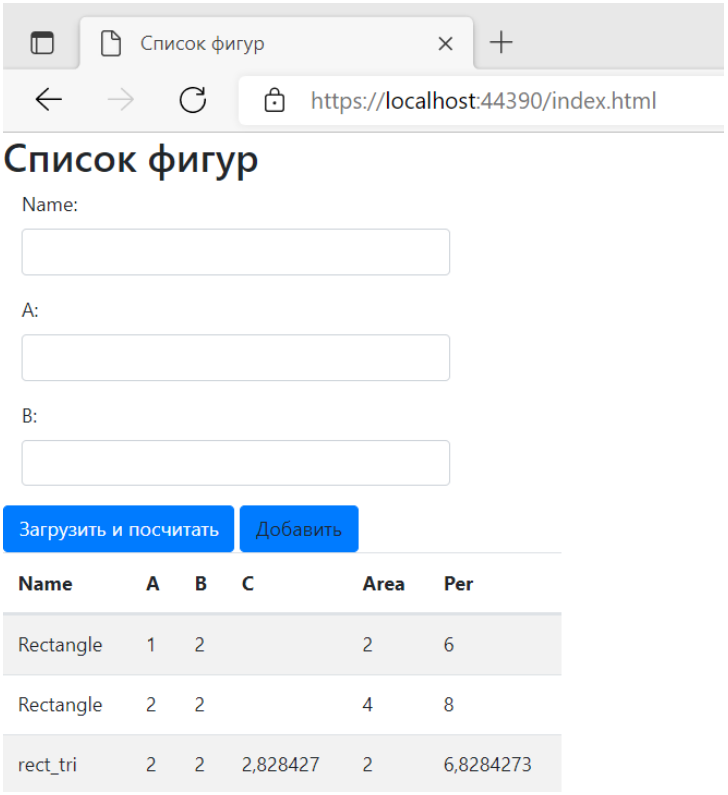


Рис. 40. Html-клиент WebAPI

Исходный код html формы приведен в Приложении В.

Аналогично можно построить и ASP.NET Core MVC приложение, повторно использовав ранее реализованный код для MVC.

Исходный код обеих версий проекта размещен в репозитории [7].

3.4.2 Реализация примера на Java

Аналогичный (но существенно упрощенный) пример был реализован на Java с использованием GUI библиотеки JavaFX. Дополнительно был реализован источник данных для работы с БД Sqlite :

```
public class DbDataSource implements DataSource {
        private final Connection connection;
    public DbDataSource(Connection connection) {
        this.connection = connection;
    }
    @Override
```

```
        public ShapeList createShapeList() throws Exception {
           try (
                  Statement stmnt = connection.createStatement();
                  ResultSet rs = stmnt.executeQuery(
                    "select * from shapes");
           ){
              ShapeList shapeList = new ShapeList();
              while (rs.next()) {
                  String type = rs.getString("shapeType");
                  double width = rs.getDouble("width");
                  double height = rs.getDouble("height");
                  if (type.equals(DataSourceUtils.TRIANGLE)) {
                      shapeList.addShape(new Triangle(width, height));
                  } else if (type.equals(DataSourceUtils.RECTANGLE)) {
                      shapeList.addShape(new Rectangle(width, height));
                  }
              }
              return shapeList;
            }
        }
    }
```

В данном случае реализован более простой вариант MVC – без шаблона наблюдатель, можно сказать, что идет последовательный вызов – форма обращается к контроллеру, который обращается к простым объектам – сущностям :

```
    public class Controller {
        public final static String FROM_FILE = "Из файла";
        public final static String FROM_TEXTAREA = "Из текста";
        public final static String FROM_DB = "Из базы данных";
        public final static String FROM_CONSTANT = "Константный список";
        private static DecimalFormat df2 = new DecimalFormat("#.##");
        public Connection connection;
        public TextField filename;
        public TextArea figuresList;
```

```
  public ChoiceBox variant;
  public Button compute;
  public Label area;
  public Label perimeter;

  public DataSource getDataSource() {
        switch ((String) variant.getValue()) {
          case FROM_FILE:
            return new
             TextDataSource(Paths.get(filename.getText()));
          case FROM_TEXTAREA:
            return new StringDataSource(figuresList.getText());
          case FROM_DB:
            return new DbDataSource(connection);
          case FROM_CONSTANT:
            return new ConstantDataSource(
                  new Triangle(10, 15),
                  new Rectangle(20, 30)
             );
        }
        throw new IllegalArgumentException(
            "Wrong value " + variant.getValue());
    }

    public void compute(MouseEvent mouseEvent) {
       try {
          ShapeList list = getDataSource().createShapeList();
          area.setText("Площадь: " +
             df2.format(list.getTotalArea()));
          perimeter.setText("Периметр: "
                + df2.format(list.getTotalPerimeter()));
       } catch (Exception e) {
           e.printStackTrace();
        }
     }
}
```

В данном случае контроллер непосредственно знает о существовании формы и виджетов, поскольку объект класса-контроллера формируется с помощью метода getController() библиотечного объекта FXMLoader, который обеспчивает и привязку данных от формы к полям контроллера. Это обусловлено особенностью библиотеки и для реализации контроллера, независимого от формы, потребуется, вероятно, вводить еще один промежуточный класс и связывать его с данным контроллером.

Внешний вид программы представлен на рисунке 41.

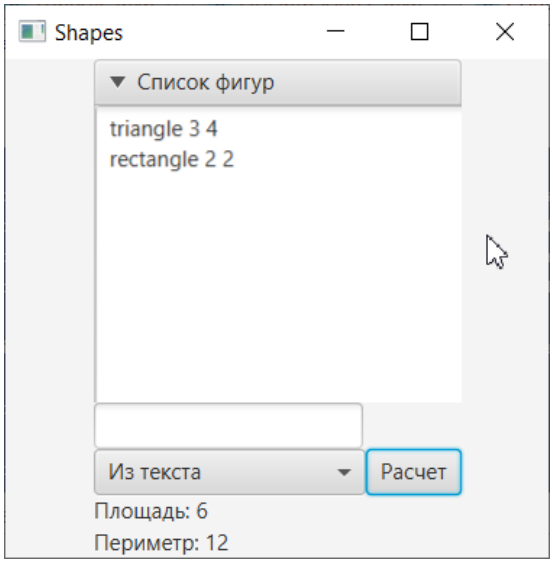


Рис. 41. Внешний вид программы на Java

3.4.3 Реализация примера на Python

В качестве примера приведем аналогичный (но тоже сильно упрощенный) проект на Python 3. В нем использованы только 2 источника данных : из файла и из сгенерированного списка, что используется для расчета произвольной заданной фигуры (из числа поддерживаемых, конечно). Реализован MVC с поддержкой Наблюдателя и обратного вызова. Для GUI использована стандартная библиотека tkinter. Следует отметить, что, поскольку Python поддерживает динамическую «утиную» типизацию, в нем нет необходимости явно прописывать интерфейсы и явно обозначать по-

лиморфизм, что с одной стороны, проще, но с другой стороны, это влияет на наглядность программы.

Например, полимофный список фигур может выглядеть так :

```
class ShapeList:
    """
    Класс списка фигур
    """
    def __init__(self):
        self.shapes = []

    def add_shape(self, shape):
        self.shapes.append(shape)

    def clear_shapes(self):
        self.shapes.clear()

    def get_total_area(self):
        total_area = 0
        for shape in self.shapes:
            total_area += shape.get_area()
        return total_area

    def get_total_perimeter(self):
        total_perimeter = 0
        for shape in self.shapes:
            total_perimeter += shape.get_perimeter()
        return total_perimeter
```

Для любителей классического подхода к ООП можно рекомендовать использование zope.interfaces или пакета abc для поддержки традиционных «абстрактных» классов.

Класс-контроллер, как и в примере на C#, использует наблюдаемый список :

```
class ObservableList:
    def __init__(self, initialValue=None):
        self.olist = initialValue
        self.callbacks = {}

    def addCallback(self, func):
        self.callbacks[func] = 1

    def delCallback(self, func):
        del self.callback[func]

    def _do_callbacks(self):
        for func in self.callbacks:
             func(self.olist.get_total_area(),
                     self.olist.get_total_perimeter())

    def get_totals(self):
        self._do_callbacks()

    def set(self, sh_list):
        self.olist=sh_list
```

Сам контроллер может выглядеть так :

```
class Controller:
    def __init__(self):
        self.sl = []
        self.model = ObservableList()

    def get_total_results_by_id(self, id, data):
        if id == 0:
           ds = TextDataSource(open(data))
        elif id == 1:
           ds = ConstDataSource(data)
```

```
        else:
            ds = StringDataSource(data)

        self.get_total_results(ds)

    def get_total_results(self, ds):
        self.sl = ds.create_shape_list()
        self.model.set(self.sl)
        self.model.get_totals()
```

Привязка формы к обратным вызовам наблюдаемого списка выполняется в самой форме :

```
def show_totals(area, perimeter):
     lb.configure(text=f' {area}\n {perimeter}')
```

и далее :

```
ctrl = Controller()
ctrl.model.addCallback(show_totals)
```

Форма GUI приложения представлена на рисунке 42.

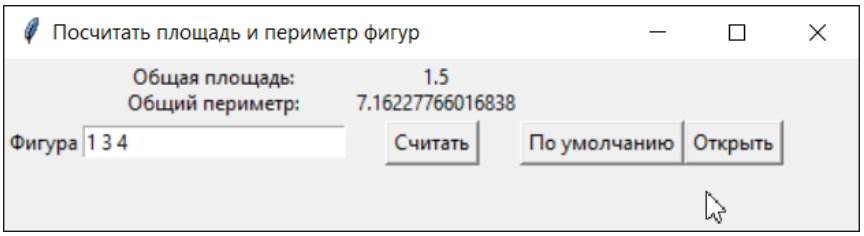


Рис. 42. Внешний вид приложения на Python

Код для создания и запуска формы достаточно простой :

```
window = Tk()
window.title('Посчитать площадь и периметр фигур')
window.geometry('500x100')
lb = Label(window, text='0')
```

```
lb.grid(column=2, row=0)
lb1 = Label(window, text = 'Общая площадь:\nОбщий периметр:')
lb1.grid(column=1, row=0)
lb2 = Label(window, text='Фигура')
lb2.grid(column=0, row=2)
en = Entry(window, width=25)
en.grid(column=1, row=2)
b1 = Button(text='Считать', command=from_str)
b1.grid(column=2, row=2)
b2 = Button(text='По умолчанию', command=from_const)
b2.grid(column=3, row=2)
b3 = Button(text='Открыть', command=from_file)
b3.grid(column=4, row=2)
ctrl = Controller()
ctrl.model.addCallback(show_totals)
window.mainloop()
```

ЗАКЛЮЧЕНИЕ

В пособии кратко рассмотрено содержание лабораторного практикума и курсовой работы по дисциплинам «Технологии программирования» и «Технологии программирования и инструментальные средства разработки систем искусственного интеллекта». При рассмотрении подходов и процессов акцент сделан на адаптивном итеративном подходе к разработке и характерных для него процессах, а при рассмотрении технологий и инструментальных средств – на средства, находящие применение при разработке систем искусственного интеллекта.

СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ

1. Андреев А. Е. Адаптивные технологии разработки программного обеспечения [Электронный ресурс]: учеб. пособие / А. Е. Андреев, С. И. Кирносенко, ВолгГТУ. - Волгоград : ВолгГТУ, 2015. - 96с.

2. Мартин Р., Мартин М. Принципы, паттерны и методики гибкой разработки на языке C#. – СПб: Символ-Плюс, 2011. – 768 с.:ил.

3. GitHub - citrux-at-vstu [Электронный ресурс]. Режим доступа : https://github.com/citrux-at-vstu

4. Colaboratory [Электронный ресурс]. Режим доступа: https://colab.research.google.com/

5. Anaconda [Электронный ресурс]. Режим доступа: https://www.anaconda.com

6. Miniconda – Conda documentation [Электронный ресурс]. Режим доступа: https://docs.conda.io/en/latest/miniconda.html

7. GitHub - aae-at-vstu [Электронный ресурс]. Режим доступа : https://github.com/aae-at-vstu/tp_shapes_sample

8. Згуральская, Е. Н. Технологии программирования : учебное пособие / Е. Н. Згуральская. — Ульяновск : УлГТУ, 2020. — 71 с. — ISBN 978-5-9795-1995-1. — Текст : электронный // Лань : электронно-библиотечная система. — URL: https://e.lanbook.com/book/165011

9. Иванова, С. М. Технологии программирования. Разработка приложений на языке C# : учебное пособие / С. М. Иванова, З. В. Ильиченкова. — Москва : РТУ МИРЭА, 2021. — 73 с. — Текст : электронный // Лань: электронно-библиотечная система. — URL: https://e.lanbook.com/book/176565

10. Маккинни, У. Python и анализ данных / У. Маккинни ; перевод с английского А. А. Слинкина. — 2-ое изд., испр. и доп. — Москва : ДМК Пресс, 2020. — 540 с. — ISBN 978-5-97060-590-5. — Текст : электронный

// Лань : электронно-библиотечная система. — URL: https://e.lanbook.com/book/131721

11. Забродин, А. В. Основы проектирования информационных систем с помощью языка UML : учебное пособие / А. В. Забродин, В. П. Бубнов. — Санкт-Петербург : ПГУПС, 2018. — 46 с. — ISBN 978-5-7641-1133-9. — Текст : электронный // Лань : электронно-библиотечная система. — URL: https://e.lanbook.com/book/111721

12. Архитектурные решения информационных систем : учебник / А. И. Водяхо, Л. С. Выговский, В. А. Дубенецкий, В. В. Цехановский. — 2-е изд., перераб. — Санкт-Петербург : Лань, 2021. — 356 с. — ISBN 978-5-8114-2556-3. — Текст : электронный // Лань : электронно-библиотечная система. — URL: https://e.lanbook.com/book/167464

13. Буч, Г. Язык UML. Руководство пользователя : руководство / Г. Буч, Д. Рамбо, И. Якобсон. — Москва : ДМК Пресс, 2008. — 496 с. — ISBN 5-94074-334-X. — Текст : электронный // Лань : электронно-библиотечная система. — URL: https://e.lanbook.com/book/1246

14. Приемы объектно-ориентированного проектирования. Паттерны проектирования : справочник / Э. Гамма, Р. Хелм, Р. Джонсон, Д. Влиссидес. — Москва : ДМК Пресс, 2007. — 368 с. — ISBN 5-93700-023-4. — Текст : электронный // Лань : электронно-библиотечная система. — URL: https://e.lanbook.com/book/1220

15. Камаев В. А. Технологии программирования [Текст] : учебник / В. А. Камаев, В. В. Костерин - М. : Высш. шк., 2005. - 359 с.. - ISBN 5-06-004870-5

16. Кузнецов М. А. Технологии распределенных систем: современные подходы [Электронный ресурс] : учеб. пособие / М. А. Кузнецов, А. Е. Андреев, ВолгГТУ. - Волгоград : ВолгГТУ, 2009. - 124 с.

ПРИЛОЖЕНИЕ А СОДЕРЖАНИЕ ФАЙЛА LAB4.IPYNB ДЛЯ ЛАБОРАТОРНОЙ РАБОТЫ № 4

Анализ и визуализация данных в Python

Pandas

Для того, чтобы начать работать с pandas, как и всегда в python, его нужно импортировать. Обычно это делают так:

```
import pandas as pd
```

Теперь попробуем загрузить набор данных. Для примера возьмём titanic.csv из каталога datasets, который можно загрузить при помощи функции read_csv:

```
data = pd.read_csv('datasets/titanic.csv')
```

Взглянем на загруженные данные

```
data
```

Можно попробовать найти в этих данных что-нибудь интересное. Например, посчитаем, какое количество мужчин и женщин было на корабле. Информация об этом хранится в столбце Sex, доступ к которому можно получить следующими способами

```
data.Sex
```

или

```
data['Sex']
```

Чтобы посчитать количество мужчин и женщин воспользуемся методом value_counts:

```
data.Sex.value_counts()
```

Можем также получить процентное соотношение, указав флаг normalize:

```
data.Sex.value_counts(normalize=True)
```

Можем определить среднюю цену билета, обратившись к столбцу Fare и воспользовавшись методом mean:

```
data.Fare.mean()
```

Из нашего набора данных можно построить новый, выбрав только нужные столбцы при помощи свойства loc, например

```
new_data = data.loc[:,['Name', 'Age']]
new_data
```

Можно отфильтровать только выживших

```
survived = new_data[data.Survived==1]
survived
```

Полученный набор данных можно экспортировать обратно в csv функцией to_csv:

```
survived.to_csv('survived.csv', index=False)
```

Numpy

Numpy -- это библиотека для работы с массивами в Python. Список из стандартной библиотеки не достаточно производителен и имеет достаточно скудный интерфейс, чтобы использовать его для научных вычислений. Начнём с импорта библиотеки, который обычно записывают так:

```
import numpy as np
```

Создадим массив

```
arr = np.array([1,2,3,4])
```

Ровно как и с питоновскими списками можно получать элемент по индексу и делать слайсы:

```
arr[0]
arr[2:]
```

Но можно и доставать элементы по спи

А можно сделать из него матрицу 2x2:

```
m = arr.reshape((2,2))
m
```

Можно транспонировать эту матрицу

```
m.T
```

Вот так можно получить доступ к первой строке матрицы

```
m[0]
```

А вот так -- ко второму столбцу:

```
m[:,1]
```

Но самое интересное -- это применение операции ко всему массиву сразу. Например:

```
a = arr * 2
a
b = arr + 1
b
```

Можно даже применять арифметические операции для двух массивов поэлементно, при условии, что их размеры совпадают:

```
a + b
a - b
a * b
a / b
a ** b
```

Для нахождения скалярного произведения двух векторов есть оператор @

```
a @ b
```

Он же отвечает за умножение матрицы на вектор и перемножение матриц по правилу строка на столбец:

```
m @ a[:2]
m @ m
```

Также массив позволяет находить статистические характеристики

```
arr.sum(), arr.mean(), arr.std()
```

Можно фильтровать элементы массива по условию

```
arr[arr < 3]
```

Matplotlib

Для визуализации данных в python есть пакет matplotlib. Обычно его импортируют так:

```
from matplotlib import pyplot as plt
%matplotlib inline
```

Вторая строчка нужна для того, чтобы графики отображались прямо в блокноте, а не в отдельном окне. Давайте что-нибудь построим:

```
x = np.linspace(0, 2*np.pi, 100)
plt.plot(x, np.sin(x))
plt.show()
x = np.linspace(0, 2*np.pi, 100)
y = np.linspace(0, np.pi, 50)
xx, yy = np.meshgrid(x, y)
zz = np.sin(xx) * np.cos(yy)
plt.contourf(xx, yy, zz, levels=100)
plt.show()
x = np.linspace(0, 10, 100)
y = np.sin(x) + 0.1 * x
plt.scatter(x + np.random.rand(*x.shape), y +
np.random.rand(*x.shape))
plt.show()
```

Scikit-learn

А теперь немного машинного обучения со scikit-learn. Рассмотрим примеры задач классификации с использованием решающих деревьев, метода k ближайших соседей, а также линейную регрессию.

Решающие деревья

Рассмотрим знакомый уже нам набор данных о пассажирах титаника. Будем решать на них задачу классификации, в которой по различным характеристикам пассажиров требуется предсказать, кто из них выжил после крушения корабля и на основании полученной модели определим наиболее важные для выживания признаки.

Для начала импортируем классификатор

```
from sklearn.tree import DecisionTreeClassifier
clf = DecisionTreeClassifier()
```

Теперь подготовим данные. Для начала импортируем их и оставим только такие признаки как пол, класс, цена билета и возраст. В качестве целевой переменной у нас будет факт выживания:

```
data = pd.read_csv('datasets/titanic.csv')
interesting = data.loc[:, ['Sex', 'Pclass', 'Fare', 'Age',
'Survived']]
```

Обратим внимание, что имеются пропуски в данных. Избавимся от них, удалив соответствующие строки

```
interesting.dropna(inplace=True)
```

Теперь отделим признаки от целевой переменной

```
X = interesting.iloc[:, :-1]
y = interesting.iloc[:, -1]
```

Заметим, что столбец Sex содержит текстовые данные и преобразуем их к числовым

```
X.Sex = X.Sex.map(lambda x: 0 if x == 'female' else 1)
```

Обучим классификатор

```
clf.fit(X, y)
```

Посмотрим на его результаты на обучающей выборке

```
prediction = clf.predict(X)
```

и определим долю правильных ответов

```
from sklearn.metrics import accuracy_score
accuracy_score(y, prediction)
```

Метод k ближайших соседей

Сейчас мы будем подбирать оптимальное значение k для алгоритма kNN. Будем использовать набор данных Wine, где требуется предсказать сорт винограда, из которого изготовлено вино, используя результаты химических анализов

Начинаем с импорта классификатора

```
from sklearn.neighbors import KNeighborsClassifier
```

Загрузим данные

```
data = pd.read_csv('datasets/wine.csv', header=None)
```

Класс записан в первом столбце (три варианта), признаки — в столбцах со второго по последний.

```
X = data.iloc[:, 1:]
y = data.iloc[:, 0]
```

Для данного метода настраиваемым параметром является количество ближайших соседей k. Для определения качества классификации и оптимального количества соседей будем рассчитывать точность классификации на кросс-валидации для количества соседей в диапазоне от 1 до 50.

Создадим генератор разбиений для проведения кросс-валидации

```
from sklearn.model_selection import KFold
cv = KFold(y.size, n_splits=5, shuffle=True)
```

И с помощью GridSearchCV будем искать оптимальное значение k

```
from sklearn.model_selection import GridSearchCV

grid = {'n_neighbors': np.arange(50) + 1}
clf = KNeighborsClassifier()
gs = GridSearchCV(clf, grid, scoring='accuracy', cv=cv)
gs.fit(X, y)
```

Оптимальное значение количества соседей

```
gs.best_params_
```

А точность при этом

```
gs.best_score_
```

Логистическая регрессия

Ещё одним способом классификации, несмотря на слово "регрессия" в названии, является логистическая регрессия. Одной из ее особенностей является возможность оценивания вероятностей классов, тогда как большинство линейных классификаторов могут выдавать только номера классов.

Загрузим данные:

```
data = pd.read_csv('datasets/data-logistic.csv', header=None)
```

```
data
```

Импортируем классификатор

```
from sklearn.linear_model import LogisticRegression
clf = LogisticRegression(C=0.1)
```

Обучим классификатор

```
clf.fit(data.iloc[:,1:], data.iloc[:,0])
```

Визуализируем результат

```
xx,yy = np.meshgrid(np.linspace(data.iloc[:,1].min(),
data.iloc[:,1].max(), 100),
np.linspace(data.iloc[:,2].min(), data.iloc[:,2].max(), 100))
zz = clf.predict(np.column_stack((xx.ravel(), yy.ravel())))
zz = zz.reshape(xx.shape)
plt.contourf(xx, yy, zz, levels=100)
plt.scatter(data[data[0]==-1].iloc[:,1], data[data[0]==-
1].iloc[:,2], marker='v')
plt.scatter(data[data[0]==1].iloc[:,1],
data[data[0]==1].iloc[:,2], marker='^')
```

Определим качество классификации при помощи метрики ROC-AUC:

```
from sklearn.metrics import roc_auc_score
roc_auc_score(data[0], clf.predict(data.iloc[:, 1:]))
```

Линейная регрессия

А сейчас мы будем предсказывать зарплату по описанию вакансии! Начнём с загрузки данных

```
data = pd.read_csv('datasets/salary-train.csv')
data
```

Подготовим данные. Для начала преобразуем все слова в столбце FullDescription в нижний регистр и заменим все спецсимволы на пробелы

```
data.FullDescription = data.FullDescription.str.lower()
data.FullDescription = data.FullDescription.replace('[^a-zA-
Z0-9]', ' ', regex = True)
```

Теперь заменим все пропущенные данные в столбцах LocationNormalized и ContractTime на 'nan':

```
data['LocationNormalized'].fillna('nan', inplace=True)
data['ContractTime'].fillna('nan', inplace=True)
```

Теперь нам нужно преобразовать текст из столбца FullDescription в набор признаков. Одним из способов это сделать является TfidVectorizer.

```
from sklearn.feature_extraction.text import TfidfVectorizer
tfid = TfidfVectorizer(min_df=5)
X_train_tfid = tfid.fit_transform(data.FullDescription)
```

Чтобы представить в виде набора числовых признаков содержимое столбцов LocationNormalized и ContractTime воспользуемся DictVectorizer

```
from sklearn.feature_extraction import DictVectorizer
enc = DictVectorizer()
X_train_categ = enc.fit_transform(data[['LocationNormalized',
'ContractTime']].to_dict('records'))
```

Объединим матрицы признаков

```
from scipy.sparse import hstack
X = hstack((X_train_categ, X_train_tfid))
```

Обучим на этих признаках регрессор

```
from sklearn.linear_model import Ridge
reg = Ridge(alpha=1)
reg.fit(X, data.SalaryNormalized)
```

Загрузим и преобразуем тестовые данные тем же образом

```
test = pd.read_csv('datasets/salary-test-mini.csv')
test.FullDescription = test.FullDescription.str.lower()
test.FullDescription = test.FullDescription.replace('[^a-zA-
Z0-9]', ' ', regex = True)
test
```

Выделим признаки и получим предсказания зарплат

```
X_test_tfid = tfid.transform(test.FullDescription)
X_test_categ = enc.transform(test[['LocationNormalized',
'ContractTime']].to_dict('records'))
X_test = hstack((X_test_categ, X_test_tfid))

test.SalaryNormalized = reg.predict(X_test)
test
```

ПРИЛОЖЕНИЕ Б – ИСХОДНЫЙ КОД ФРАГМЕНТОВ ПРИЛОЖЕНИЯ ДЛЯ ОБРАБОТКИ СПИСКА ФИГУР НА C#

```
public interface Ishape  {
    String Name { get; }
    float getArea();
    float getPerimeter();
}

public abstract class AShape : Ishape  {
    protected float a;
    protected float b;

    public AShape(float _a, float _b)    {
        a = _a;
        b = _b;
    }

    public float A { get => a; }
    public float B { get => b; }

    public abstract String Name { get; }
    public abstract float getArea();
    public abstract float getPerimeter();

    public override string ToString()   {
        return "a = " + a.ToString() + " b = " + b.ToString() +
            " area = " + getArea().ToString() +
            " perimeter = " + getPerimeter().ToString();
    }
}

public class Square : AShape   {
    public Square(float _a, float _b) : base(_a, _b) { }
    public override String Name { get => "square"; }
    public override float getArea() {
        return a * b;
    }

    public override float getPerimeter()  {
        return 2 * (a + b);
    }

    public override string ToString()  {
        return Name + ": " + base.ToString();
    }
}
```

```
public class RectThreeAngle : AShape  {
    public RectThreeAngle(float _a, float _b) : base(_a, _b) { }
    public override String Name { get => "rect_tri"; }
    public override float getArea()   {
        return a * b / 2;
    }

    public override float getPerimeter()   {
        return a + b + C;
    }

    public float C {
        get {
            return (float)Math.Sqrt(a * a + b * b);
        }
    }

    public override string ToString() {
        return Name + ": " + " c = " + C.ToString() + " " +
            base.ToString();
    }
}


public class ShapesList  {
    List<IShape> shapes;
    public event PercentChangedHandler PercentChangedEvent;

    public ShapesList()  {
        shapes = new List<IShape>();
    }

    public ShapesList(List<IShape> _shapes)  {
        shapes = _shapes;
    }

    public void AddShape(IShape shape)  {
        shapes.Add(shape);
    }

    public void Clear() {
        shapes.Clear();
    }

    public float GetTotalArea() {
        float tot = 0;
        int nMax = shapes.Count;
        int cnt = 0;
```

```
            bool lEvent = PercentChangedEvent != null;
            foreach (IShape shape in shapes)  {
                tot += shape.getArea();
                if (lEvent) {
                    cnt++;
                    if (cnt >= nMax / 100) {
                        PercentChangedEvent();
                        cnt = 0;
                    }
                }
            }
            return tot;
        }

        public float GetTotalPerimeter() {
            // Аналогично предыдущему методу
        }

        public List<String> GetStrList() {
            List<String> strl = new List<string>();
            foreach (IShape shape in shapes)
                strl.Add(shape.ToString());
            return strl;
        }
    }

    public class ShapeCreator   {
        public static IShape CreateShapeFromCSString(String src) {
            IShape shape = null;
            String[] arr = src.Split(';');
            try {
                if (arr[0] == "0")
                    shape = new Square(float.Parse(arr[1]),
                      float.Parse(arr[2]));
                else
                    shape = new RectThreeAngle(float.Parse(arr[1]),
                      float.Parse(arr[2]));
            }
            catch { }
            return shape;
        }
    }

namespace Presenter {
    public interface IView {
        void ShowMessage(string msg);
        bool AskYesOrNo(string quest);
        void ShowTotResults(float area, float perimeter);
        void ShowShapeList(List<Shape4View> shapes);
```

```
    void OneMorePercent();
}

public class Presenter {
    IView view;
    IDataSource ids;

    public Presenter(IView iv) {
        view = iv;
        N_Max = 1000000;
    }

    public int N_Max { get; set; }

    // Обработчик события
    public void PercentChanged() {
        view.OneMorePercent();
    }

    // Для отчета по БД или текстовому файлу - по переданному id
    // сам источник формируется внутри по данным по умолчанию
    public void ShowReport(int ds_type_id = 0)  {
        // Инициализация источника данных [из фабрики]
        // Логика прецедента
        if (ds_type_id == 1)
            ShowReport(new CSVsource("shapes.csv"));
        else if (ds_type_id == 2)
            ShowReport(new ConstantListDS(N_Max));
        else
            ShowReport(new ConstantListDS(3));
    }

    // Для отчета по данным, переданным в виде строк извне
    public void ShowReport(List<string> lst)    {
        // Формирование источника данных из списка строк
        List<IShape> sl = new List<IShape>();
        // Логика прецедента
        foreach (string s in lst)
            sl.Add(ShapeCreator.CreateShapeFromCSString(s));
        ShowReport(sl);
    }

    public void ShowReport(IDataSource _ids) {
        ShowReport(_ids.GetShapes());
    }

    public void ShowReport(List<IShape> _sl) {
        // Логика прецедента
        if (_sl != null) {
```

```
                // Объект списка
                ShapesList sl = new ShapesList(_sl);
                int nShapes = _sl.Count; // ids.ShapesCount;
                // Если nShapes большое - выводить прогресс-бар ...
                if (nShapes >= 1000) {
                    sl.PercentChangedEvent += PercentChanged;
                }

                // Основные результаты
                float totArea = sl.GetTotalArea();
                float totPerimeter = sl.GetTotalPerimeter();
                view.ShowTotResults(totArea, totPerimeter);
                bool lShowDetails = nShapes <= 5;
                if (!lShowDetails && (nShapes <= 100)) {
                    lShowDetails = view.AskYesOrNo(
                      "Количество фигур = " + nShapes.ToString() +
                        " Выводить их ?");
                }
                if (nShapes > 100) {
                view.ShowMessage(
                     "Слишком много фигур для отображения !");
                }
                if (lShowDetails) {
                    List<Shape4View> l4view = new
                       List<Shape4View>();
                    foreach (IShape sh in _sl)  {
                      l4view.Add(
           Shape4ViewAdapter.Shape4ViewAdapter.getShape4View(sh));
                    }
                    view.ShowShapeList(l4view);
                }
             }
             else
                view.ShowMessage("Нет данных для обработки !");
        }
    }
}
```

ПРИЛОЖЕНИЕ В КОД HTML ФОРМЫ ДЛЯ ДОСТУПА К ВЕБ-СЕРВИСУ

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8" />
    <meta name="viewport" content="width=device-width" />
    <title>Список фигур</title>
<link
href="https://stackpath.bootstrapcdn.com/bootstrap/4.4.0/css/bootstr
ap.min.css" rel="stylesheet" />
</head>
<body> <h2>Список фигур</h2>
<form name="shapeForm">
    <input type="hidden" name="id" value="0" />
    <div class="form-group col-md-5">
        <label for="name">Name:</label>
        <input class="form-control" name="name" />
    </div>
    <div class="form-group col-md-5">
        <label for="">A:</label>
        <input class="form-control" name="A" type="number" />
    </div>
    <div class="form-group col-md-5">
        <label for="">B:</label>
        <input class="form-control" name="B" type="number" />
    </div>
    <div class="panel-body">
        <button  type="submit"  id="submit"  class="btn  btn-primary">
         Загрузить и посчитать</button>
        <a id="add" class="btn btn-primary">Добавить</a>
    </div>
</form>
<table class="table table-condensed table-striped  col-md-6">
<thead><tr><th>Name</th><th>A</th><th>B</th><th>C</th><th>Area</th>
<th>Per</th></tr></thead><tbody></tbody>
</table>
<script>
        // Получение списка фигур (в зависимости от кода)
        async function GetShapes(id) {
            const response = await fetch("/api/web/" + id, {
                method: "GET",
                headers: { "Accept": "application/json" }
            });
            const shapes = await response.json();
            let rows = document.querySelector("tbody");
```

```
            shapes.forEach(shape => {
                rows.append(row(shape));
            });        }
        // Добавление фигуры
        async function CreateShape(Name, A, B) {
            const response = await fetch("api/web", {
                method: "POST",
                headers:     {     "Accept":     "application/json",
                "Content-Type": "application/json"},
                body: JSON.stringify({
                    name: Name,
                    a: parseFloat(A),
                    b: parseFloat(B)
                })
            });
            if (response.ok === true) {
                const user = await response.json();
                document.querySelector("tbody").append(row(user));
            }
        }
        // создание строки для таблицы
        function row(shape) {
            const tr = document.createElement("tr");
            const nameTd = document.createElement("td");
            nameTd.append(shape.name);
            tr.append(nameTd);

            // И аналогично – другие столбцы...
            const PerTd = document.createElement("td");
            PerTd.append(shape.per);
            tr.append(PerTd);
            return tr;
        }

        // отправка формы
        document.forms["shapeForm"].addEventListener("submit", e =>
        {
            e.preventDefault();
            const form = document.forms["shapeForm"];
            const name = form.elements["name"].value;
            const A = form.elements["A"].value;
            const B = form.elements["B"].value;
            CreateShape(name, A, B);
        });
        // загрузка фигур
        GetShapes(0);
</script>
</body>
</html>
```

Учебное издание

Андрей Евгеньевич Андреев
Михаил Андреевич Кузнецов
Владимир Левatович Абдрахманов

Технологии программирования и инструментальные средства разработки систем искусственного интеллекта

Учебно-методическое пособие

Волгоградский государственный технический университет.
400005, г. Волгоград, просп. В. И. Ленина, 28, корп. 1.