Министерство науки и высшего образования Российской Федерации
Севастопольский государственный университет
Кафедра ИС

Отчет
по лабораторной работе №2
«Исследование типов данных, определяемые пользователем. Наследование. Обработка исключений в C#»
по дисциплине
«ПЛАТФОРМА .NET»

Выполнил студент группы ИС/б-17-2-о
Горбенко К. Н.
Проверил
Забаштанский А.К.

Севастополь
2019

# 1 ЦЕЛЬ РАБОТЫ

1. Познакомиться с пользовательскими типами данных в языке C#: структура и перечисление.

2. Ознакомиться со структурой класса, его созданием и использованием, описанием членов класса: полей, свойств, инициализации объектов класса с помощью конструкторов.

3. Изучить механизм создания иерархий классов в C# и применение интерфейсов при наследовании.

4. Изучить механизм генерации и обработки исключений.

# 2 ЗАДАНИЕ НА ЛАБОРАТОРНУЮ РАБОТУ

1. Проработать примеры программ 1-6, данные в теоретических сведениях. Создать на их основе программы. Получить результаты работы программ и уметь их объяснить. Внести в отчет с комментариями.

2. Для заданной структуры данных разработать абстрактный класс и класс-наследник. В классе реализовать несколько конструкторов. Создать методы, работающие с полями класса. Часть из них должны быть виртуальными. Добавить методы-свойства и индексаторы.

3. Разработать интерфейсные классы, добавляющие некоторые методы в класс-потомок. Изучить причины возникновения коллизий имен при наследовании и способы их устранения.

4. Разработать классы исключительных ситуаций и применить их для обработки возникающих исключений.

5. Написать демонстрационную программу.

Описание данных пользовательских типов:

**ПЕЧАТНОЕ ИЗДАНИЕ**: название, ФИО автора, стоимость, оглавление.

# 3 ХОД РАБОТЫ

Рассмотрим примеры программ 1-5.

### 3.1 Программы

1. Программа демонстрирует использование перечислений со значениями по умолчанию и установленными значениями. Кроме того, программа демонстрирует использование структур. При этом, не показано свойство структур как значимых классов, что, по моему мнению, является более важным свойством, чем наличие конструкторов, о чем говорят методические указания.

2. Программа показывает возможности использования свойств как способа доступа к полям класса. При этом упрощается регулирование доступа и уменьшается количество необходимого для этого программного кода.

3. Программа демонстрирует использование индексаторов для созданных классов. При этом возможно добавление дополнительной логики к уже существующим классам с индексаторами.

4. Программа демонстрирует возможности объектного программирования в языке C#. Метод базового класса переопределяется методом класса-наследника. Кроме того, класс-наследник добавляет свои собственные методы к уже унаследованным.

5. Программа показывает способы устранения коллизий имен при наследовании и реализации интерфейсов. Для устранения коллизий необходимо явно указывать интерфейс, для которого определяется метод.

### 3.2 Реализация программ в соответствии с вариантом

Напишем класс, соответствующий печатному изданию:

```
1 public abstract class PrintedEdition
2 {
3     public string Name   { get; set; }
4     public string Author { get; set; }
5     public double Price  { get; set; }
6     public IDictionary<string, string> Contents { get; set; }
7     public PrintedEdition() { }
8     public PrintedEdition(string name)
9     {
10        Name = name ?? throw new ArgumentNullException("name was null.");
11    }
12    public PrintedEdition(string name, string author, double price, IDictionary
          <string, string> contents)
13    {
14        Name = name ?? throw new ArgumentNullException(nameof(name));
15        Author = author ?? throw new ArgumentNullException(nameof(author));
```

```
16          Contents = contents ?? throw new ArgumentNullException(nameof(contents)
                );
17          Price = price;
18      }
19      public string this[string chapter]
20      {
21          get
22          {
23              if (chapter == null)
24              {
25                  throw new ArgumentNullException(nameof(chapter));
26              }
27              return Contents.ContainsKey(chapter) ? Contents[chapter] : throw
                    new ChapterNotFoundExeption($"The key \"{chapter}\" was not
                    found");
28          }
29          set
30          {
31              if (value != null)
32              {
33                  Contents[chapter] = value;
34              }
35          }
36      }
37
38      public abstract string Print();
39  }
```

Абстрактный класс содержит свойства, конструкторы, индексатор и абстрактный метод. Кроме того, его индексатор выбрасывает объявленное исключение, представленное ниже:

```
1 public class ChapterNotFoundExeption : KeyNotFoundException
2 {
3     public ChapterNotFoundExeption() { }
4     public ChapterNotFoundExeption(string name) : base(name) { }
5     public ChapterNotFoundExeption(string name, Exception inner) { }
6 }
```

Напишем теперь его наследника:

```
1 public class Book : PrintedEdition, IPublishable, INewPubishable
2 {
3     public string TitleImage { get; set; }
4     public Book() { }
5     public Book(string name) : base(name) { }
6
7     public Book(string name, string author, double price, IDictionary<string,
```

```
        string> contents, string titleImage) : base(name, author, price,
        contents)
8       {
9           TitleImage = titleImage ?? throw new ArgumentNullException(nameof(
                titleImage));
10      }
11      public override string Print()
12      {
13          var builder = new StringBuilder();
14          foreach (var chapter in Contents)
15              builder.Append($"Chapter {chapter.Key}\n\n{chapter.Value}\n\n");
16          return builder.ToString();
17      }
18
19      public Book RePublish()
20      {
21          return new Book()
22          {
23              Name = this.Name + "new edition",
24              Author = this.Author,
25              Price = this.Price,
26              Contents = this.Contents,
27              TitleImage = this.TitleImage
28          };
29      }
30
31      Book IPublishable.RePublish()
32      {
33          return new Book() {
34              Name = "IPublishable"
35          };
36      }
37
38      Book INewPubishable.RePublish()
39      {
40          return new Book() {
41              Name = "INewPublishable"
42          };
43      }
44  }
```

Данный класс является наследником класса PrintedEdition и реализует интерфейсы IPublishable и INewPubishable, приведенные далее. Его конструкторы вызывают конструкторы базового класса, а метод Print определяет данный метод базового класса.

Кроме того, метод RePublish реализуется для разных интерфейсов по-разному.

Интерфейсы IPublishable и INewPubishable:

```
public interface IPublishable
{
    Book RePublish();
}

public interface INewPubishable
{
    Book RePublish();
}
```

Демонстрационная программа:

```
public class Program
{
    public static void Main(string[] args)
    {
        var contents = new Dictionary<string, string>()
        {
            { "First", "Some content here" }
        };
        var book = new Book("name", "me", 500, contents, "image.jpg");

        try
        {
            Console.WriteLine(book.RePublish().Name);
            Console.WriteLine((book as IPublishable).RePublish().Name);
            Console.WriteLine((book as INewPubishable).RePublish().Name);
            var chapter = book["not existing chapter"];
        }
        catch (ChapterNotFoundExeption e)
        {
            Console.WriteLine(e.Message);
        }
    }
}
```

# 4 ВЫВОДЫ

В ходе лабораторной работы были рассмотрены механизмы ООП в языке C#. Поблемы, возникающие наследовании и реализации интерфейсов - коллизии имен, решаются явным указанием интерфейса. Собственные исключения описываются как классы, унаследованные от любого другого класса исключения.

Классы от структур отличаются тем, что классы передаются по значению, а

структуры по ссылке.