Лабораторная работа №6:
Базовые принципы наследования и полиморфизма

Цель:

Целью данной работы является изучение базовых приёмов наследования и полиморфизма на примере их реализации в языке C#

Задание:

В рамках выполнения лабораторной работы требуется:

1. Разработать структуру товаров и/или услуг абстрактного интернет магазина. Товары/услуги должны иметь как общие свойства (цена есть у всех), так и уникальные (срок годности, объём, дата оказания и т.д.).
2. Разработать и реализовать программное средство для создания, редактирования, сохранения и загрузки списка доступных товаров/услуг.
3. Разработать и реализовать программное средство для отображения списка товаров и/или услуг. В процессе отображения, товары и/или услуги должны быть сгруппированы согласно их иерархии категорий и подкатегорий.

Пример структуры товаров:

Книги и диски. Книги бывают по программированию, кулинарии и эзотерике. У всех товаров есть название (строка), цена (число) и штрих-код (строка). У всех книг – количество страниц (число). У книг по программированию – язык программирования (строка). У книг по кулинарии – основной ингредиент (строка). У книг по эзотерике – минимальный возраст читателя (число). Диски бывают CD и DVD. Независимо от типа диска, его содержимым может быть: музыка, видео, ПО.

Справочная информация:

Механизм наследования выполняет три основные функции:

- обеспечивает иерархичность данных
- позволяет повторно использовать уже реализованные классы
- позволяет реализовать полиморфизм подтипов

Примером простого наследования может служить следующий исходный код:

```
//Базовый класс, содержит общие данные
class CStudent
{
    public string fio;
    public int subj_1;

    public virtual void set(string st)
    {
        //метод заполняющий поля класса
    }
}
```

```
//дочерний класс
class CMRMStudent : CStudent
{
    public int subj_2;

    public override void set(string st)
    {
        base.set(st);   //вызов метода базового класса
        //метод дочернего класса заполняющий новые поля
    }
}
```

Базовым (или родительским), называется класс, от которого происходит наследование. Дочерним, называется класс, наследуемый от базового.

Дочерний класс содержит все поля и методы базового класса, при этом обращение к ним в явной форме, осуществляется через ссылку **base**.

Модификатор **virtual**, означает что, в дочернем классе может быть объявлен метод с таким же именем и набором параметров, однако выполняющий иные действия.

Модификатор **override**, позволяет переопределить в дочернем классе метод, объявленный в базовом классе с модификатором **virtual**.

Подобные конструкции, в сочетании с ниже описанным свойством ссылок на базовые типы данных, позволяет реализовать полиморфизм включения.

Полиморфизма включения:

```
//обе ссылки на объекты имеют одинаковый тип
CStudent student_1 = new CStudent();
CStudent student_2 = new CMRMStudent();
//...
string data;
//..
student_1.set(data);    //будет вызван метод базового класса
student_2.set(data);    //будет вызван метод дочернего класса
```

Из примера видно, что ссылка на базовый класс, может указывать на объект дочернего типа, при этом при обращении к методам, будет автоматически вызываться нужная реализация.

Для дочерних классов, в полной мере действуют правила на модификаторы доступа **private** и **public**, однако, добавляется ещё один **protected**.

Пример использования модификаторов доступа при наследовании:

```
//Базовый класс, содержит общие данные
class CStudent
{
    private string fio;
    protected int subj_1;

    public virtual void set(string st)
    {
        //метод заполняющий поля класса
    }
}
```

```
//дочерний класс
class CMRMStudent : CStudent
{
    public int subj_2;

    public override void set(string st)
    {
        //ошибка, поля private доступны только в том классе, где они объявлены
        base.fio = "Иванов";
        base.subj_1 = 5;    //protected поле доступно в дочерних классах
        base.set(st);       //public поля и методы доступны везде
    }
}

//обе ссылки на объекты имеют одинаковый тип
CStudent student_1 = new CStudent();
CStudent student_2 = new CMRMStudent();
//...
//ошибка, protected поле не доступно за пределами родительского и дочерних классов
student_1.subj_1 = 5;
student_2.subj_1 = 5;
```

Как видно из примера, модификатор доступа **protected**, может обеспечить свободный доступ к данным внутри иерархии классов, но при этом защитить данные от доступа извне.

Интерфейс класса:

Для того, что бы гарантировать, что при добавлении нового класса в уже существующую иерархию, он будет поддерживать минимально необходимый для работы программного средства функционал, используются интерфейсы классов.

Пример интерфейса:

```
interface IStudent
{
   //реализация этого интерфейса, обязана содержать метод set
   //принимающий в качестве параметра строку
   void set(string st);
}

//Базовый класс, содержит общие данные
class CStudent : IStudent
{
   private string fio;
   protected int subj_1;

   public virtual void set(string st)
   {
        //метод заполняющий поля класса
   }
}
IStudent student = new CStudent();
student.set(data);
```

В C#, интерфейс не может описывать модификаторы доступа и поля, однако может задавать обязательные методы, индексаторы и свойства.

Представленный в примере интерфейс, гарантирует, что любой унаследованный от него класс, будет содержать реализацию метода **set** описанного в интерфейсе.

Существуют ситуации, когда на уровень интерфейса необходимо вынести не только прототипы методов и свойств, но так же поля и реализацию некоторых методов. В таком случае, применяется механизм абстрактных классов.

Пример абстрактного класса:

```
//Базовый класс, содержит общие данные
abstract class CStudent
{
    private string fio;

    public abstract void set(string st);    //метод не имеющий реализации
}

//дочерний класс
class CMRMStudent : CStudent
{
    public int subj_2;

    public override void set(string st)
    {
           //реализация метода
    }
}
CStudent student_1 = new CStudent();    //ошибка, абстрактный класс не может быть создан
CStudent student_2 = new CMRMStudent(); //корректно
```

Абстрактный класс, это класс, содержащий абстрактные (не имеющие реализации) методы. Поскольку один или более методов подобного класса не имеет реализации, невозможно создать объект этого класса. Однако, ссылка на абстрактный класс, по-прежнему может указывать на любой объект дочернего типа.

Для определения типа объекта, на который ссылается та или иная переменная родительского типа, существуют следующие стандартные методы:

- Проверка типа при помощи оператора **is**
- Преобразование типов с помощью оператора **as**

Пример использования оператора **is**:

```
CStudent student_1 = new CStudent();
CStudent student_2 = new CMRMStudent();
//...
string data = "";

if (student_1 is CStudent)  //условие истино
    Console.WriteLine("student_1 имеет тип CStudent");
if (student_2 is CStudent)  //условие истино
    Console.WriteLine("student_1 совместим CStudent");
if (student_1 is CMRMStudent)   //условие ложно
    Console.WriteLine("не отобразится на экране");
if (student_2 is CMRMStudent)  //условие истино
    Console.WriteLine("student_2 имеет тип CMRMStudent");
```

Пример использования оператора **as**:

```
CStudent student_1 = new CStudent();
CStudent student_2 = new CMRMStudent();
//...
CStudent stud = student_2 as CStudent;
if (stud == null)
    Console.WriteLine("Переменная student_2 не может быть преведена к типу CStudent");
else
    Console.WriteLine("Преведение прошло успешно");
```

Доступ к базе данных SQLite:

Для хранения списка товаров и/или услуг можно использовать либо файл формата XML, либо базу данных формата SQLite.

Примечание:

Создание и просмотр базы данных лучше всего осуществлять при помощи стороннего ПО. Например, DB Browser for SQLite: http://sqlitebrowser.org/

Добавить поддержку баз данных SQLite в Microsoft Visual Studio можно установив соответственный nu-get package: https://www.nuget.org/packages/System.Data.SQLite

Ниже приведены некоторые функции для работы с базой данных SQLite.

Создание базы данных:

```
using System.Data.SQLite;

…

SQLiteConnection.CreateFile("D:\\MyDocs\\method\\intsys\\MyDatabase.sqlite");
```

Подключение к уже существующей базе данных:

```
SQLiteConnection m_dbConnection;
m_dbConnection = new SQLiteConnection("Data Source=D:\\
                                      MyDatabase.sqlite;Version=3;");
m_dbConnection.Open();
…
m_dbConnection.Close();
```

Добавление таблицы:

```
string sql = "CREATE TABLE test (name VARCHAR(20), score INT)";

SQLiteCommand command = new SQLiteCommand(sql, m_dbConnection);

command.ExecuteNonQuery();
```

Запись данных:

```
string sql = "INSERT INTO highscores (name, score) VALUES ('Me', 3)";
SQLiteCommand command = new SQLiteCommand(sql, m_dbConnection);
command.ExecuteNonQuery();
```

Чтение данных:

```
string sql = "SELECT * FROM highscores ORDER BY score desc";
SQLiteCommand command = new SQLiteCommand(sql, m_dbConnection);
SQLiteDataReader reader = command.ExecuteReader();
while (reader.Read())
    textBox1.Text += reader["name"] + " : " + reader["score"];
```

Список литературы:

1. Шилдт Г. - C# 4.0: полное руководство. Издательство: Вильямс, 2011 г. (страницы с 329 по 391 и с 537 по 540)
2. Основы программирования на C#: http://www.intuit.ru/studies/courses/2247/18/info (лекции 18, 19 и 22)
3. Справочник по языку SQL: http://sql.itsoft.ru/