Министерство образования и науки Российской Федерации

Федеральное государственное автономное образовательное учреждение высшего профессионального образования

«Уральский федеральный университет имени первого Президента России Б.Н. Ельцина»

Институт фундаментального образования

Кафедра «Информационные системы и технологии»

**Инкапсуляция и Наследование**

ОТЧЕТ

по лабораторной работе №2

Руководитель : Александров О.Е

Студент: Антропов А.Ю.

Группа: ФО-250003

Екатеринбург

2016

**Инкапсуляция и Наследование**

Инкапсуляция – это скрытие реализации объекта от конечного пользователя, которое в Си-шарп осуществляется при помощи модификаторов доступа (private, public…). Конечным пользователем объекта здесь выступает либо объект наследник, либо программист.

- *public* – доступ к члену возможен из любого места одной сборки, либо из другой сборки, на которую есть ссылка;  
- *protected* – доступ к члену возможен только внутри класса, либо в классе-наследнике (при наследовании);  
- *internal* – доступ к члену возможен только из сборки, в которой он объявлен;  
- *private* – доступ к члену возможен только внутри класса;  
- *protected internal* - доступ к члену возможен из одной сборки, либо из класса-наследника другой сборки.

Наследование, вместе с инкапсуляцией и полиморфизмом, является одной из трех основных характеристик (или *базовых понятий*) объектно-ориентированного программирования. Наследование позволяет создавать новые классы, которые повторно используют, расширяют и изменяют поведение, определенное в других классах. Класс, члены которого наследуются, называется *базовым классом*, а класс, который наследует эти члены, называется *производным классом*. Производный класс может иметь только один непосредственный базовый класс. Однако наследование является транзитивным. Если ClassC является производным от ClassB, и ClassB является производным от ClassA, ClassC наследует члены, объявленные в ClassB и ClassA.

**Изменения.**

**Изменения в Class Person:**

Добавлено свойство класса Person GetLines, осуществляющее запись дерева в файл посредством WriteToFile. GetLines имеет тип string[] для обработки родителей данного человека по отдельности:

public string[] GetLines()

{

string[] s = new string[2];

if (this.Mothers == null) { s[0] = ""; }

else { s[0] = this.Mothers.ToString() + " -> " + this.ToString(); }

if (this.Fathers == null) { s[1] = ""; }

else { s[1] = this.Fathers.ToString() + " -> " + this.ToString(); };

return s;

}

Добавлено переопределение класса метода-родителя ToString(). Теперь он возвращает пол и имя необходимого человека в стандартном для этой программы виде:

//Перегруженный метод

public override string ToString()

{

return "(" + this.Sex + ")" + this.Name;

}

**Изменения в Class FamilyTree:**

Добавлен вывод всех Person’ов дерева MyTree посредством метода ListAll(без сортировки):

//Печать дерева

public void ListAll()

{

foreach (Person p in Person.AllPersons)

{

System.Console.WriteLine(p.ToString());

}

}

Добавлен метод записи в файл WriteToFile с аргументом string FileName(указание пути к файлу).

//Запись в файл

public void WriteToFile(string FileName)

{

string[] s;

StreamWriter outputFile = new StreamWriter(FileName);

foreach (Person p in Person.AllPersons)

{

s = p.GetLines();

if (s[0] != "") { outputFile.WriteLine(s[0]); }

else if (s[1] != "") { outputFile.WriteLine(s[1]); }

}

outputFile.Close();

}

Добавлен метод чтения из файла WriteToFile с аргументами string FileName(указание пути к файлу) и Boolean ConsoleOut.

//Чтение из файла

public void ReadFromFile(string FileName, Boolean ConsoleOut = true)

{

string s = "";

StreamReader inputFile = new StreamReader(FileName);

while (!inputFile.EndOfStream)

{

if (ConsoleOut)

{

s = inputFile.ReadLine();

}

ParseLine(s);

}

inputFile.Close();

}

**Изменения в Class Program:**

Добавлена константа string cFileName, в которой указан путь к файлу:

const string cFileName = @"C:\tmp\FamilyTree.txt";

В методе Main создан объект экземпляра класса FamilyTree ft. Затем происходит чтение из файла, в случае отсутствия папки – выводит ошибку, в случае отсутствия файла – создает его.

static void Main(string[] args)

{

FamilyTree ft = new FamilyTree();

//Чтение из файла

if (File.Exists(cFileName))

{

try

{

ft.ReadFromFile(cFileName);

}

catch (Exception e)

{

System.Console.WriteLine(e.Message);

};

}

В строку Cmd записывается команда с консоли а bool DoCmd отвечает за цикл while.

string Cmd;

bool DoCmd = true;

В цикле while в консоли выводит запрос на введение какой-либо команды. Их три варианта (пока что): “(“, “q” и “l”. Если вы не ввели команду в консоль выводится ошибка о том, что вы не ввели команду, если такой команды нет, тоже выведется ошибка. С команды “(“ начинается сама программа (обработка записанной в консоли строки Cmd). Команда “q” прерывает цикл while и завершает программу. Команда “l” выводит в консоль всех Person’ов, которые вы ввели(или которые остались в файле).

while (DoCmd)

{

System.Console.WriteLine("Введите команду: ");

Cmd = System.Console.ReadLine();

if (Cmd.Length < 1) { System.Console.WriteLine("Вы не ввели команду"); }

switch (Cmd.Substring(0, 1))

{

case "(":

try

{

ft.ParseLine(Cmd);

}

catch (Exception e) { System.Console.WriteLine(e.Message); }

break;

case "q": DoCmd = false; System.Console.WriteLine("Программа завершена"); break;

//Вывод всего дерева

case "l":

try { ft.ListAll(); }

catch (Exception e) { System.Console.WriteLine(e.Message); }

break;

default: Console.WriteLine("Unknown Command: " + Cmd + ". Попробуйте еще раз..."); break;

}

}

//Запись в файл

ft.WriteToFile(cFileName);

System.Console.WriteLine(ft);

System.Console.WriteLine("Программа завершена");

System.Console.ReadKey();

}

}