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»

**Полиморфизм**

ОТЧЕТ

по лабораторной работе №3

Руководитель : Александров О.Е

Студент: Антропов А.Ю.

Группа: ФО-250003

Екатеринбург

2016

**Полиморфизм**

 Полиморфизм часто называется третьим столпом объектно-ориентированного программирования после инкапсуляции и наследования. Полиморфизм — слово греческого происхождения, означающее "многообразие форм" и имеющее несколько аспектов.

1. Во время выполнения объекты производного класса могут обрабатываться как объекты базового класса в таких местах, как параметры метода и коллекции или массивы. Когда это происходит, объявленный тип объекта перестает соответствовать своему типу во время выполнения.
2. Базовые классы могут определять и реализовывать [виртуальные](https://msdn.microsoft.com/ru-ru/library/9fkccyh4.aspx) методы, а производные классы — [переопределять](https://msdn.microsoft.com/ru-ru/library/ebca9ah3.aspx) их, т. е. предоставлять свое собственное определение и реализацию. Во время выполнения, когда клиент вызывает метод, CLR выполняет поиск типа объекта во время выполнения и вызывает перезапись виртуального метода. Таким образом, в исходном коде можно вызвать метод на базовом классе и привести версию производного класса метода, который необходимо выполнить.

**Изменения.**

**Изменения в Class Person:**

 Добавлен метод для поиска потомков во всех поколениях Person’a с аргументом int generation (поколение). В строку s записывается поколение (например, -1), затем Имя, введенное с консоли и “:”. После этого поколение уменьшается на 1, и с помощью оператора foreach производится поиск потомков в листе Childes. Результат поиска записывается (добавляется) в строку s, после чего производится поиск потомков у детей и т.д. К данному методу можно обратиться в консоли, с помощью команды p.

//Потомки

 public void ListDescendants(int generation)

 {

 string s = generation.ToString() + this.ToString() + ": ";

 generation--;

 foreach (Person c in Childes)

 {

 s = s + c.ToString() + ";";

 c.ListDescendants(generation);

 }

 System.Console.WriteLine(s);

 }

**Изменения в Class FamilyTree:**

 В метод печати дерева ListAll добавлена сортировка по имени в листе AllPersons.

//Печать дерева

 public void ListAll()

 {

 //Сортировка

 Person.AllPersons.Sort(delegate(Person p1, Person p2) { return p1.Name.CompareTo(p2.Name); });

 foreach (Person p in Person.AllPersons)

 {

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

 }

 }

 Добавлен метод для поиска предков во всех поколениях Person’a с аргументами Person p и int gen (поколение). В строку s записывается имя введенного Person’a. Затем определяются родители у Person’a p(Person f(Отец) и Person m(Мать)). После этого с помощью условия if проверяются, есть ли у Person’a родители. Результат поиска записывается в строку s, после чего ее выводят в консоль. И если у Person’a есть родители, то производится поиск предков у родителей и т.д. Условие s != p.ToString() необходимо для того, чтобы в консоли не выводился сам последний проверяющийся Person. К данному методу можно обратиться в консоли, с помощью команды d.

 //Вывод предков

 public void ListAncestors(Person p, int gen)

 {

 //Проверка принадлежности к дереву

 string s = p.ToString();

 Person f = p.Fathers;

 Person m = p.Mothers;

 if (f != null) { s = f.ToString() + " "; };

 if (m != null) { s = s + m.ToString() + " "; };

 if (s != "" && s != p.ToString()) { System.Console.WriteLine(gen.ToString() + ". " + s); };

 if (f != null) { ListAncestors(f, gen - 1); };

 if (m != null) { ListAncestors(m, gen - 1); };

 }

 Метод ListAncestors с аргументом Name(имя) позволяет определить ошибку на введение неизвестного имени. Он обращается к FindPerson, чтобы найти имя, если не находит, то в консоль выводится ошибка. Если же имя найдено, то программа обращается к методу ListAncestors с аргументами Person p0 и int gen = -1.

 public void ListAncestors(string Name)

 {

 Person p0 = FindPerson(Name);

 if (p0 == null)

 { System.Console.WriteLine("Неизвестное имя " + Name); }

 else

 {

 ListAncestors(p0, -1);

 }

 }

**Изменения в Class Program:**

 В Class Program добавлены две команды: “p” и “d”.

 С помощью команды “p” в консоль выводятся все предки Person’a (все поколения). Обращение к команде выглядит следующим образом “p Имя Person’a”. Эта команда обращается к методу ListAncestors, который позволяет вывести в консоль всех предков Person’a.

// Вывод всех родителей

 case "p":

 try { ft.ListAncestors(Cmd.Substring(1, Cmd.Length - 1).Trim()); }

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

 break;

С помощью команды “d” в консоль выводятся все потомки Person’a (все поколения). Обращение к команде выглядит следующим образом “d Имя Person’a + поколение (число)”. С помощью разделителя Split строка делится на две части по пробелу: в одной строке остается имя, во второй целое число поколения. Если вы не ввели поколение, то оно автоматически будет равным -1. Эта команда обращается к методу ListAllDescendants с аргументами Имя и поколение, который позволяет вывести в консоль всех потомков Person’a.

 // Вывод всех потомков

 case "d":

 try

 {

 int generation;

 string[] s = Cmd.Split(' ');

 if (s.Length > 2) { generation = Convert.ToInt32(s[2]); } else { generation = -1; }

 ft.ListAllDescendants(s[1].Trim(), generation);

 }

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

 break;