МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

Учреждение образование «Полоцкий государственный университет»

Факультет информационных технологий

Кафедра технологий программирования

**Лабораторная работа №3**

по дисциплине: «Объектно-ориентированные технологии

программирования и стандарты проектирования»

на тему: «Наследование»

Вариант №8 «Отдел кадров»

|  |  |
| --- | --- |
| Выполнил | Студент группы 21-ИТ-1  Макеёнок Д.И. |
|  |  |
| Проверил | Ассистент  Хирьянов И.Д. |

Полоцк 2023

**Цель работы:** научиться создавать иерархию классов, создавать объекты базовых классов и классов-наследников, вызывать методы базового класса и класса-наследника, понимать влияние модификаторов доступа при наследовании, понимать порядок вызова конструкторов и деструкторов при наследовании.

**Ход работы**

**Задание:**

1. Дополнить и при необходимости модифицировать приложение, разработанное согласно варианта лабораторной работы No2.

2. Создать несколько классов-наследников для дополнительного класса (т.е. дополнительный класс станет базовым классом для новых создаваемых классов). Каждый класс-наследник должен включать в себя атрибуты, отличающие его от других классов.

3. Определить в новых классах деструктор и конструкторы (по умолчанию, с параметрами, копирования).

4. В каждом конструкторе и деструкторе выдавать сообщение, показывающее, какой именно конструктор или деструктор и какого класса был вызван.

5. Определить в новых классах методы для ввода и вывода атрибутов как самого класса-наследника, так и базового класса. Метод вывода должен иметь то же имя, что и метод вывода в базовом классе.

6. Дополнить меню в основном теле программы тем же функционал что и в лабораторной работе No1, только теперь для классов наследников.

7. Модифицировать и, при необходимости, дополнить основной класс функционалом для добавления, удаления, просмотра и редактирования классов-наследников в массиве объектов (тип массива остается базовый класс).

8. Реализовать все необходимые проверки на вводимые данные.

9. Сделать выводы.

**Вариант 8**

Тема проекта: приложение «Отдел кадров».

Спроектировать ПО, предназначенное для создания и просмотра информации

по штатным работникам предприятия. Функции, которые должны быть

реализованы в приложении: добавление, удаление, редактирование и

просмотр работников, структуризация работников по отделам, детальная

информация о работнике.

Листинг 1.1 - реализация меню программы:

namespace lab1;

class Program

{

static void Main()

{

int dep;

bool flag;

Department department = new Department();

Human eployment = new Human();

while (true)

{

Console.WriteLine("Что делаем?\n" +

"1 - Добавить сотрудника\n" +

"2 - Удалить сотрудника\n" +

"3 - Просмотреть сотрудников\n" +

"4 - Редактировать сотрудника\n" +

"5 - Добавить сотрудника через конструктор с параметрами\n" +

"6 - Копировать сотрудника\n" +

"7 - Выдать сотруднику работу\n" +

"8 - Уволить сотрудника\n");

int.TryParse(Console.ReadLine(), out int n);

switch (n)

{

case 1:

Console.WriteLine("В какой отдел добавить сотрудника?(1-3)");

while (true)

{

if (!(int.TryParse(Console.ReadLine(), out dep) && CorrectImput.InRange(1, 3, dep)))

{

Console.Write("Некоректный ввод, попробуй ещё раз: ");

continue;

}

dep--;

eployment.NewEployment();

department.AddNewEployee(dep, eployment);

Console.WriteLine("Успешно");

// Console.ReadKey();

Console.Clear();

break;

}

break;

case 2:

Console.WriteLine("Из какого отдела удалить сотрудника?(1-3)");

while (true)

{

if (!(int.TryParse(Console.ReadLine(), out dep) && CorrectImput.InRange(1, 3, dep)))

{

Console.Write("Некоректный ввод, попробуй ещё раз: ");

continue;

}

dep--;

flag = department.DeleteEployee(dep);

if (flag)

{

Console.WriteLine("Успешно");

break;

}

else

{

break;

}

}

// Console.ReadKey();

//Console.Clear();

break;

case 3:

Console.WriteLine("Какой отдел вывести?(1-3) / all - вывести всех");

while (true)

{

string tem = Console.ReadLine();

if (int.TryParse(tem, out dep))

{

if (!CorrectImput.InRange(1, 3, dep))

{

Console.WriteLine("Некоректный ввод, попробуй ещё раз: ");

continue;

}

dep--;

department.ShowEployee(dep);

break;

}

else

{

if (tem == "all")

{

department.ShowEployee();

break;

}

else

{

Console.WriteLine("Некоректный ввод, попробуй ещё раз: ");

continue;

}

}

}

//Console.ReadKey();

//Console.Clear();

break;

case 4:

Console.WriteLine("Из какого отдела редактируемый сотрудник?(1-3)");

while (true)

{

if (!(int.TryParse(Console.ReadLine(), out dep) && CorrectImput.InRange(1, 3, dep)))

{

Console.WriteLine("Некоректный ввод, попробуй ещё раз: ");

continue;

}

dep--;

flag = department.EditingEployee(dep);

if (flag)

{

Console.WriteLine("Успешно");

break;

}

else

{

break;

}

}

Console.ReadKey();

Console.Clear();

break;

case 5:

Console.WriteLine("В какой отдел добавить сотрудника?(1-3)");

while (true)

{

if (!(int.TryParse(Console.ReadLine(), out dep) && CorrectImput.InRange(1,3,dep)))

{

Console.Write("Некоректный ввод, попробуй ещё раз: ");

continue;

}

dep--;

department.AddEmployeeByConstructor(dep);

Console.WriteLine("Успешно");

// Console.ReadKey();

Console.Clear();

break;

}

break;

case 6:

Console.WriteLine("Из какого отдела копируемый сотрудник?(1-3)");

while (true)

{

if (int.TryParse(Console.ReadLine(), out dep) && CorrectImput.InRange(1, 3, dep))

{

dep--;

if (department.CheckEmployee(dep))

{

Console.WriteLine("Введите номер копируемого сотрудника ");

if (int.TryParse(Console.ReadLine(), out int number)) { }

else break;

Console.WriteLine("Введите количество копий ");

if (!int.TryParse(Console.ReadLine(), out int quantity))

break;

department.Copy(dep, number, quantity);

Console.WriteLine("Успешно");

Console.ReadKey();

Console.Clear();

break;

}

else

{

Console.WriteLine("В этом отделе никого нет.");

break;

}

}

else Console.Write("Некоректный ввод, попробуй ещё раз: ");

}

break;

case 7:

Console.WriteLine("Из какого отдела сотрудник?(1-3)");

while (true)

{

if (int.TryParse(Console.ReadLine(), out dep) && CorrectImput.InRange(1, 3, dep))

{

dep--;

Console.WriteLine("Введите номер сотрудника ");

if (int.TryParse(Console.ReadLine(), out int number)) { }

else break;

flag = department.AddNewWorker(dep, number);

if (flag)

{

Console.WriteLine("Успешно");

break;

}

else

{

break;

}

Console.Clear();

break;

}

else Console.Write("Некоректный ввод, попробуй ещё раз: ");

}

break;

case 8:

Console.WriteLine("Из какого отдела сотрудник?(1-3)");

while (true)

{

if (int.TryParse(Console.ReadLine(), out dep) && CorrectImput.InRange(1, 3, dep))

{

dep--;

Console.WriteLine("Введите номер сотрудника ");

if (int.TryParse(Console.ReadLine(), out int number)) { }

else break;

flag = department.DismissEmployee(dep, number);

if (flag)

{

Console.WriteLine("Успешно");

break;

}

else

{

break;

}

Console.Clear();

break;

}

else Console.Write("Некоректный ввод, попробуй ещё раз: ");

}

break;

}

}

}

}

Листинг 1.2 - реализация главного класса:

using System;

using System.Data;

using System.Runtime.Intrinsics.Arm;

using System.Security.Cryptography.X509Certificates;

using System.Xml.Linq;

namespace lab1

{

class Department

{

List<Human>[] depart = new List<Human>[3];

public void AddNewEployee(int num, Human hum)

{

depart[num].Add(new Human(hum));

}

public Department()

{

depart[0] = new List<Human>();

depart[1] = new List<Human>();

depart[2] = new List<Human>();

}

public bool DeleteEployee(int num)

{

int count = 0;

string name = "";

bool flag = false, MoreOne = false;

if (depart[num].Count == 0)

{

Console.WriteLine("В этом отделе никого нет.");

return false;

}

Console.WriteLine("Фамилия удаляемого сотрудника:");

name = Console.ReadLine();

for (int i = 0; i < depart[num].Count; i++)

{

if (depart[num][i].GetSurename().IndexOf(name) != -1)

{

Console.Write(i + ".");

depart[num][i].Show();

count = i;

if (flag)

{

MoreOne = true;

}

flag = true;

}

}

if (MoreOne)

{

Console.Write("Сотрудника под каким номером нужно удалить?\n");

while (true)

{

if (int.TryParse(Console.ReadLine(), out int i))

{

depart[num].RemoveAt(i);

GC.Collect();

return true;

}

else Console.WriteLine("Некоректный ввод, попробуй ещё раз: ");

}

}

else

{

if (flag)

{

depart[num].RemoveAt(count);

GC.Collect();

return true;

}

else

{

Console.WriteLine("Неверная фамилия, или такой нет");

return false;

}

}

}

public void ShowEployee()

{

int num = 0;

for (int i = 0; i < 3; i++)

{

Console.WriteLine(++num + "-------------------------------");

for (int j = 0; j < depart[i].Count; j++)

{

Console.WriteLine(j + ".");

depart[i][j].Show();

}

if (depart[i].Count == 0) Console.WriteLine("Тут пусто\n");

}

}

public void ShowEployee(int num)

{

for (int i = 0; i < depart[num].Count; i++)

{

Console.WriteLine(i + ".");

depart[num][i].Show();

}

if (depart[num].Count == 0) Console.WriteLine("Тут пусто\n");

}

public bool EditingEployee(int num)

{

int count = 0;

string name = "";

bool flag = false, MoreOne = false;

if (depart[num].Count == 0)

{

Console.WriteLine("В этом отделе никого нет.");

return false;

}

Console.Write("Фамилия редактируемого сотрудника:");

name = Console.ReadLine();

for (int i = 0; i < depart[num].Count; i++)

{

if (depart[num][i].GetSurename() == name)

{

if (depart[num][i].GetSurename().IndexOf(name) != -1)

{

Console.Write(i + ".");

depart[num][i].Show();

count = i;

if (flag)

{

MoreOne = true;

}

flag = true;

}

}

}

if (MoreOne)

{

Console.Write("Сотрудника под каким номером нужно редактировать?\n");

while (true)

{

if (int.TryParse(Console.ReadLine(), out int i))

{

count = i;

break;

}

else Console.WriteLine("Некоректный ввод, попробуй ещё раз: ");

}

}

else

{

if (!flag)

{

Console.WriteLine("Неверная фамилия, или такой нет");

return false;

}

}

Console.WriteLine("Что нужно изменить?\n" +

"1 - Имя сотрудника\n" +

"2 - Фамилию сотрудника\n" +

"3 - День рождения сотрудника\n" +

"4 - Год начала работы сотрудника\n");

int edit = Convert.ToInt32(Console.ReadLine());

switch (edit)

{

case 1:

depart[num][count].AddHumanName();

return true;

case 2:

depart[num][count].AddHumanSurename();

return true;

case 3:

depart[num][count].AddHumanBirthday();

return true;

case 4:

depart[num][count].AddHumanStartYear();

return true;

}

return false;

}

public void AddEmployeeByConstructor(int dep)

{

int startyear;

string name;

string surename;

DateTime birthday;

Console.Write("Введите имя сотрудника: ");

while (true)

{

name = Console.ReadLine();

if (CorrectImput.IsLatters(name)) break;

}

Console.Write("Введите фамилию сотрудника: ");

while (true)

{

surename = Console.ReadLine();

if (CorrectImput.IsLatters(surename)) break;

}

Console.WriteLine("Введите день рождения сотрудника: ");

while (true)

{

var chek = DateTime.TryParse(Console.ReadLine(), out birthday);

if (!chek)

{

Console.Write("Некоректный ввод, попробуй ещё раз: ");

continue;

}

break;

}

Console.Write($"Введите год начала работы(2000 - {DateTime.Now.Year}): ");

while (true)

{

if (int.TryParse(Console.ReadLine(), out startyear) && CorrectImput.InRange(2000, DateTime.Now.Year, startyear) && (startyear - 18) >= birthday.Year)

{

break;

}

else

{

Console.Write("Некоректный ввод, попробуй ещё раз: ");

}

}

Human hum = new Human(startyear, name, birthday, surename);

depart[dep].Add(hum);

}

public void Copy(int dep, int number, int quantity)

{

if (depart[dep][number] is Workers wor)

{

for (int i = 0; i < quantity; i++)

{

Workers workers = new Workers(wor);

depart[dep].Add(workers);

}

}

else if (depart[dep][number] is Dismissed dis)

{

for (int i = 0; i < quantity; i++)

{

Dismissed workers = new Dismissed(dis);

depart[dep].Add(workers);

}

}

else

{

for (int i = 0; i < quantity; i++)

{

Human human = new Human(depart[dep][number]);

depart[dep].Add(human);

}

}

}

public bool CheckEmployee(int dep)

{

if (depart[dep].Count > 0)

{

return true;

}

else return false;

}

public bool AddNewWorker(int dep,int number)

{

if (depart[dep].Count == 0)

{

Console.WriteLine("В этом отделе никого нет.");

return false;

}

Workers worker = new Workers(depart[dep][number].GetStartYear(), depart[dep][number].GetName(), depart[dep][number].Getbirthday(), depart[dep][number].GetSurename(), Workers.AddWorkersProfes());

depart[dep][number] = worker;

return true;

}

public bool DismissEmployee(int dep,int number)

{

if (depart[dep].Count == 0)

{

Console.WriteLine("В этом отделе никого нет.");

return false;

}

Dismissed dismiss = new Dismissed(depart[dep][number].GetStartYear(), depart[dep][number].GetName(), depart[dep][number].Getbirthday(), depart[dep][number].GetSurename(), Dismissed.AddDismissedProfes(depart[dep][number].GetStartYear()));

depart[dep][number] = dismiss;

return true;

}

}

}

Листинг 1.3 – реализация дополнительного класса

using System.Linq.Expressions;

using System.Security.Cryptography;

namespace lab1

{

class Human

{

protected string \_name;

protected string \_surename;

protected DateTime \_birthday;

protected int \_startYear;

public Human()

{

\_startYear = 0;

\_name = "пусто";

\_surename = "пусто";

\_birthday = DateTime.Now;

Console.WriteLine("Вызван конструкотр без параметров");

}

public Human(int startYear, string name, DateTime birthday, string surename)

{

\_startYear = startYear;

\_name = name;

\_surename = surename;

\_birthday = birthday;

Console.WriteLine("Вызван конструкотр с параметрами");

}

public Human(Human human)

{

\_startYear = human.\_startYear;

\_name = human.\_name;

\_surename = human.\_surename;

\_birthday = human.\_birthday;

Console.WriteLine("Вызван конструкотр копирования");

}

~Human()

{

Console.WriteLine("Вызван деструктор");

}

public string GetName()

{

return \_name;

}

public DateTime Getbirthday()

{

return \_birthday;

}

public int GetStartYear()

{

return \_startYear;

}

public string GetSurename()

{

return \_surename;

}

public void SetName(string name)

{

\_name = name;

}

public void Setbirthday(DateTime birthday)

{

\_birthday = birthday;

}

public void SetStartYear(int startyear)

{

\_startYear = startyear;

}

public void SetSurename(string surename)

{

\_surename = surename;

}

public void AddHumanName()

{

Console.Write("Введите имя сотрудника: ");

while (true)

{

string str = Console.ReadLine();

if (CorrectImput.IsLatters(str))

{

SetName(str);

break;

}

}

}

public void AddHumanSurename()

{

Console.Write("Введите фамилию сотрудника: ");

while (true)

{

string str = Console.ReadLine();

if (CorrectImput.IsLatters(str))

{

SetSurename(str);

break;

}

}

}

public void AddHumanBirthday()

{

DateTime birthday;

Console.WriteLine("Введите день рождения сотрудника: ");

while (true)

{

var chek = DateTime.TryParse(Console.ReadLine(),out birthday);

if (!chek)

{

Console.Write("Некоректный ввод, попробуй ещё раз: ");

continue;

}

break;

}

Setbirthday(birthday);

}

public void AddHumanStartYear()

{

Console.Write($"Введите год начала работы(2000 - {DateTime.Now.Year}): ");

while (true)

{

var chek = int.TryParse(Console.ReadLine(), out int year) && CorrectImput.InRange(2000, DateTime.Now.Year, year) && (year - 18) >= \_birthday.Year;

if (!chek)

{

Console.Write("Некоректный ввод, попробуй ещё раз: ");

continue;

}

SetStartYear(year);

break;

}

}

public void NewEployment()

{

AddHumanName();

AddHumanSurename();

AddHumanBirthday();

AddHumanStartYear();

}

public virtual void Show()

{

Console.Write(

"Имя и Фамилия - " + \_name + " " + \_surename + "\n" +

$"День рождения - {\_birthday.ToString("D")}\n" +

"Год начала работы - " + \_startYear + "\n");

}

}

}

Листинг 1.4 – реализация наследственного класса Workers

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Xml.Linq;

namespace lab1

{

internal class Workers : Human

{

private string \_profession;

public Workers()

{

\_profession = "non";

Console.WriteLine("Вызван конструкотр без параметров наследовательного класса Workers");

}

public Workers(int startYear, string name, DateTime birthday, string surename, string profes) : base (startYear, name, birthday, surename)

{

\_profession = profes;

Console.WriteLine("Вызван конструкотр с параметрами наследовательного класса Workers");

Console.ReadKey();

}

public Workers(Workers workers): base(workers)

{

\_profession = workers.\_profession;

Console.WriteLine("Вызван конструкотр копирования наследовательного класса Workers");

}

~Workers()

{

Console.WriteLine("Вызван деструктор наследовательного класса Workers");

//Console.ReadKey();

}

public string GetProfession()

{

return \_profession;

}

public static string AddWorkersProfes()

{

Console.Write("Введите профессию сотрудника: ");

while (true)

{

string str = Console.ReadLine();

if (CorrectImput.IsLatters(str))

{

return str;

}

}

}

public override void Show()

{

Console.Write(

"Имя и Фамилия - " + GetName() + " " + GetSurename() + "\n" +

$"День рождения - {Getbirthday().ToString("D")}\n" +

"Год начала работы - " + GetStartYear() + "\n"+

"Профессия - " + \_profession + "\n" );

}

}

}

Листинг 1.5 – реализация наследственного класса Dismissed

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace lab1

{

internal class Dismissed : Human

{

private DateTime \_dismissedDate;

public Dismissed()

{

\_dismissedDate = new DateTime();

Console.WriteLine("Вызван конструкотр без параметров наследовательного класса Dismissed");

}

public Dismissed(int startYear, string name, DateTime birthday, string surename, DateTime date) : base(startYear, name, birthday, surename)

{

\_dismissedDate = date;

Console.WriteLine("Вызван конструкотр с параметрами наследовательного класса Dismissed");

}

public Dismissed(Dismissed dismissed) : base(dismissed)

{

\_dismissedDate = dismissed.\_dismissedDate;

Console.WriteLine("Вызван конструкотр копирования наследовательного класса Dismissed");

}

~Dismissed()

{

Console.WriteLine("Вызван деструктор наследовательного класса Dismissed");

//Console.ReadKey();

}

public DateTime GetDismissedDate()

{

return \_dismissedDate;

}

public static DateTime AddDismissedProfes(int startyear)

{

int day, month, year;

Console.WriteLine("Ввод Данных дня увольнения сотрудника\nВведите день(1 - 31): ");

while (true)

{

var chek = int.TryParse(Console.ReadLine(), out day) && CorrectImput.InRange(1, 31, day);

if (!chek)

{

Console.Write("Некоректный ввод, попробуй ещё раз: ");

continue;

}

break;

}

Console.Write("Введите месяц(1 - 12): ");

while (true)

{

var chek = int.TryParse(Console.ReadLine(), out month) && CorrectImput.InRange(1, 12, month);

if (!chek)

{

Console.Write("Некоректный ввод, попробуй ещё раз: ");

continue;

}

break;

}

Console.Write($"Введите год({startyear} - {DateTime.Now.Year}): ");

while (true)

{

var chek = int.TryParse(Console.ReadLine(), out year) && CorrectImput.InRange(startyear, DateTime.Now.Year, year);

if (!chek)

{

Console.Write("Некоректный ввод, попробуй ещё раз: ");

continue;

}

break;

}

return new DateTime(year, month, day);

}

public override void Show()

{

Console.Write(

"Имя и Фамилия - " + GetName() + " " + GetSurename() + "\n" +

$"День рождения - {Getbirthday().ToString("D")}\n" +

"Год начала работы - " + GetStartYear() + "\n" +

"Дата увольнения - " + \_dismissedDate.ToString("D") + "\n");

}

}

}

Листинг 1.6 – реализация проверки

namespace lab1

{

static class CorrectImput

{

public static bool IsLatters(string str)

{

foreach (char c in str)

{

if (!((c >= 'a' && c <= 'z') || (c >= 'A' && c <= 'Z') || (c >= 'а' && c <= 'я') || (c >= 'А' && c <= 'Я')))

{

Console.Write("Некоректный ввод, попробуй ещё раз: ");

return false;

}

}

return true;

}

public static bool InRange(int left, int right, int number)

{

if (number >= left && number <= right) return true;

return false;

}

}

}

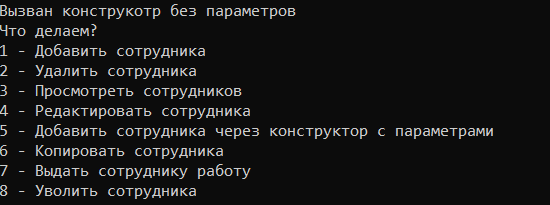


Рисунок 1. Начальное меню и конструктор без параметров

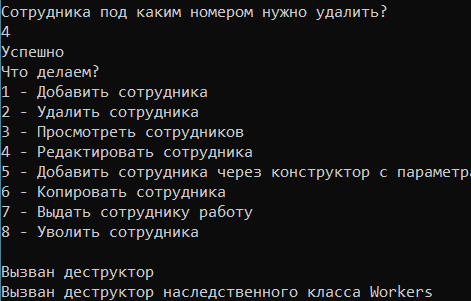


Рисунок 2. Вызван деструктор

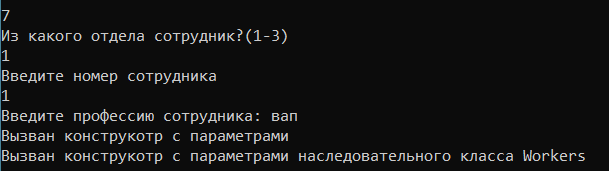


Рисунок 3. Вызван конструктор с параметрами наследственного класса

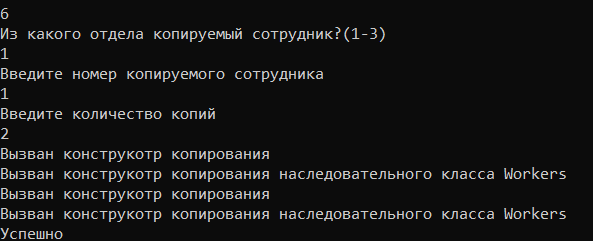


Рисунок 4. Вызван конструктор копирования наследственного класса

**Вывод:** В ходе данный лабораторной работе было изучено как создавать иерархию классов, создавать объекты базовых классов и классов-наследников, вызывать методы базового класса и класса-наследника, понимать влияние модификаторов доступа при наследовании, понимать порядок вызова конструкторов и деструкторов при наследовании.