Практическая работа № 4.  
Классы

**Цель практической работы**

Закрепление теоретических знаний по созданию классов.

**Постановка задачи**

**Объе́ктно-ориенти́рованное программирование (сокр. ООП)** – методология программирования, основанная на представлении программы в виде совокупности взаимодействующих **объектов**, каждый из которых является экземпляром определённого **класса**, а классы образуют **иерархию наследования**.

Идеологически ООП – подход к программированию как к моделированию информационных **объектов**, решающий на новом уровне основную задачу структурного программирования: структурирование информации с точки зрения управляемости, что существенно улучшает управляемость самим процессом моделирования, что, в свою очередь, особенно важно при реализации крупных проектов.

Описанием объекта является **класс**, а объект представляет экземпляр этого класса. Можно еще провести следующую аналогию. У нас у всех есть некоторое представление о человеке, у которого есть имя, возраст, какие-то другие характеристики. То есть некоторый шаблон - этот шаблон можно назвать классом. Конкретное воплощение этого шаблона может отличаться, например, одни люди имеют одно имя, другие - другое имя. И реально существующий человек (фактически экземпляр данного класса) будет представлять объект этого класса.

class название\_класса

{

// содержимое класса

}

class Person

{

}

**Создание объекта класса**

После определения класса мы можем создавать его объекты. Для создания объекта применяются **конструкторы**. По сути конструкторы представляют специальные методы, которые называются так же как и класс, и которые вызываются при создании нового объекта класса и выполняют инициализацию объекта.

Person tom = new Person(); // создание объекта класса Person

class Person // определение класса Person

{

}

Person tom = new Person(); // создание объекта класса Person

class Person // определение класса Person

{

public Person() //Конструктор

{

//Если конструктор пустой, то создавать его не обязательно

}

}

В строке Person tom = new Person(); Круглые скобки означают вызов конструктора.

Заметим, что для создания объекта всегда используется специальное слово new.

Чтобы конструктор был доступен вне класса Person используется модификатор public.

**Поля и методы класса**

Класс может хранить некоторые данные. Для хранения данных в классе применяются **поля**. По сути **поля класса** - это переменные, определенные на уровне класса.

Кроме того, класс может определять некоторое поведение или выполняемые действия. Для определения поведения в классе применяются методы.

class Person

{

public string name = "Undefined"; // имя

public int age; // возраст

public void Print()

{

Console.WriteLine($"Имя: {name} Возраст: {age}");

}

}

В отличие от переменных, определенных в методах, поля класса могут иметь **модификаторы**, которые указываются перед полем.

**Обращение к функциональности класса**

Для обращения к функциональности класса - полям, методам (а также другим элементам класса) применяется точечная нотация точки - после объекта класса ставится точка, а затем элемент класса:

using System;

Person tom = new Person(); // создание объекта класса Person

string personName = tom.name; // Получаем значение полей в переменные

int personAge = tom.age;

Console.WriteLine($"Имя: {personName} Возраст {personAge}");

// Имя: Undefined Возраст: 0

tom.name = "Tom"; // устанавливаем новые значения полей

tom.age = 37;

// обращаемся к методу Print

tom.Print(); // Имя: Tom Возраст: 37

class Person //Конструкторов может быть несколько

{

public string name;

public int age;

public Person()

{

name = "Неизвестно"; age = 18;

}

public Person(string n)

{

name = n; age = 18;

}

public Person(string n, int a)

{

name = n; age = a;

}

public void Print() =>

Console.WriteLine($"Имя: {name} Возраст: {age}");

}

**Цепочка вызова конструкторов**

В примере выше определены три конструктора. Все три конструктора выполняют однотипные действия - устанавливают значения полей name и age. Но этих повторяющихся действий могло быть больше. И мы можем не дублировать функциональность конструкторов, а просто обращаться из одного конструктора к другому также через ключевое слово **this**, передавая нужные значения для параметров:

class Person

{

public string name;

public int age;

public Person() : this("Неизвестно")

{

}

public Person(string name) : this(name, 18)

{

}

public Person(string name, int age)

{

this.name = name;

this.age = age;

}

public void Print() =>

Console.WriteLine($"Имя: {name} Воз-раст: {age}");

}

**Инициализаторы объектов**

Для инициализации объектов классов можно применять **инициализаторы**. Инициализаторы представляют передачу в фигурных скобках значений доступным полям и свойствам объекта:

Person tom = new Person { name = "Tom", age = 31 };

// или так

// Person tom = new() { name = "Tom", age = 31 };

tom.Print(); // Имя: Tom Возраст: 31

С помощью инициализатора мы можем установить значения **только доступных из вне** класса полей и свойств объекта. Например, в примере выше поля **name** и **age** имеют модификатор доступа public, поэтому они доступны из любой части программы.

Инициализатор выполняется после конструктора, поэтому если и в конструкторе, и в инициализаторе устанавливаются значения одних и тех же полей, и свойств, то значения, устанавливаемые в конструкторе, заменяются значениями из инициализатора.

Person tom = new Person { Name = "Tom", Company = { Title = "Microsoft" } };

tom.Print(); // Имя: Tom Компания: Microsoft

class Person

{

public string Name;

public Company Company;

public Person()

{

Name = "Undefined";

Company = new Company();

}

public void Print() => Console.WriteLine($"Имя: {Name} Компания: {Company.Title}");

}

class Company

{

public string Title;

}

**Задание на практическую работу**

1. Разработать консольное приложение для выполнения заданий.
2. Предусмотреть понятный, аккуратный интерфейс ввода и вывода данных на консоль.

**Задание 1.**

Создайте класс **StudentNotebook**, который будет хранить информацию о студенте и его учебных достижениях. Класс должен содержать:

1. **Поля класса**:
   * studentName (имя студента)
   * studentAge (возраст)
   * subjects (список предметов)
   * grades (словарь: предмет → список оценок)
2. **Конструктор**:
   * Принимает имя и возраст
   * Инициализирует пустые списки предметов и оценок
3. **Методы**:
   * AddSubject() - добавляет новый предмет
   * AddGrade() - добавляет оценку по предмету
   * GetAverageGrade() - вычисляет средний балл
   * PrintInfo() - выводит информацию о студенте
   * PrintGrades() - показывает все оценки

**Задание 2.**

Создайте класс Pet, который будет представлять домашнее животное. Класс должен содержать:

1. **Поля класса**:
   * name (кличка животного)
   * animalType (вид животного: кошка, собака и т.д.)
   * age (возраст в годах)
   * hungerLevel (уровень сытости от 0 до 100)
   * happinessLevel (уровень счастья от 0 до 100)
2. **Методы**:
   * Feed() - увеличивает сытость на 20 единиц (не больше 100)
   * Play() - увеличивает счастье на 15, уменьшает сытость на 10
   * PrintStatus() - выводит текущее состояние животного
   * NextYear() - увеличивает возраст на 1 год
   * IsHappy() - возвращает true, если счастье > 70 и сытость > 50
3. **Конструктор**:
   * Принимает кличку, вид и начальный возраст
   * Устанавливает сытость и счастье в 50

**Задание 3.**

Создайте класс Fraction, представляющий обыкновенную дробь (числитель/знаменатель). Класс должен содержать:

1. **Поля класса**:
   * numerator (числитель, целое число)
   * denominator (знаменатель, целое число ≠ 0)
2. **Конструкторы**:
   * Основной (принимает числитель и знаменатель)
   * Для целых чисел (знаменатель = 1)
3. **Методы**:
   * Add() - сложение дробей
   * Subtract() - вычитание дробей
   * Multiply() - умножение дробей
   * Divide() - деление дробей
   * Simplify() - сокращение дроби
   * ToDecimal() - преобразование в десятичную дробь
   * ToString() - строковое представление (например, "3/4")
4. **Дополнительно**:
   * Проверка на равенство дробей
   * Приведение к общему знаменателю

**Пример разработки приложения**

**Задание 1.**

Создайте класс **StudentNotebook**, который будет хранить информацию о студенте и его учебных достижениях. Класс должен содержать:

1. **Поля класса**:
   * studentName (имя студента)
   * studentAge (возраст)
   * subjects (список предметов)
   * grades (словарь: предмет → список оценок)
2. **Конструктор**:
   * Принимает имя и возраст
   * Инициализирует пустые списки предметов и оценок
3. **Методы**:
   * AddSubject() - добавляет новый предмет
   * AddGrade() - добавляет оценку по предмету
   * GetAverageGrade() - вычисляет средний балл
   * PrintInfo() - выводит информацию о студенте
   * PrintGrades() - показывает все оценки

**Шаг 1: Понимание задачи**

Требуется создать класс StudentNotebook, который будет хранить информацию о студенте (имя, возраст), его предметах и оценках по ним. Класс должен предоставлять методы для добавления предметов, добавления оценок, вычисления среднего балла, вывода информации о студенте и его оценках.

**Шаг 2: Определение структуры класса**

* **Поля**:
  + studentName (string) — имя студента.
  + studentAge (int) — возраст студента.
  + subjects (List<string>) — список предметов.
  + grades (Dictionary<string, List<int>>) — словарь, где ключ — предмет, а значение — список оценок по нему.
* **Конструктор**:
  + Принимает имя и возраст студента.
  + Инициализирует пустые списки subjects и grades.
* **Методы**:
  + AddSubject(string subject) — добавляет предмет в список.
  + AddGrade(string subject, int grade) — добавляет оценку по предмету.
  + GetAverageGrade() — вычисляет средний балл по всем предметам.
  + PrintInfo() — выводит имя и возраст студента.
  + PrintGrades() — выводит все оценки по предметам.

**Шаг 3: Реализация класса на C#**

public class StudentNotebook

{

// Поля класса

private string studentName;

private int studentAge;

private List<string> subjects;

private Dictionary<string, List<int>> grades;

// Конструктор

public StudentNotebook(string name, int age)

{

studentName = name;

studentAge = age;

subjects = new List<string>();

grades = new Dictionary<string, List<int>>();

}

// Метод для добавления предмета

public bool AddSubject(string subject)

{

if (subjects.Contains(subject))

return false;

subjects.Add(subject);

grades[subject] = new List<int>();

return true;

}

// Метод для добавления оценки по предмету

public bool AddGrade(string subject, int grade)

{

if (!subjects.Contains(subject))

return false;

grades[subject].Add(grade);

return true;

}

// Метод для вычисления среднего балла

public double GetAverageGrade()

{

if (grades.Count == 0)

return 0;

double sum = 0;

foreach (var (subject, subjectGrades) in grades)

{

sum += GetAverageGrade(subject);

}

return sum / grades.Count;

}

// Метод для вычисления среднего балла по предмету

public double GetAverageGrade(string subject)

{

if (grades.Count == 0)

return 0;

if (!grades.TryGetValue(subject, out var grages))

return 0;

return grages.Average();

}

// Метод для вывода информации о студенте

public void PrintInfo()

{

// Смешивание логики и I/O плохо, но мы оставим

Console.WriteLine($"Имя: {studentName}");

Console.WriteLine($"Возраст: {studentAge}");

}

// Метод для вывода всех оценок

public void PrintGrades()

{

// Смешивание логики и I/O плохо, но мы оставим

Console.WriteLine("Оценки:");

foreach (var subject in subjects)

{

Console.Write($"{subject}: ");

if (grades.ContainsKey(subject) && grades[subject].Count > 0)

{

Console.WriteLine(string.Join(", ", grades[subject]));

}

else

{

Console.WriteLine("нет оценок");

}

}

}

}

# Содержание пояснительной записки

1. Постановка задачи. Приводится теоретический материал, использованный при написании приложения.

2. Формулировка задания и вариант. Приводится задание на лабораторную работу и вариант этого задания.

3. Описание выполняемых действий. Необходимо привести описание последовательности разработки программы, реализации используемых методов, алгоритмов, блок-схем.

4. Анализ результатов. Привести анализ входных и выходных данных. Показать результаты выполнения программного кода. Предоставить скриншоты обработки тестовых примеров. Сделать выводы.

5. Листинг программы. Привести листинг разработанного программного кода, содержание файлов входных и выходных данных.

# Используемое программное обеспечение

1. Среда программирования MS Visual Studio Community 2022 (Свободно распространяемое программное обеспечение (в учебных целях));
2. Microsoft Office Standard 2007 (Open License: 42267924);
3. Open Office (Свободно распространяемое программное обеспечение).
4. Браузер (Свободно распространяемое программное обеспечение).

# Список литературы

* + - 1. Мейер Б. Объектно-ориентированное программирование и программная инженерия [Электронный ресурс]/ Мейер Б. – Электрон. текстовые данные. – М.: Интернет-Университет Информационных Технологий (ИНТУИТ), 2016. – 285 c.
      2. Биллиг, В. A. Основы объектного программирования на С# (C# 3.0, Visual Studio 2008) [Электронный ресурс] : учебное пособие / В. A. Биллиг. — Электрон. текстовые данные. — Москва, Саратов : Интернет-Университет Информационных Технологий (ИНТУИТ), Вузовское образование, 2017. — 583 c. — 978-5-4487-0145-0. — Режим доступа: http://www.iprbookshop.ru/72339.html
      3. Павловская, Т. А. Программирование на языке высокого уровня C# [Электронный ресурс] / Т. А. Павловская. — 2-е изд. — Электрон. текстовые данные. — М. : Интернет-Университет Информационных Технологий (ИНТУИТ), 2016. — 245 c. — 2227-8397. — Режим доступа: http://www.iprbookshop.ru/73713.html
      4. Агапов, В. П. Основы программирования на языке С# [Электронный ресурс] : учебное пособие / В. П. Агапов. — Электрон. текстовые данные. — М. : Московский государственный строительный университет, ЭБС АСВ, 2012. — 128 c. — 978-5-7264-0576-6. — Режим доступа: http://www.iprbookshop.ru/16366.html
      5. Медведев, М. А. Программирование на СИ# [Электронный ресурс] : учебное пособие / М. А. Медведев, А. Н. Медведев ; под ред. А. В. Присяжный. — Электрон. текстовые данные. — Екатеринбург : Уральский федеральный университет, ЭБС АСВ, 2015. — 64 c. — 978-5-7996-1561-1. — Режим доступа: http://www.iprbookshop.ru/69667.html
      6. Казанский А.А. Объектно-ориентированное программирование на языке Microsoft Visual С# в среде разработки Microsoft Visual Studio 2008 и .NET Framework. 4.3 [Электронный ресурс]: учебное пособие и практикум/ Казанский А.А.— Электрон. текстовые данные.— М.: Московский государственный строительный университет, ЭБС АСВ, 2011.— 180 c
      7. Уйманова Н.А. Основы объектно-ориентированного программирования [Электронный ресурс]: практикум/ Уйманова Н.А., Таспаева М.Г.— Электрон. текстовые данные.— Оренбург: Оренбургский государственный университет, ЭБС АСВ, 2017.— 156 c.— Режим доступа: http://www.iprbookshop.ru/78808.html.— ЭБС «IPRbooks»
      8. Новиков П.В. Объектно-ориентированное программирование [Электронный ресурс]: учебно-методическое пособие к лабораторным работам/ Новиков П.В.— Электрон. текстовые данные.— Саратов: Вузовское образование, 2017.— 124 c.— Режим доступа: http://www.iprbookshop.ru/64650.html.— ЭБС «IPRbooks»