**Міністерство освіти і науки України**

**Національний авіаційний університет**

**Навчально-науковий інститут комп’ютерних**

**інформаційних технологій**



Лабораторна робота №1

з дисципліни «Об’єктно орієнтоване програмування»

«ДОСЛІДЖЕННЯ РОБОТИ З ФАЙЛАМИ ТА ПРОЕКТУВАННЯ ЗВ’ЯЗКІВ МІЖ СУТНОСТЯМИ»

Виконав студент:

групи ПІ-223Б

Карпов І.Б.

Перевірив викладач:

Дишлевий О.П.

Київ 2022

**Мета:** дослідити роботу з файлами з використанням потоків у мові C# при роботі з різними пов’язаними сутностями.

**Завдання**

1. Дослідити структуру класів потокового введення/виведення для роботи з файлами у мові C#.
2. Спроектувати, представити у вигляді діаграми класів функціонал згідно з варіантом.
3. Розробити програму га мові C#, яка відповідає вимогам у завданні та варіанті. Для демонстрації роботи використати текстові повідомлення на консолі через операції вводу/виводу. Програма повинна показати застосування класів потокового введення/виведення для роботи з файлами шляхом запису, читання та маніпуляцій над даними спроектованих сутностей та зв’язків між ними.
4. Діаграма(-и) та вихідний код повинні відповідати базовим принципам проектування: ООП, composition over inheritance, loose coupling – high cohesion, inversion of control (IoC).
5. Для отримання балів, що відповідають «задовільно» необхідно реалізувати завдання, відповідно варіанту. Операції роботи із файлами, вводу-виводу повинні бути в окремих класах, а не в класах бізнес-сутностей.
6. Для отримання балів, що відповідають «добре» повинно бути реалізовано усе, що на «задовільно». А також:
   1. В окремий проект (project) рішення (solution) виділити операції читання-запису у файл. Тут не повинно бути ніяких ніяких маніпуляцій з даними чи ввід-вивід на консоль. Тобто в рішенні буде мінімум 2 проекти.
   2. Операції вводу-виводу на екран - в окремому класі **ConsoleMenu**.
   3. Також необхідно використати абстракції (абстрактні класи та інтерфейси), для демонстрації зв’язків між сутностями у варіанті.
7. Для отримання балів, що відповідають «відмінно», спроектувати зв’зки між сутностями таким чином, щоб можна було легко додавати інші пов’язані типи (наприклад, Pupil чи Musician) без необхідності змінювати існуючі. А також – нову поведінку. Наприклад, коли операцію Play() може виконувати музикант та студент.
8. \*Завдання підвищеної складності – працювати з одним єдиним файлом-джерелом даних, записуючи та читаючи дані різних типів та сутностей.

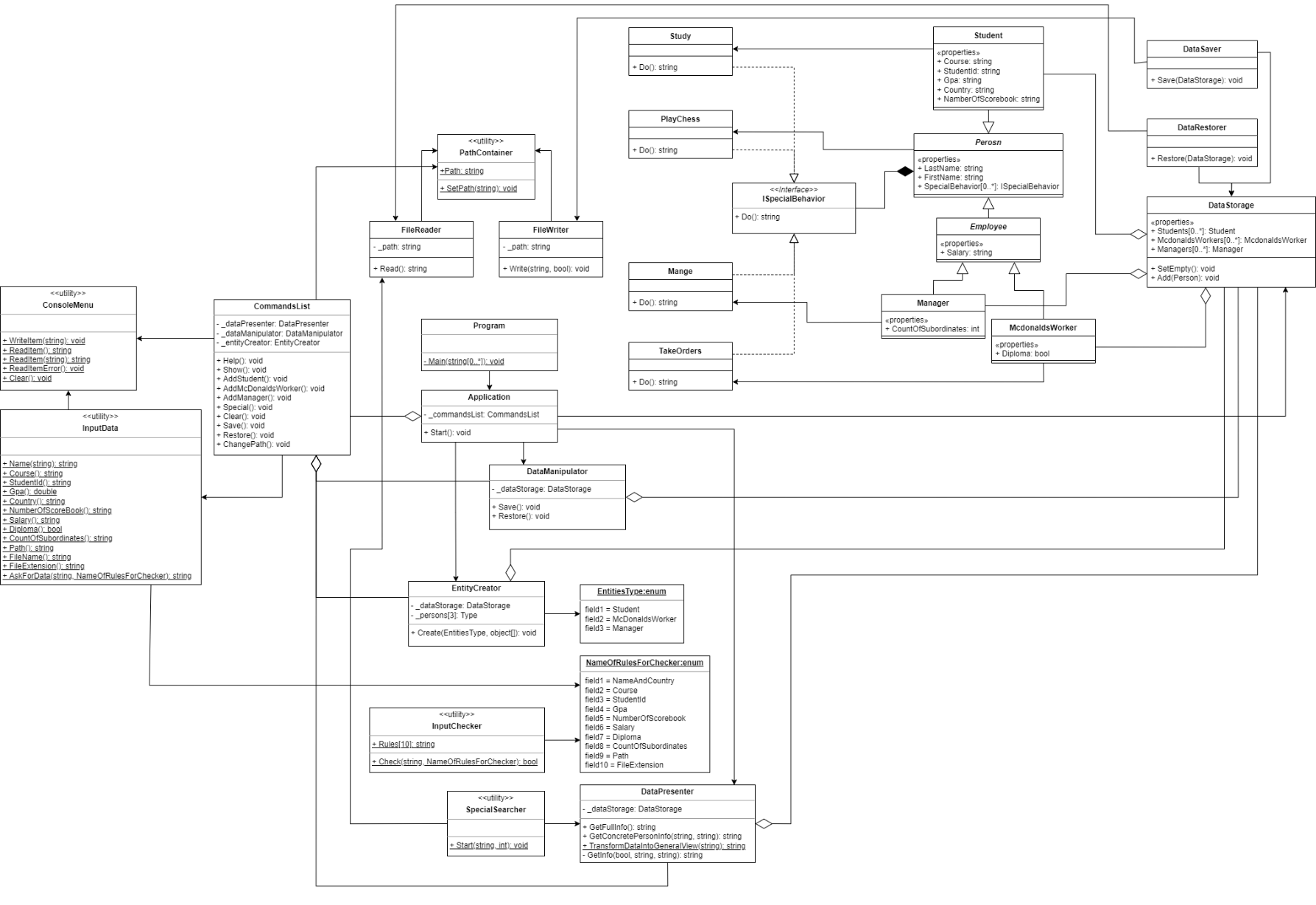
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Варіант | Елементи класу Student | Операції з студентами | Додаткові сутності | Додаткові вміння |
| 5 | Прізвище, курс, Студентський квиток, Середній бал, Країна, номер залікової книжки | Обчислити кількість студентів 3-го курсу, які проживають в Україні. Отримати їх дані. | McdonaldsWorker,  Manger | Грати в шахи |

**Хід роботи**

Діаграми та код програми зберігаються на GitHub за цим посиланням: <https://github.com/IvanKarpov-1/course2-term1-OOP-lab1>

В ході виконання лабораторної роботи було розроблено діаграму класів та код програми.

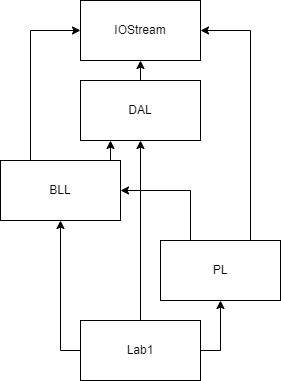
Діаграма класів для всього рішення виглядає наступним чином:



Програма була поділена на п’ять проектів:

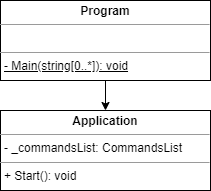
1. Lab1 – початкова точка запуску програми;
2. IOStream – потоковий запис та читання даних;
3. DAL – рівень доступу до даних;
4. BLL – рівень бізнес логіки програми;
5. PL – презентаційний рівень.

Відношення між проектами:

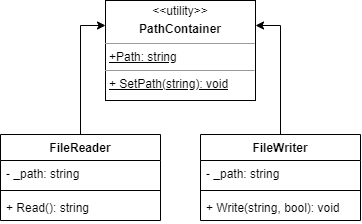


Детальніший розбір проектів:

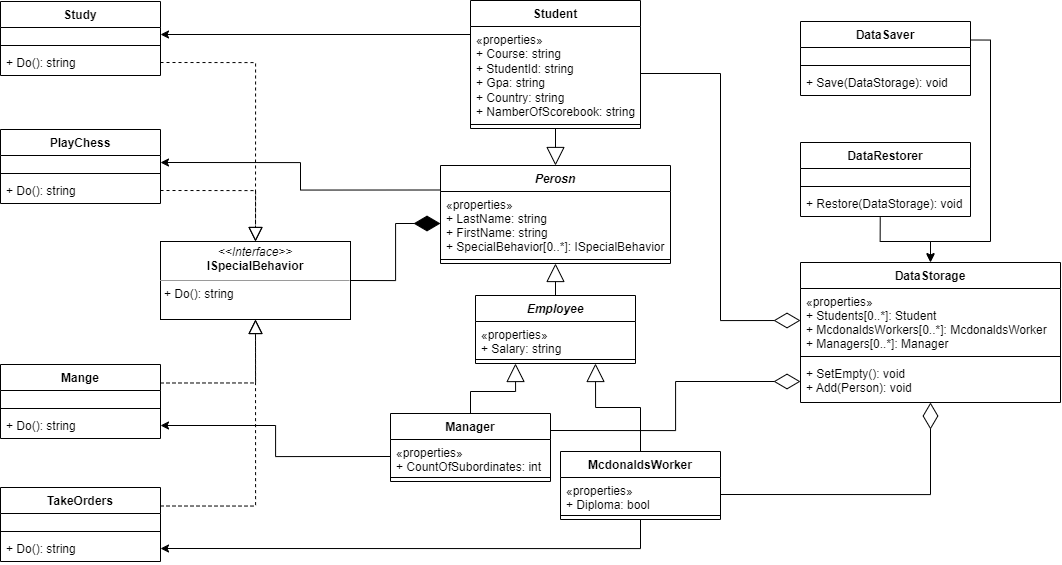
1. Lab1:



1. IOStream:



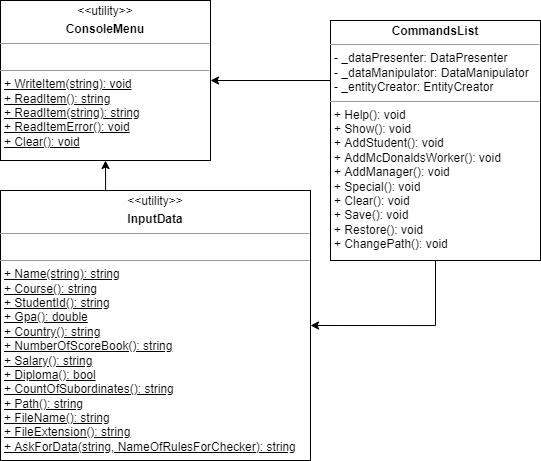
1. DAL:



1. BLL:



1. PL:



Написання коду розпочалось з розробки класів, що відповідають за потоковий запис/читання файлу. Ці класи реалізовують інтерфейс IDisposable щоб можна було їх використовувати в конструкції using.

public class FileReader : IDisposable

{

private readonly string \_path = PathContainer.Path;

public FileReader()

{

}

public FileReader(string path)

{

\_path = path;

}

public string Read()

{

string text;

using (var file = new StreamReader(\_path))

{

text = file.ReadToEnd();

}

return text;

}

public void Dispose()

{

}

}

public class FileWriter : IDisposable

{

private readonly string \_path = PathContainer.Path;

public FileWriter()

{

}

public FileWriter(string path)

{

\_path = path;

}

public void Write(string data, bool append)

{

using (var file = new StreamWriter(\_path, append))

{

file.Write(data);

}

}

public void Dispose()

{

}

}

public static class PathContainer

{

public static string Path { get; private set; } = AppDomain.CurrentDomain.BaseDirectory + "DB.txt";

public static void SetPath(string path)

{

Path = path;

}

}

Дані сутностей зберігаються в класі DataStorage проекту DAL у вигляді трьох масивів. Під час роботи, програма записує сюди дані, а потім, коли потрібно, класи DataSaver та DataRestorer записують чи зчитують ці дані в та з файлу шляхом використання попередньо показаних класів.

Код методу Save класу DataSaver, який форматує дані до потрібного вигляду, та викликає метод Write класу FileWriter:

public void Save(DataStorage dataStorage)

{

var data = "";

foreach (var property in typeof(DataStorage).GetProperties())

{

var valuesFromPropertyArray = (Array)property.GetValue(dataStorage);

for (var i = 0; i < valuesFromPropertyArray.Length; i++)

{

var singleValue = valuesFromPropertyArray.GetValue(i);

var typeOfValue = singleValue.GetType();

data += $"{singleValue.ToString().Remove(0, 4)} ";

data += $"{typeOfValue.GetProperty("FirstName")?.GetValue(singleValue)}";

data += $"{typeOfValue.GetProperty("LastName")?.GetValue(singleValue)}\n";

data += "{ ";

foreach (var valuesProperty in typeOfValue.GetProperties(BindingFlags.Instance |

BindingFlags.NonPublic |

BindingFlags.Public | BindingFlags.Static).Reverse())

{

if (valuesProperty.PropertyType.IsArray == false)

{

data += $"\"{valuesProperty.Name}\": \"{valuesProperty.GetValue(singleValue)}\",\n";

}

}

data += "};\n";

data = data.Replace(",\n};", "};");

}

}

using (var fileWriter = new FileWriter())

{

fileWriter.Write(data, false);

}

}

Для того, щоб можна було додавати інші сутності, при цьому не змінюючи існуючий код, було прийнято рішення використовувати рефлексію.

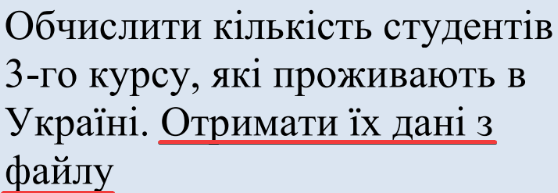
Клас DataRestorer теж використовує рефлексію, а також регулярні вирази, для записування зчитаних даних до масиві.

За перетворення даних до зручного для читання вигляду, щоб потім їх модна було вивести на консоль, відповідає клас DataPresenter проекту BLL. Він має методи GetFullInfo() для отримання повної інформації та GetConcretePersonInfo(string, string) для отримання інформації про конкретних людей за їх Іменем та Прізвищем.

Весь інтерфейс та методи роботи з ним знаходяться в проекті PL. Клас ConsoleMenu відповідає безпосередньо за ввід/вивід даних на консоль. В класі CommandsList знаходяться команди, які викликає користувач в процесі роботи. Клас InputData відповідає за ввід конкретних коректних даних.

Для перевірки коректності введених даних використовується клас InputChecker проекту BLL. Клас DataManipulator цього ж проекту слугує проміжним класом, що викликає методи Save та Restore класів DataSaver та DataRestorer. Клас EntityCreator створює об’єкти класів сутностей за переданими йому даними, що ввів користувач, щоб зберегти їх до бази даних.

Оскільки, за завданням, операції зі студентами потрібно проводити звертаючись напряму до фалу, був створений клас SpecialSearcher.

Він відповідає за обчислення кількості студентів 3-го курсу, які проживають в Україні, та за отримання їх даних.

В проекті Lab1 знаходиться точка входу в програму, а також клас Application, який розуміє ввід користувача і викликає відповідні команди.

**Відповіді на контрольні запитання**

1. **Що таке потік?**

Потік — це основна одиниця, якій операційна система виділяє час процесора.

1. **Наведіть приклади режимів відкриття файлів.**

Режимів відкривання існує 6 варіантів: Append, Create, CreateNew, Open, OpenOrCreate, Truncate.

Наступний приклад відкриє існуючий файл.

var stream = new FileStream(name, FileMode.Open, FileAccess.Read);

1. **Поясніть різницю між файлами з послідовним та довільним доступом.**

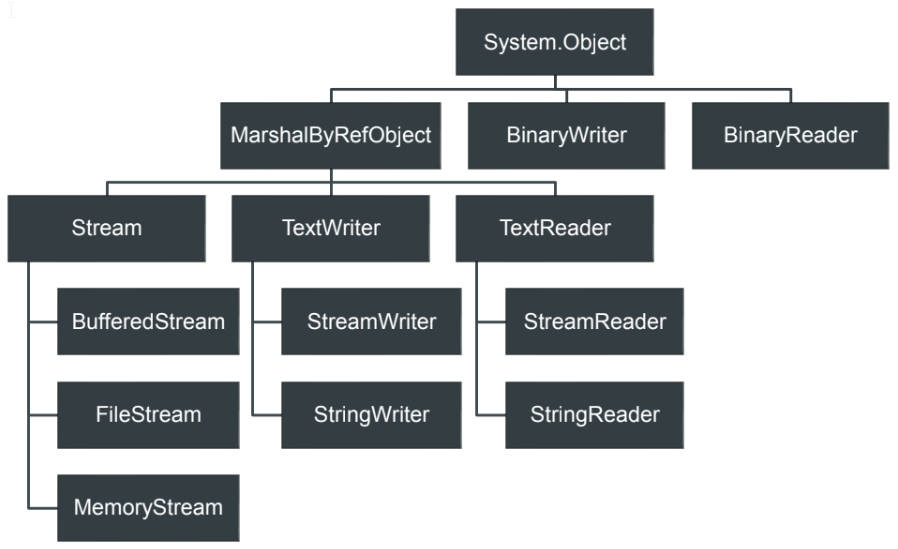
Файли з послідовним доступом зберігають інформацію в неструктурованому вигляді і щоб отримати якусь конкретну інформацію з файла, його потрібно починати читати від самого початку і порівнювати інформацію з «шуканою».

Файли з довільним доступом в свою чергу зберігають інформацію в структурованому вигляді. Пошук в таких файлах здійснюється в області адрес і завершується зверненням безпосередньо до шуканої ділянки.

1. **Поясніть, як реалізується довільний доступ у файлах.**

Дисковий простір таких файлів поділено на однакові ділянки, що мають однакову структуру полів. І знаючи адресу ділянки можна отримати доступ до її даних.

1. **Опишіть ієрархію класів потокового введення/виведення в С#.**



1. **Наведіть методи читання/запису для потоку FileStream в C#.**

var stream1 = new FileStream(name, FileMode.OpenOrCreate, FileAccess.Write);

var stream2 = new FileStream(name, FileMode.Open, FileAccess.Read);

1. **Поясніть, як створити символьний потік в C#.**

Для виконання операцій символьного вводу/виводу об’єкти класа FileStream укладаються в оболонку класса StreamWrite чи StramReader оскільки в цих класах виконується автоматичне перетворення байтового потоку в символьний.

var path = "data.txt";

var text = "abc";

var stream = new FileStream(path, FileMode.OpenOrCreate, FileAccess.Write);

var streamWriter = new StreamWriter(stream);

streamWriter.WriteLine(text);

1. **Поясніть, як виконується перенаправлення потоків в C#.**

Якщо програма викликається з командної строки можна використовувати символи “>” та “<” для перенаправлення Console.In і Console.Out відповідно.

В коді можна використовувати методи класу Console:

* static void SetIn(TextReader input);
* static void SetOut(TextWriter output);
* static void SetError(TextWriter output);

Увесь потік, що успадкований від класів TextReader чи TextWriter, можна застосовувати як перенаправлений.

1. **Наведіть приклади методів для читання/запису значень наперед визначених типів у двійкові потоки в C#.**

using (var writer = new BinaryWriter(File.Open("DB.txt", FileMode.OpenOrCreate)))

{

writer.Write("Hello");

}

using (var reader = new BinaryReader(File.Open("DB.dat", FileMode.Open)))

{

var name = reader.ReadString();

}

1. **Посніть використання регулярних виразів.**

Регулярні вирази, це шаблони, за якими здійснюється пошук в тексті.

Наприклад за регулярним виразом “^[0-9].\*[QW]$” буде здійснюватись пошук фраз які: 1) починаються з любої цифри ( ^[0-9] ), 2) потім слідують любі символи ( .\* ), 3) фраза повинна закінчуватись або буквою Q, або W.

За допомогою регулярних виразів можна здійснювати пошук в тексті, робити заміну, перевіряти правильність введених даних.

1. **В чому відмінність між композицією та наслідуванням. Який зв’язок і коли варто використовувати? Чому?**

При наслідуванні похідний клас переймає всі поля, властивості та методи первісного класу, тобто наслідує стан і поведінку. Такий тип зв’язку називається як “Is-a”.

При композиції, клас ціле зберігає в собі класи частини. Такий тип зв’язку називається “Has-a”.

Якщо ми розробляємо якісь сутності (наприклад, як в цій лабораторній роботі) і ми знаємо, що в цих сутностей є щось спільне, це спільне можна винести в окремий клас і від нього унаслідуватись. В даному випадку, якщо потрібно буде здійснити якісь зміни в цих спільних частинах, змінювати доведеться лише один клас.

Композиція, це коли для роботи класу потрібен інший клас, але унаслідувати його неможна. В такому випадку створюється закрите поле чи властивість з типом класу, який потрібен для роботи. При композиції, клас ціле визначає час існування класу частини.

1. **Поясніть твердження «composition over inheritance». Чому воно виникло?**

Composition over inheritance - це принцип, згідно з яким класи повинні досягати поліморфної поведінки та повторного використання шляхом використання композиції.

1. **В чому різниця між абстрактним класом та інтерфейсом?**

Абстрактні класи, як правило, представляють абстрактне поняття або сутність із частковим виконанням або відсутністю. З іншого боку, інтерфейс - це абстрактний тип, який використовується для вказівки контракту, який повинен бути реалізований класами. Абстрактні класи слід успадковувати (або розширювати), тоді як інтерфейси повинні бути реалізовані. Абстрактні класи можуть містити абстрактні методи, тоді як інтерфейс повинен містити лише абстрактні методи. Абстрактні класи можуть містити будь-які змінні, але інтерфейси можуть визначати лише константи. Клас не може успадковувати більше ніж одного абстрактного класу, але може реалізувати кілька інтерфейсів. Інтерфейс не може реалізувати інший інтерфейс. Однак інтерфейс може розширити клас.

1. **В чому полягає користь використання абстракцій замість конкретних реалізацій? Наведіть приклади.**

В такому випадку код легко розширювати та підтримувати, легше робити зміни (краще змінити щось один раз для всіх класів, ніж змінювати це в кожному класі окремо).

Прикладом може слугувати ця лабораторна робота. Тут щоб створити три сутності я створив два додаткові абстрактні класи і виніс в них спільні частини. Тепер я не тільки можу легко робити потрібні зміни, а і можу використовувати абстрактні класи у вигляді потрібних аргументів в методах з можливістю передавати в метод будь-які класи наслідники.

1. **В чому полягає принцип інверсії контролю (inversion of control – IoC)?**

Інверсія контролю (IoC) — це принцип. Він використовується для інвертування різних видів елементів для досягнення слабкого зв’язку. Тут засоби контролю стосуються будь-яких додаткових обов’язків, які має клас, крім основної відповідальності. Це включає контроль над потоком програми та контроль над потоком створення об’єкта або створення та зв’язування залежного об’єкта.