МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Національний аерокосмічний університет ім. М. Є. Жуковського

«Харківський авіаційний інститут»

факультет програмної інженерії та бізнесу

кафедра інженерії програмного забезпечення

**Проєкт частина Б**

з дисципліни « Об’єктно орієнтоване програмування »

*назва дисципліни*

на тему: «Успадкування. Абстрактні класи. Поліформізм»

Виконав: студент 2 курсу групи № 622п

освітньої програми

121 інженерія програмного забезпечення

(шифр і назва ОП)

Зайченко Ярослав Ігорович

(прізвище й ініціали студента)

Прийняв: доц. Вдовітченко О.В.

(посада, науковий ступінь, прізвище й ініціали)

Кількість балів:

Харків – 2023

# ПОСТАНОВКА ЗАДАЧІ

1. Удосконалити проєкт, скорегував діаграму класів, додавши реалізацію хоча б одним класом будь-якого інтерфейсу(ів) .NET (IClonable, IComparable, IEnumereable тощо).
2. Передбачити множинну реалізацію інтерфейсів.
3. Додати хоча б один абстрактний клас, від якого успадкувати декілька нових чи вже існуючих класів. Бажано, щоб абстрактний клас мав не тільки абстрактні методи, але й абстрактні властивості.
4. Передбачити хоча б один варіант реалізації поліморфізму:

* **Варіант 1:** об’єкти похідного класу обробляються як об’єкти базового класу (в параметрах методів, в колекціях та ін.);
* **Варіант 2:** перевизначення успадкованих методів, визначення типу об’єкта під час виконання програми і виклик відповідного методу.

1. Скорегувати каркас проєкту.
2. Замінити заглушки в проєкті конкретними реалізаціями методів, властивостей тощо.
3. Запустити unit-тести, досягти проходження всіх unit-тестів.
4. Спроєктувати меню додатку, пункти меню мають відповідати предметній області і бути логічними.
5. Реалізувати основну програму (файл Program.cs) у відповідності до спроєктованого меню.
6. Виконати функціональне тестування всіх пунктів меню.
7. Оформити звіт:

* Титульний аркуш
* Завдання
* Опис предметної області
* Діаграма класів
* Демонстрація реалізації поліморфізму
* Результати запуску unit-тестів.
* Результати функціонального тестування
* Програмний код класів
* Програмний код основної програми

ХІД РОБОТИ

Опис предметної області

Шкільна система розроблена для управління навчальним процесом у навчальному закладі. Система включає в себе основні об'єкти та взаємодії між ними, які відображають типові аспекти навчання та організації шкільного життя.

**Об'єкти системи:**

Учень:

* Властивості: Ім'я, прізвище, вік.
* Взаємодія: Учень може вивчати кілька предметів.

Предмет:

* Властивості: Назва предмету.
* Взаємодія: Предмет може мати викладача та бути вивченим декількома учнями.

Клас (Група учнів):

* Властивості: Список учнів, класний керівник.
* Взаємодія: Клас складається з учнів та має класного керівника.

Вчитель:

* Властивості: Ім'я, прізвище.
* Взаємодія: Вчитель може вести декілька предметів та мати список учнів.

**Відносини між об'єктами:**

Асоціація:

* Учень та Предмет: Учень асоційований з предметом, оскільки може вивчати кілька предметів.

Агрегація:

* Клас та Учень: Клас агрегує учнів, оскільки клас складається з учнів.

Композиція:

* Розклад та Предмет: Розклад використовує композиційне відношення з предметом, оскільки розклад може визначати конкретні предмети.

Ця шкільна система дозволяє ефективно керувати інформацією про учнів, вчителів, предмети та їх взаємодії, сприяючи організації навчання в навчальному закладі.

Діаграма класів

Рисунок 1 – Згенерована діаграма класів на основі файлів програми в середовищі розробки

Демонстрація реалізації поліморфізму

У розробці було використано концепцію поліморфізму для обробки об'єктів похідного класу ElementaryStudent як об'єктів базового класу Student.

Детальний програмний код базового та похідного класів подано в додатках **Ф та Г**.

Даний підхід дозволяє передавати екземпляри класу ElementaryStudent та Student через параметри методів, зберігати їх у колекціях та інших структурах даних, якщо очікується об'єкт базового класу Student.

Приклад використання поліморфізму подано нижче:

List<Student> studentsList = new List<Student>();

// Створення об'єктів різних класів, але базового типу

Student student1 = new Student("Іван", "Іванов", 20);

ElementaryStudent elementaryStudent1 = new ElementaryStudent("Петро", "Петров", 8);

// Додавання об'єктів до колекції типу Student

studentsList.Add(student1);

studentsList.Add(elementaryStudent1);

// Виклик методів базового класу на об'єктах похідного класу

foreach (Student student in studentsList)

{

student.ChooseSubject(new Subject("Математика"));

}

Результати запуску unit-тестів.

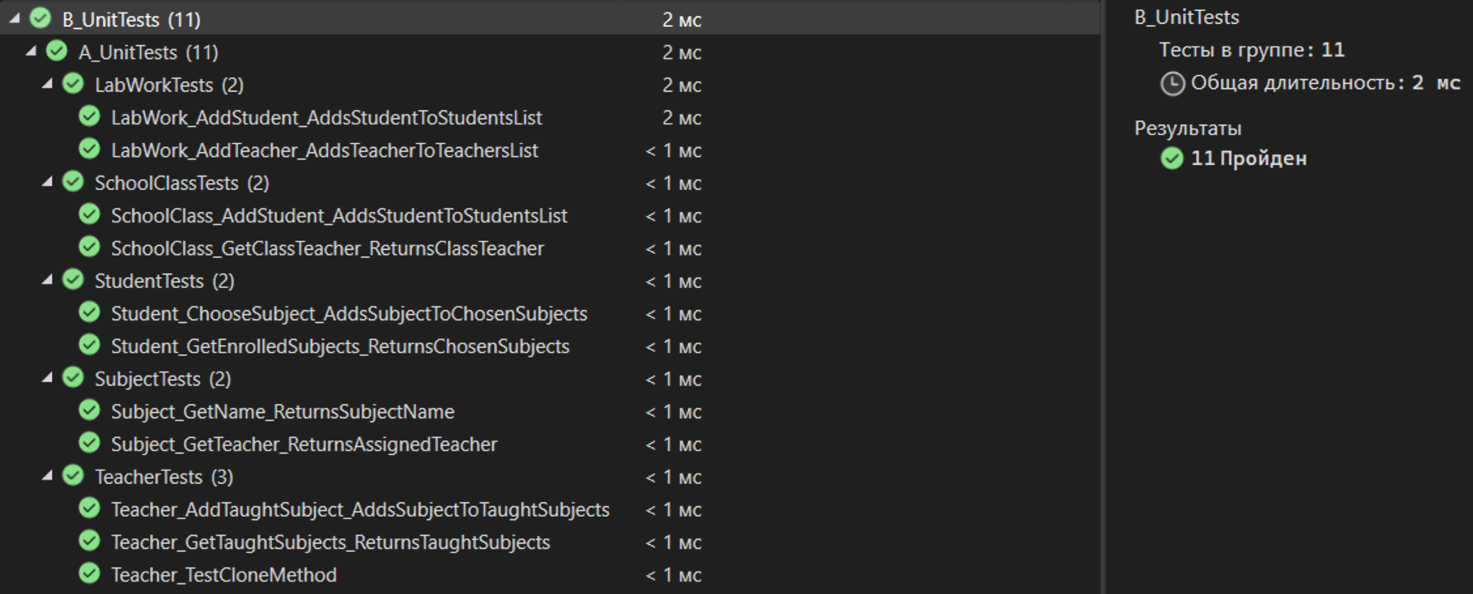


Рисунок 2 – всі тести розроблені раніше пройдені успішно

Результати функціонального тестування

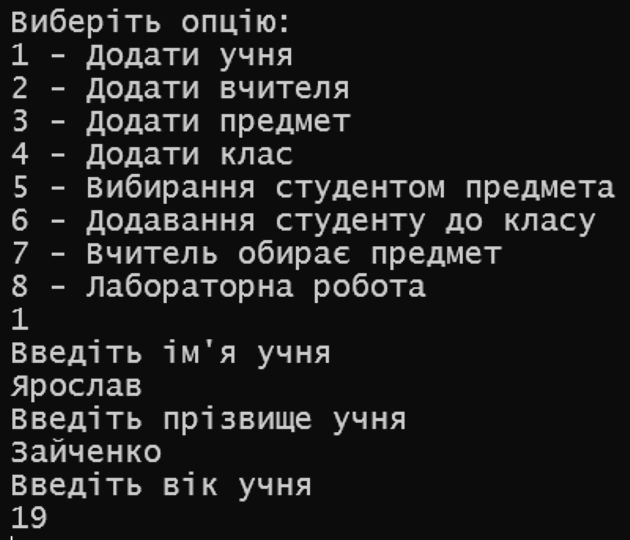


Рисунок 3 – додавання учня в програму

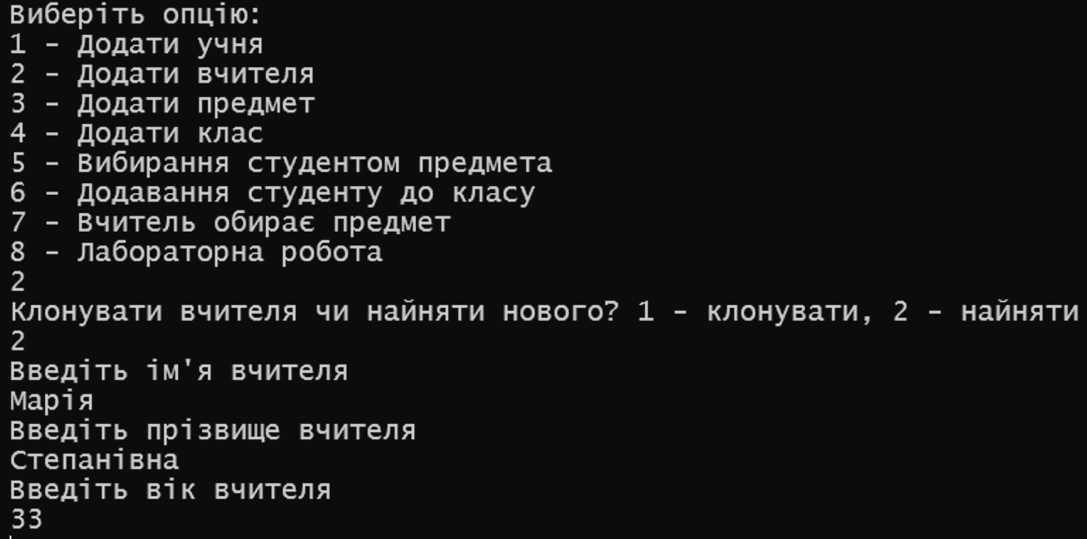


Рисунок 4 – додавання вчителя в програму

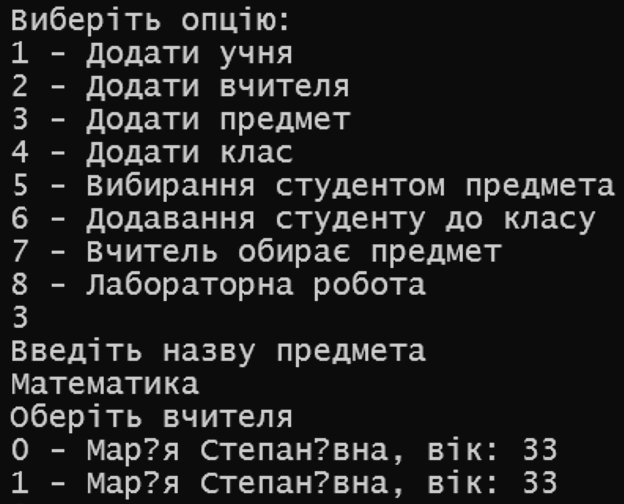


Рисунок 5 – додавання предмета в програму

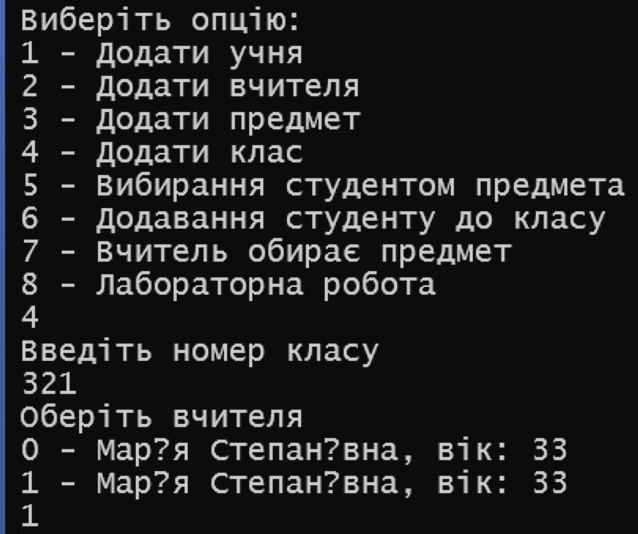


Рисунок 6 – додавання класу з вибором класного керівника в програму

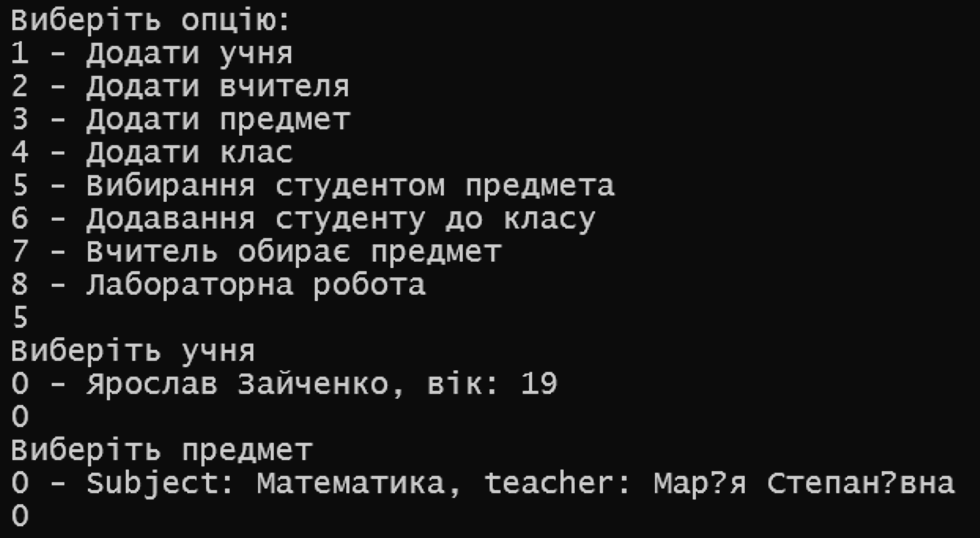


Рисунок 7 – вибір учнем предмету

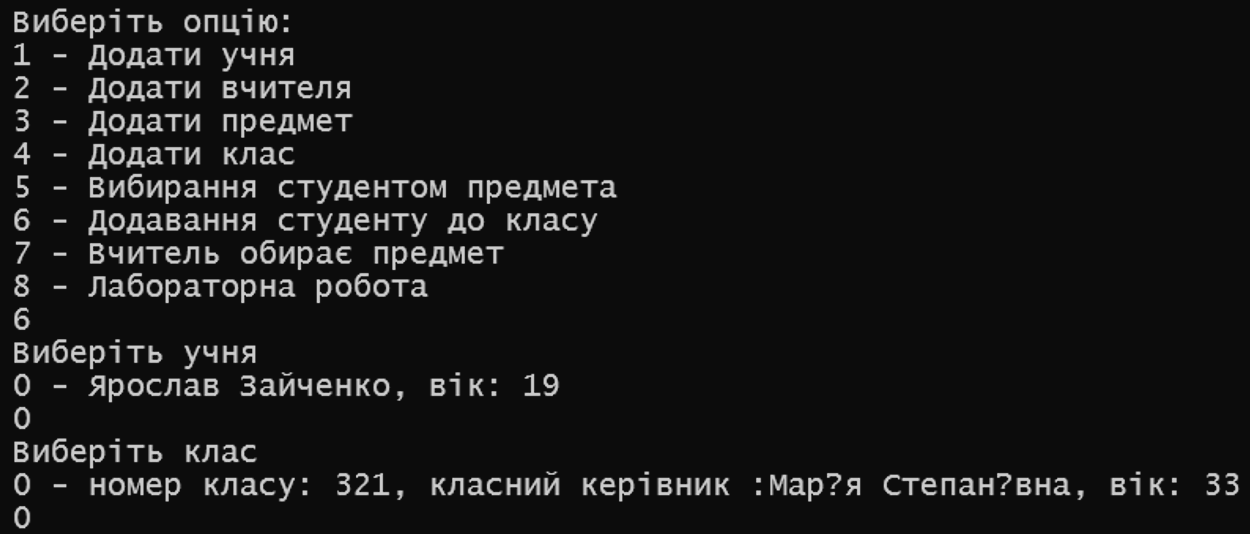


Рисунок 8 – додавання учня до класу

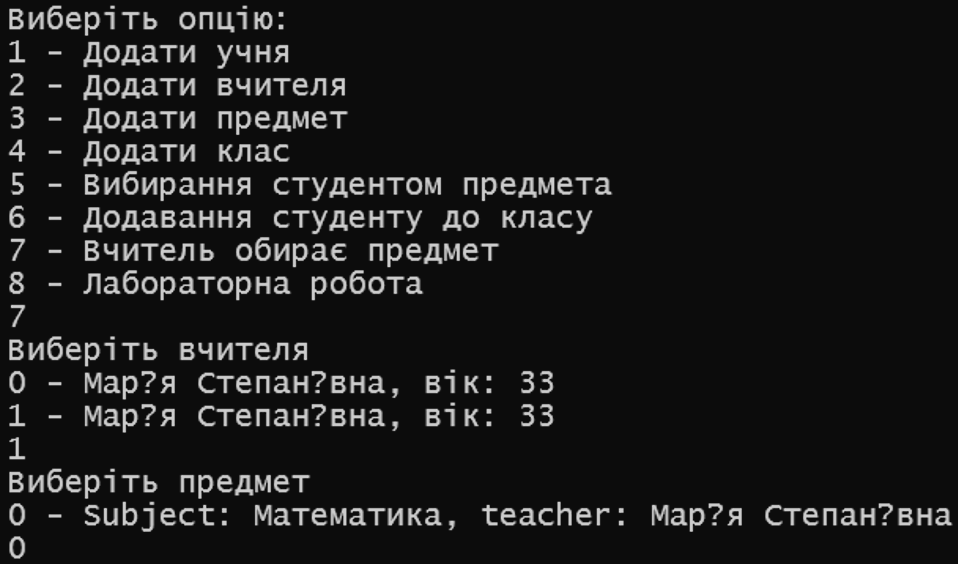


Рисунок 9 – надання предмету викладачу

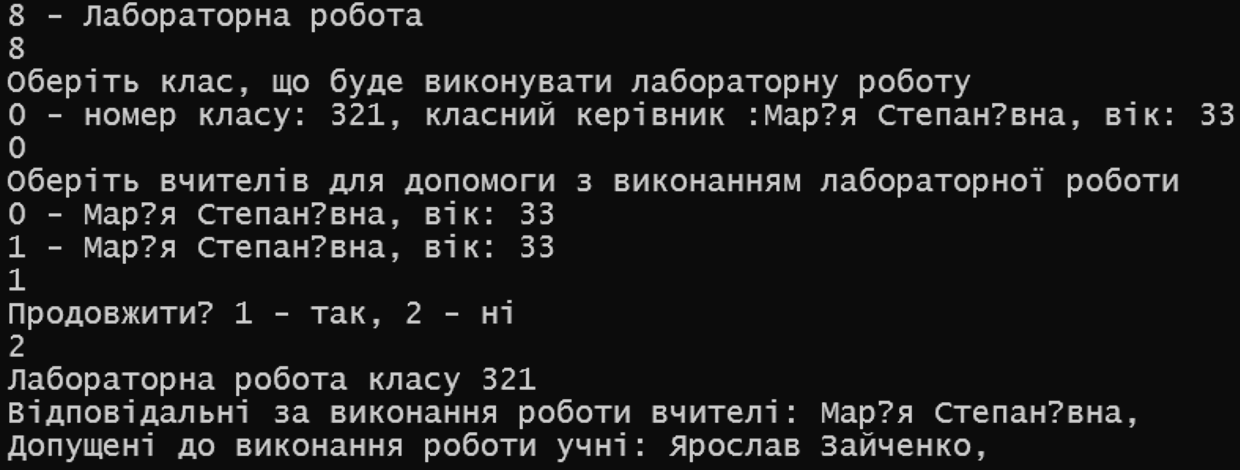


Рисунок 10 – проведення лабораторної, з вибором класу та викладача

Реалізація коду основної програми

Детальний код основної програми можна переглянути в додатку А.

Реалізація класів програми

Детальний код класів можна переглянути в додатках Б - З.

# ВИСНОВОК

Важливим аспектом удосконалення була додаткова імплементація інтерфейсів .NET, таких як IClonable, IComparable, IEnumereable. Це дозволило розширити можливості класів у проєкті та використовувати стандартні механізми клонування, порівняння та перебору.

Множинна реалізація інтерфейсів стала важливим кроком у забезпеченні гнучкості та розширюваності коду. Кожен клас може тепер мати низку різних інтерфейсів, що визначає його поведінку та можливості.

Додавання абстрактного класу, який має не тільки абстрактні методи, але й абстрактні властивості, розширило базовий функціонал та стало основою для успадкування нових та існуючих класів. Це сприяло створенню єдиної архітектури з відокремленням загальної логіки.

У реалізації поліморфізму обрано варіант, де об'єкти похідного класу обробляються як об'єкти базового класу. Це надає можливість використовувати об'єкти похідного класу там, де очікується об'єкт базового класу, що робить код більш гнучким та розширюваним.

У підсумку, завдяки цим змінам, проєкт став більш структурованим, гнучким та піддається легким модифікаціям. Основна програма була спроєктована відповідно до предметної області, що забезпечує користувачам логічний та зручний інтерфейс. Усі ці кроки були важливими для досягнення мети проєкту та надання високоякісного, розширюваного та тестованого програмного продукту.

***Додаток А***Машинний лістинг програми **Program.cs**

***Додаток А***Машинний лістинг класу **SchoolSystem.cs**

using Project\_part\_A.Classes;

namespace Project\_part\_A

{

public class SchoolSystem

{

private List<SchoolClass> Classes { get; }

private List<Teacher> Teachers { get; }

private List<Student> Students { get; }

public SchoolSystem(List<SchoolClass> classes, List<Teacher> teachers, List<Student> students)

{

throw new NotImplementedException();

}

public void AddClass(SchoolClass schoolClass)

{

throw new NotImplementedException();

}

public List<SchoolClass> GetClasses()

{

throw new NotImplementedException();

}

public void AddTeacher(Teacher teacher)

{

throw new NotImplementedException();

}

public List<Teacher> GetTeachers()

{

throw new NotImplementedException();

}

public void AddStudent(Student student)

{

throw new NotImplementedException();

}

public List<Student> GetStudents()

{

throw new NotImplementedException();

}

static void Main(string[] args)

{

Console.WriteLine("Hello, World!");

}

}

}

***Додаток Б***Машинний лістинг класу **LabWork.cs**

using Project\_part\_A.Interfaces;

namespace Project\_part\_A.Classes

{

public class LabWork : IPrintable

{

public List<SchoolClass> Classes { get; set; } = new List<SchoolClass>();

public List<Teacher> Teachers { get; set; } = new List<Teacher>();

public List<Student> Students { get; set; } = new List<Student>();

public void AddClass(SchoolClass schoolClass)

{

throw new NotImplementedException();

}

public List<SchoolClass> GetClasses()

{

throw new NotImplementedException();

}

public void AddTeacher(Teacher teacher)

{

throw new NotImplementedException();

}

public List<Teacher> GetTeachers()

{

throw new NotImplementedException();

}

public void AddStudent(Student student)

{

throw new NotImplementedException();

}

public List<Student> GetStudents()

{

throw new NotImplementedException();

}

public void PrintToDisplay()

{

throw new NotImplementedException();

}

}

}

***Додаток В***

Машинний лістинг класу **SchoolClass.cs**

namespace Project\_part\_A.Classes

{

public class SchoolClass

{

public int ClassNumber { get; set; }

public Teacher ClassTeacher { get; set; }

public List<Student> Students { get; set; } = new List<Student>();

public SchoolClass(int classNumber, Teacher classTeacher)

{

throw new NotImplementedException();

}

public void AddStudent(Student student)

{

throw new NotImplementedException();

}

public List<Student> GetStudents()

{

throw new NotImplementedException();

}

public Teacher GetClassTeacher()

{

throw new NotImplementedException();

}

}

}

***Додаток Г***Машинний лістинг класу **Student.cs**

using Project\_part\_A.Interfaces;

namespace Project\_part\_A.Classes

{

public class Student : IPerson

{

public string FirstName { get; set; }

public string LastName { get; set; }

public int Age { get; set; }

public List<Subject> ChosenSubjects { get; set; } = new List<Subject>();

public Student(string firstName, string lastName, int age)

{

throw new NotImplementedException();

}

public void ChooseSubject(Subject subject)

{

throw new NotImplementedException();

}

public List<Subject> GetEnrolledSubjects()

{

throw new NotImplementedException();

}

}

}

*Додаток Ґ*Машинний лістинг класу Subject.cs

namespace Project\_part\_A.Classes

{

public class Subject

{

public string Name { get; set; }

public Teacher Teacher { get; set; }

public Subject(string name, Teacher teacher)

{

throw new NotImplementedException();

}

public string GetName()

{

throw new NotImplementedException();

}

public Teacher GetTeacher()

{

throw new NotImplementedException();

}

}

}

***Додаток Д***Машинний лістинг класу **Teacher.cs**

using Project\_part\_A.Interfaces;

namespace Project\_part\_A.Classes

{

public class Teacher : IPerson

{

public string FirstName { get; set; }

public string LastName { get; set; }

public int Age { get; set; }

public List<Subject> TaughtSubjects { get; set; } = new List<Subject>();

public Teacher(string firstName, string lastName)

{

throw new NotImplementedException();

}

public void AddTaughtSubject(Subject subject)

{

throw new NotImplementedException();

}

public List<Subject> GetTaughtSubjects()

{

throw new NotImplementedException();

}

}

}

***Додаток Е***Машинний лістинг класу **IPerson.cs**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Project\_part\_A.Interfaces

{

public interface IPerson

{

string FirstName { get; set; }

string LastName { get; set; }

int Age { get; set; }

}

}

***Додаток Є***Машинний лістинг класу **IPrintable.cs**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Project\_part\_A.Interfaces

{

public interface IPrintable

{

void PrintToDisplay();

}

}

***Додаток Ж***Машинний лістинг класу **LabWorkUnitTest.cs**

using Project\_part\_A.Classes;

namespace A\_UnitTests

{

[TestClass]

public class LabWorkTests

{

[TestMethod]

public void LabWork\_AddClass\_AddsClassToClassesList()

{

// Arrange

var labWork = new LabWork();

var schoolClass = new SchoolClass(12, new Teacher("Mr.", "Davis"));

// Act

labWork.AddClass(schoolClass);

// Assert

Assert.IsTrue(labWork.GetClasses().Contains(schoolClass));

}

[TestMethod]

public void LabWork\_AddTeacher\_AddsTeacherToTeachersList()

{

// Arrange

var labWork = new LabWork();

var teacher = new Teacher("Mrs.", "Smith");

// Act

labWork.AddTeacher(teacher);

// Assert

Assert.IsTrue(labWork.GetTeachers().Contains(teacher));

}

[TestMethod]

public void LabWork\_AddStudent\_AddsStudentToStudentsList()

{

// Arrange

var labWork = new LabWork();

var student = new Student("Jane", "Doe", 16);

// Act

labWork.AddStudent(student);

// Assert

Assert.IsTrue(labWork.GetStudents().Contains(student));

}

}

}

***Додаток З***Машинний лістинг класу **SchoolClassUnitTest****.cs**

using Project\_part\_A.Classes;

namespace A\_UnitTests

{

[TestClass]

public class SchoolClassTests

{

[TestMethod]

public void SchoolClass\_AddStudent\_AddsStudentToStudentsList()

{

// Arrange

var schoolClass = new SchoolClass(10, new Teacher("Mrs.", "Johnson"));

var student = new Student("John", "Doe", 15);

// Act

schoolClass.AddStudent(student);

// Assert

Assert.IsTrue(schoolClass.GetStudents().Contains(student));

}

[TestMethod]

public void SchoolClass\_GetClassTeacher\_ReturnsClassTeacher()

{

// Arrange

var teacher = new Teacher("Mr.", "Smith");

var schoolClass = new SchoolClass(11, teacher);

// Act

var classTeacher = schoolClass.GetClassTeacher();

// Assert

Assert.AreEqual(teacher, classTeacher);

}

}

}

***Додаток І***Машинний лістинг класу **SchoolSystemUnitTest.cs**

using Project\_part\_A.Classes;

using Project\_part\_A;

namespace A\_UnitTests

{

[TestClass]

public class SchoolSystemTests

{

[TestMethod]

public void SchoolSystem\_AddClass\_ShouldAddClassToList()

{

// Arrange

var schoolSystem = new SchoolSystem(new List<SchoolClass>(), new List<Teacher>(), new List<Student>());

var schoolClass = new SchoolClass(10, new Teacher("Mr.", "Johnson"));

// Act

schoolSystem.AddClass(schoolClass);

// Assert

CollectionAssert.Contains(schoolSystem.GetClasses(), schoolClass);

}

[TestMethod]

public void SchoolSystem\_AddTeacher\_ShouldAddTeacherToList()

{

// Arrange

var schoolSystem = new SchoolSystem(new List<SchoolClass>(), new List<Teacher>(), new List<Student>());

var teacher = new Teacher("Mrs.", "Smith");

// Act

schoolSystem.AddTeacher(teacher);

// Assert

CollectionAssert.Contains(schoolSystem.GetTeachers(), teacher);

}

[TestMethod]

public void SchoolSystem\_AddStudent\_ShouldAddStudentToList()

{

// Arrange

var schoolSystem = new SchoolSystem(new List<SchoolClass>(), new List<Teacher>(), new List<Student>());

var student = new Student("John", "Doe", 15);

// Act

schoolSystem.AddStudent(student);

// Assert

CollectionAssert.Contains(schoolSystem.GetStudents(), student);

}

}

}

***Додаток Й***Машинний лістинг класу **StudentUnitTest.cs**

using Project\_part\_A.Classes;

namespace A\_UnitTests

{

[TestClass]

public class StudentTests

{

[TestMethod]

public void Student\_ChooseSubject\_AddsSubjectToChosenSubjects()

{

// Arrange

var student = new Student("John", "Doe", 18);

var subject = new Subject("Math", new Teacher("Mr.", "Smith"));

// Act

student.ChooseSubject(subject);

// Assert

Assert.IsTrue(student.ChosenSubjects.Contains(subject));

}

[TestMethod]

public void Student\_GetEnrolledSubjects\_ReturnsChosenSubjects()

{

// Arrange

var student = new Student("Jane", "Doe", 17);

var mathSubject = new Subject("Math", new Teacher("Ms.", "Jones"));

var englishSubject = new Subject("English", new Teacher("Mrs.", "Brown"));

student.ChooseSubject(mathSubject);

student.ChooseSubject(englishSubject);

// Act

var enrolledSubjects = student.GetEnrolledSubjects();

// Assert

Assert.AreEqual(2, enrolledSubjects.Count);

CollectionAssert.Contains(enrolledSubjects, mathSubject);

CollectionAssert.Contains(enrolledSubjects, englishSubject);

}

}

}

***Додаток К***Машинний лістинг класу **SubjectUnitTest.cs**

using Project\_part\_A.Classes;

namespace A\_UnitTests

{

[TestClass]

public class SubjectTests

{

[TestMethod]

public void Subject\_GetName\_ReturnsSubjectName()

{

// Arrange

var teacher = new Teacher("Mr.", "Johnson");

var subject = new Subject("Math", teacher);

// Act

var subjectName = subject.GetName();

// Assert

Assert.AreEqual("Math", subjectName);

}

[TestMethod]

public void Subject\_GetTeacher\_ReturnsAssignedTeacher()

{

// Arrange

var teacher = new Teacher("Ms.", "Brown");

var subject = new Subject("English", teacher);

// Act

var assignedTeacher = subject.GetTeacher();

// Assert

Assert.AreEqual(teacher, assignedTeacher);

}

}

}

***Додаток Л***Машинний лістинг класу **TeacherUnitTest.cs**

using Project\_part\_A.Classes;

namespace A\_UnitTests

{

[TestClass]

public class TeacherTests

{

[TestMethod]

public void Teacher\_AddTaughtSubject\_AddsSubjectToTaughtSubjects()

{

// Arrange

var teacher = new Teacher("Mr.", "Johnson");

var subject = new Subject("History", teacher);

// Act

teacher.AddTaughtSubject(subject);

// Assert

Assert.IsTrue(teacher.TaughtSubjects.Contains(subject));

}

[TestMethod]

public void Teacher\_GetTaughtSubjects\_ReturnsTaughtSubjects()

{

// Arrange

var teacher = new Teacher("Mrs.", "Williams");

var mathSubject = new Subject("Math", teacher);

var physicsSubject = new Subject("Physics", teacher);

teacher.AddTaughtSubject(mathSubject);

teacher.AddTaughtSubject(physicsSubject);

// Act

var taughtSubjects = teacher.GetTaughtSubjects();

// Assert

Assert.AreEqual(2, taughtSubjects.Count);

CollectionAssert.Contains(taughtSubjects, mathSubject);

CollectionAssert.Contains(taughtSubjects, physicsSubject);

}

}

}