**КПІ ім. Ігоря Сікорського**

**Факультет інформатики та обчислювальної техніки**

**Кафедра інформатики та програмної інженерії**

**Звіт до комп’ютерного практикуму з курсу**

**“Основи програмування. Частина 2”**

Прийняв Виконав

Ст. викладач кафедри ІПІ Студент групи ІП-41

Вітковська І.І. Коломієць Максим

“14” травня 2025 р.

**Київ 2025**

**Комп’ютерний практикум №8**

***Тема:*** *Відношення між класами та об’єктами*

**Мета лабораторної роботи**– дослідити типи відношень між класами та об’єктами в ООП, навчитися проектувати об’єктно-орієнтовану модель предметної галузі.

**Завдання:**

1. Вивчити типи відношень між класами в ООП.

2. Спроектувати об’єктно-орієнтовану модель предметної галузі згідно з варіантом, визначивши необхідні для цього класи та їх структуру**.**

3.Вимоги до проектування:

-розробити  не менше 6 типів даних;

-застосувати всі базові принципи ООП;

-застосувати всі види відношень;

- застосувати обробку виключень, де це є необхідним;

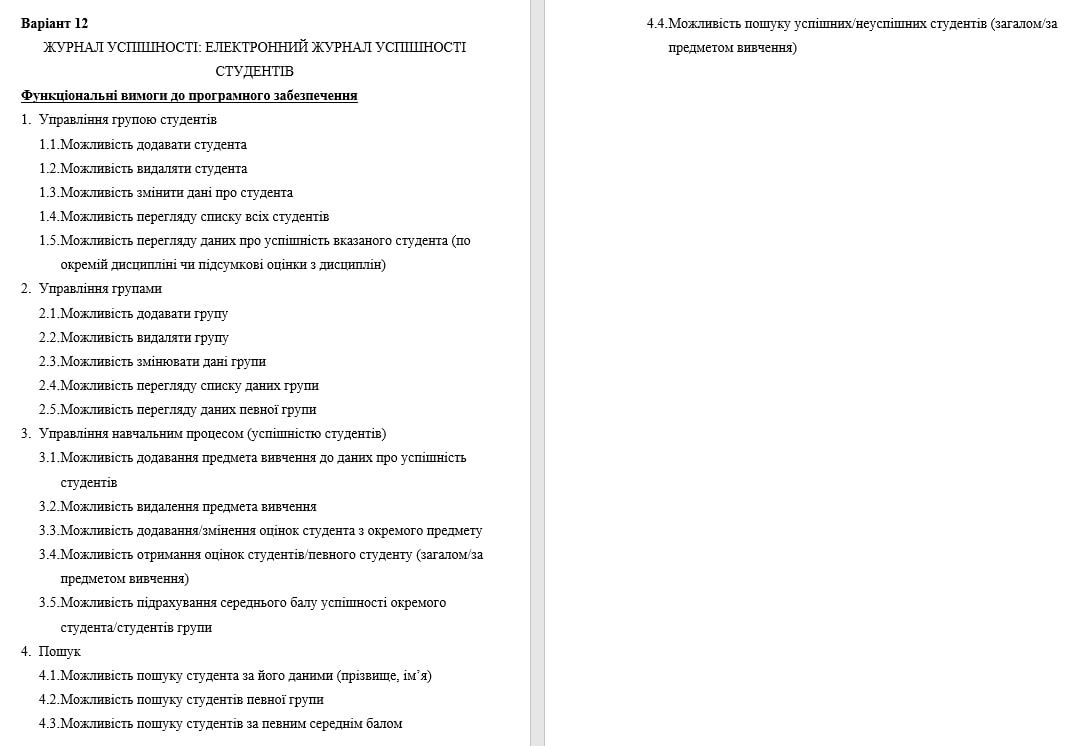
-дотримуватись єдиної конвенції найменувань та принципів написання "чистого" коду;

- код дозволяється коментувати лише xml-коментарями.

4. Написати програму, в якій реалізувати попередньо спроектовану об’єктно-орієнтовану модель.

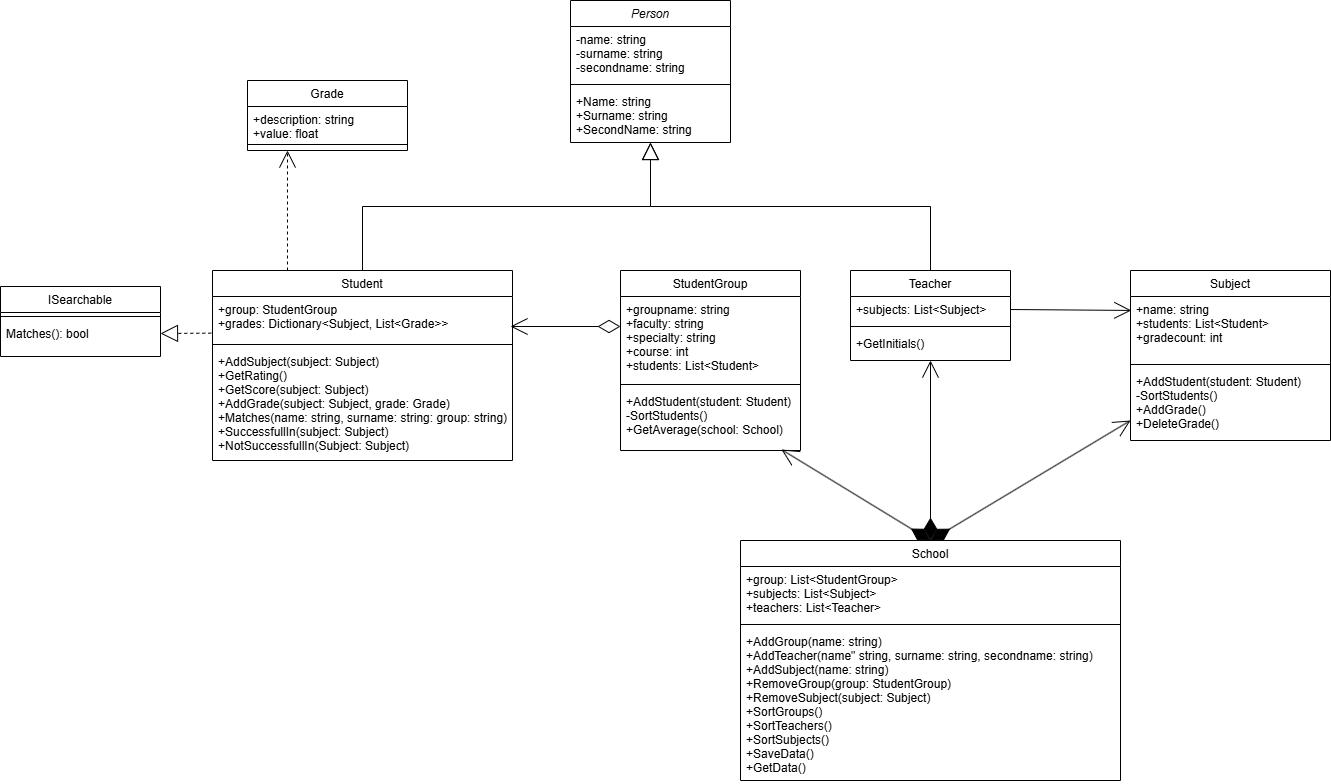
5. Програмний інтерфейс, наприклад, введення\виведення з консолі, реалізовувати окремим проектом. Код програмного інтерфейсу має бути простим (демонструється використання класів предметної галузі шляхом створення об’єктів та їх застосування, відсутня перевірка коректності вводу, введення з консолі мінімальне або відсутнє взагалі).

**Варіант 12:**



**Об’єктно-орієнтована модель**

**Діаграма класів:**

**  
Програмна реалізація**

public class Person

{

private string name;

private string secondname;

private string surname;

public string Name

{

get { return name; }

set { name = value; }

}

public string SecondName

{

get { return secondname; }

set { secondname = value; }

}

public string Surname

{

get { return surname; }

set { surname = value; }

}

}

public class Teacher : Person

{

public List<Subject> subjects = new List<Subject>();

public Teacher(string n, string sn, string s)

{

Name = n;

Surname = sn;

SecondName = s;

}

public string GetInitials()

{

return $" {this.Surname} {this.Name[0]}.{this.SecondName[0]}.";

}

}

internal interface ISearchable

{

bool Matches(string name, string surname, string group);

}

public class Student : Person, ISearchable

{

public StudentGroup group;

public Dictionary<Subject, List<Grade>> grades;

public Student(string n, string sn,string s, StudentGroup g)

{

Name = n;

Surname = sn;

SecondName = s;

group = g;

grades = new Dictionary<Subject, List<Grade>>();

}

public void AddSubject(Subject subject)

{

List<Grade> studentgrades = new List<Grade>();

this.grades.Add(subject, studentgrades);

}

public float GetRating()

{

float sum = 0;

int counter = 0;

foreach(var entry in grades)

{

Subject subject = entry.Key;

sum+= this.GetScore(subject);

counter++;

}

if (counter == 0)

{

return 0;

}

return sum / counter;

}

public float GetScore(Subject subject)

{

float sum = 0;

foreach(Grade g in grades[subject])

{

sum += g.value;

}

if (sum > 100)

{

return 100;

}

return sum;

}

public void AddGrade(Subject subject, Grade grade)

{

List<Grade> g = grades[subject];

g.Add(grade);

}

public bool Matches(string name, string surname, string group)

{

bool nameMatch = string.IsNullOrEmpty(name) || this.Name.ToLower().StartsWith(name.ToLower());

bool surnameMatch = string.IsNullOrEmpty(surname) || this.Surname.ToLower().StartsWith(surname.ToLower());

bool groupMatch = string.IsNullOrEmpty(group) || this.group.Groupname.ToLower().StartsWith(group.ToLower());

return nameMatch && surnameMatch && groupMatch;

}

public bool SuccessfullIn(Subject subject)

{

if(subject == null|| !grades.ContainsKey(subject))

{

return false;

}

return GetScore(subject) >=60;

}

public bool NotSuccessfullIn(Subject subject)

{

if (subject == null || !grades.ContainsKey(subject))

{

return false;

}

return GetScore(subject) < 60;

}

}

public class StudentGroup

{

private string groupname;

private string faculty = "ФІОТ";

private string specialty = "121 Інженерія програмного забезпечення";

private int course = 1;

public string Groupname

{

get { return groupname; }

set { groupname = value; }

}

public string Faculty

{

get { return faculty; }

set { faculty = value; }

}

public string Specialty

{

get { return specialty; }

set { specialty = value; }

}

public int Course

{

get { return course; }

set {

if (value >= 1 && value <= 4)

{

course = value;

}

else

{

value = 1;

}

}

}

public List<Student> students = new List<Student>();

public StudentGroup(string name)

{

groupname = name;

}

public void AddStudent(Student student)

{

students.Add(student);

this.SortStudents();

}

public void SortStudents()

{

students = students.OrderBy(s => s.Surname).ToList();

}

public float GetAverage(School school)

{

float sum = 0;

int count = 0;

foreach (Student student in students)

{

sum += student.GetRating();

count++;

}

if(count == 0)

{

return 0;

}

return sum / count;

}

}

public class Grade

{

public string description;

public float value;

public Grade(string description, float grade)

{

this.description = description;

if (grade < 0)

{

this.value = 0;

}

else if (grade > 100)

{

this.value = 100;

}

else

{

this.value = grade;

}

}

}

public class Subject

{

public string name;

public List<Student> students;

public int gradecount = 0;

public Subject(string name)

{

this.name = name;

students = new List<Student>();

}

public void AddStudent(Student student)

{

students.Add(student);

student.AddSubject(this);

for (int i = 0; i < this.gradecount; i++)

{

Grade grade = new Grade("-", 0);

student.grades[this].Add(grade);

}

SortStudents();

}

private void SortStudents()

{

students = students.OrderBy(s => s.Surname).OrderBy(s => s.group.Groupname).ToList();

}

public void AddGrade()

{

foreach (Student student in students)

{

Grade grade = new Grade("-", 0);

student.grades[this].Add(grade);

}

gradecount++;

}

public void DeleteGrade()

{

if (gradecount > 0)

{

foreach (Student student in students)

{

student.grades[this].RemoveAt(student.grades[this].Count - 1);

}

gradecount--;

}

}

}

public class School

{

public List<StudentGroup> groups = new List<StudentGroup>();

public List<Subject> subjects = new List<Subject>();

public List<Teacher> teachers = new List<Teacher>();

public StudentGroup AddGroup(string name)

{

StudentGroup group = new StudentGroup(name);

groups.Add(group);

this.SortGroups();

return group;

}

public Teacher AddTeacher(string name, string surname, string secondname)

{

Teacher teacher = new Teacher(name, surname, secondname);

teachers.Add(teacher);

this.SortTeachers();

return teacher;

}

public Subject AddSubject(string name)

{

Subject subject = new Subject(name);

subjects.Add(subject);

this.SortSubjects();

return subject;

}

public void RemoveGroup(StudentGroup group)

{

if (groups.Contains(group))

{

groups.Remove(group);

group = null;

}

}

public void RemoveSubject(Subject subject)

{

if (subjects.Contains(subject))

{

subjects.Remove(subject);

foreach (Teacher teacher in teachers)

{

if(teacher.subjects.Contains(subject))

{

teacher.subjects.Remove(subject);

}

}

subject = null;

}

}

public void SortGroups()

{

groups = groups.OrderBy(g => g.Groupname).ToList();

}

public void SortTeachers()

{

teachers = teachers.OrderBy(t => t.Surname).ToList();

}

public void SortSubjects()

{

subjects = subjects.OrderBy(s => s.name).ToList();

}

public void SaveData()

{

string pathGroups = "!groups.txt";

string pathStudents = "!students.txt";

string pathSubjects = "!subjects.txt";

string pathTeachers = "!teachers.txt";

using (StreamWriter writer = new StreamWriter(pathGroups))

{

foreach (StudentGroup group in this.groups)

{

writer.WriteLine(group.Groupname);

writer.WriteLine(group.Faculty);

writer.WriteLine(group.Specialty);

writer.WriteLine(group.Course);

}

}

using (StreamWriter writer = new StreamWriter(pathStudents))

{

foreach (StudentGroup group in this.groups)

{

foreach (Student student in group.students)

{

writer.WriteLine(student.Name);

writer.WriteLine(student.Surname);

writer.WriteLine(student.SecondName);

writer.WriteLine(student.group.Groupname);

}

}

}

using (StreamWriter writer = new StreamWriter(pathSubjects))

{

writer.WriteLine(this.subjects.Count);

foreach (Subject subject in this.subjects)

{

writer.WriteLine(subject.name);

writer.WriteLine(subject.students.Count);

writer.WriteLine(subject.gradecount);

foreach (Student student in subject.students)

{

writer.WriteLine(student.Surname);

writer.WriteLine(student.Name);

writer.WriteLine(student.SecondName);

writer.WriteLine(student.group.Groupname);

foreach (Grade grade in student.grades[subject])

{

writer.WriteLine(grade.description);

writer.WriteLine(grade.value);

}

}

}

}

using (StreamWriter writer = new StreamWriter(pathTeachers))

{

writer.WriteLine(this.teachers.Count);

foreach (Teacher teacher in this.teachers)

{

writer.WriteLine(teacher.Name);

writer.WriteLine(teacher.Surname);

writer.WriteLine(teacher.SecondName);

writer.WriteLine(teacher.subjects.Count);

foreach (Subject subject in teacher.subjects)

{

writer.WriteLine(subject.name);

}

}

}

}

public void GetData()

{

string pathGroups = "!groups.txt";

string pathStudents = "!students.txt";

string pathSubjects = "!subjects.txt";

string pathTeachers = "!teachers.txt";

using (StreamReader reader = new StreamReader(pathGroups))

{

string line;

while ((line = reader.ReadLine()) != null)

{

StudentGroup group = this.AddGroup(line);

group.Faculty = reader.ReadLine();

group.Specialty = reader.ReadLine();

group.Course = int.Parse(reader.ReadLine());

}

}

using (StreamReader reader = new StreamReader(pathStudents))

{

string name, surname, secondname, groupname;

while ((name = reader.ReadLine()) != null)

{

surname = reader.ReadLine();

secondname = reader.ReadLine();

groupname = reader.ReadLine();

StudentGroup group = this.groups.FirstOrDefault(g => g.Groupname == groupname);

Student student = new Student(name, surname, secondname, group);

group.AddStudent(student);

}

}

using (StreamReader reader = new StreamReader(pathSubjects))

{

int gradecount, studentcount;

string subjectname, studentname, studentsurname, studentsecondname, groupname, gradedescription;

float gradedata;

int subjectscount = int.Parse(reader.ReadLine());

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

{

subjectname = reader.ReadLine();

studentcount = int.Parse(reader.ReadLine());

gradecount = int.Parse(reader.ReadLine());

Subject subject = this.AddSubject(subjectname);

for (int j = 0; j < studentcount; j++)

{

studentsurname = reader.ReadLine();

studentname = reader.ReadLine();

studentsecondname = reader.ReadLine();

groupname = reader.ReadLine();

StudentGroup foundgroup = this.groups.FirstOrDefault(g =>

g.Groupname == groupname);

Student foundStudent = foundgroup.students.FirstOrDefault(s =>

s.Name == studentname &&

s.Surname == studentsurname && s.SecondName == studentsecondname);

subject.AddStudent(foundStudent);

for (int k = 0; k < gradecount; k++)

{

gradedescription = reader.ReadLine();

gradedata = float.Parse(reader.ReadLine());

Grade grade = new Grade(gradedescription, gradedata);

foundStudent.AddGrade(subject, grade);

}

}

subject.gradecount = gradecount;

}

}

using (StreamReader reader = new StreamReader(pathTeachers))

{

int teachercount, subjectcount;

string name, surname, secondname, subjectname;

teachercount = int.Parse(reader.ReadLine());

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

{

name = reader.ReadLine();

surname = reader.ReadLine();

secondname = reader.ReadLine();

subjectcount = int.Parse(reader.ReadLine());

Teacher teacher = this.AddTeacher(name,surname,secondname);

for( int j = 0; j < subjectcount; j++)

{

subjectname = reader.ReadLine();

Subject foundsubject = this.subjects.FirstOrDefault(s =>s.name == subjectname);

teacher.subjects.Add(foundsubject);

}

}

}

}

}