КРИВОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
Кафедра моделювання та програмного забезпечення

ЗАТВЕРДЖУЮ
Перший проректор
_____ Владислав ЧУБАРОВ
“_____”_____2025 р.

РОБОЧА ПРОГРАМА НАВЧАЛЬНОЇ ДИСЦИПЛІНИ

Розробка програм на платформі .NET
(шифр і назва навчальної дисципліни)

спеціальність 121 – «Інженерія програмного забезпечення»
(шифр і назва напряму підготовки)

факультет Інформаційних технологій
(назва інституту, факультету, відділення)

Форма навчання	Курс	Семестр	Всього годин за планом	Кількість національних кредитів	Всього аудиторних годин	Аудиторних годин, (у тому числі КЗ)		Самостійна робота (год.)	Контрольно-модульні роботи	Залік (сем.)	Екзамен (сем.)
						Лекції	Лабораторні				
Денна	2	3	120	4	48	16	32	72	-	*	
Денна скорочена	1	1	120	4	48	16	32	72	-	*	
Заочна	2	3	120	4	12	6	6	108	-	*	
Заочна скорочена	1	1	120	4	12	6	6	108	-	*	

Кривий Ріг – 2025 рік

Робочу програму навчальної дисципліни «Розробка програм на платформі .NET» для здобувачів першого (бакалаврського) рівня вищої освіти за освітньою програмою «Інженерія програмного забезпечення» розроблено згідно з ОПП галузі знань 12 «Інформаційні технології» зі спеціальності 121 «Інженерія програмного забезпечення».

Розробники: старший викладач кафедри МПЗ Рибальченко О.Г.
інженер-програміст
відділу розробки програмного забезпечення ІТ-продуктів ТОВ «Т18» Заіка Б. В.

Робоча програма затверджена на засіданні кафедри моделювання та програмного забезпечення

Протокол від “ 22 ” листопада 2024 року № 4

Завідувач кафедри МПЗ, доцент, к.п.н. _____ Андрій СТРЮК

Схвалено вченою радою факультету інформаційних технологій

Протокол від “ 26 ” грудня 2024 року № 5

Голова вченої ради _____ Іван МУЗИКА

Схвалено групою забезпечення ОПП

Протокол від “ 22 ” листопада 2024 року № 4

Гарант ОПП _____ Андрій СТРЮК

ЗМІСТ

1. ОПИС НАВЧАЛЬНОЇ ДИСЦИПЛІНИ

<table>
<tr><th rowspan="2">Найменування показників</th><th rowspan="2">Галузь знань, спеціальність, ступінь вищої освіти</th><th colspan="4">Характеристика навчальної дисципліни</th></tr>
<tr><th>денна форма навчання</th><th>заочна форма навчання</th><th>денна скорочена форма навчання</th><th>заочна скорочена форма навчання</th></tr>
<tr><td>Кількість кредитів – 4</td><td>Галузь знань
12 Інформаційні технології
(шифр і назва)</td><td colspan="4">Нормативна</td></tr>
<tr><td>Модулів – 1</td><td rowspan="4">Спеціальність
121 Інженерія програмного забезпечення
(код та найменування спеціальності)</td><td colspan="4">Рік підготовки:</td></tr>
<tr><td>Змістових модулів – 1</td><td>2</td><td>2</td><td>1</td><td>1</td></tr>
<tr><td rowspan="2">Загальна кількість годин - 120</td><td colspan="4">Семестр</td></tr>
<tr><td>3-й</td><td>3-й</td><td>1-й</td><td>1-й</td></tr>
<tr><td rowspan="10">Тижневих годин для денної форми навчання:
аудиторних – 3
самостійної роботи студента – 4,5</td><td rowspan="10">Ступінь вищої освіти: бакалавр</td><td colspan="4">Лекції</td></tr>
<tr><td>16 год.</td><td>6 год.</td><td>16 год.</td><td>6 год.</td></tr>
<tr><td colspan="4">Практичні, семінарські</td></tr>
<tr><td>-</td><td>-</td><td>-</td><td>-</td></tr>
<tr><td colspan="4">Лабораторні</td></tr>
<tr><td>32 год.</td><td>6 год.</td><td>32 год.</td><td>6 год.</td></tr>
<tr><td colspan="4">Самостійна робота</td></tr>
<tr><td>72 год.</td><td>108 год.</td><td>72 год.</td><td>108 год.</td></tr>
<tr><td colspan="4">Вид контролю</td></tr>
<tr><td colspan="2">Залік – 3 семестр</td><td colspan="2">Залік – 1 семестр</td></tr>
</table>

Примітка.

Співвідношення кількості годин аудиторних занять до самостійної і індивідуальної роботи становить:

для денної форми навчання – 0,67

для денної скороченої форми навчання – 0,67

для заочної форми навчання – 0,111

для заочної скороченої форми навчання – 0,111

2. МЕТА ТА ЗАВДАННЯ НАВЧАЛЬНОЇ ДИСЦИПЛІНИ

2.1. **Метою** викладання навчальної дисципліни «Розробка програм на платформі .NET» є формування у здобувачів теоретичного базису щодо сучасних підходів і методик розробки різного роду застосувань на платформі .NET, а також формування практичних навичок з оптимального використання .NET у подальшій фаховій практиці.

2.2. **Завданнями** вивчення дисципліни «Розробка програм на платформі .NET» є:

- опанування синтаксису і семантики мови C#, вивчення теорії типів і системи типізації в .NET;
- поглиблення знань з об'єктно-орієнтованого програмування та математичних основ ООП;
- застосування концепції .NET для реалізації ООП;
- ознайомлення з подієво-орієнтованим програмуванням та компонентним підходом до програмування;
- набуття вміння створювати додатки мовою C# на основі сучасних методів об'єктно-орієнтованого та компонентного програмування у інтегрованих середовищах програмування Visual Studio та Rider.

2.3. Відповідно до освітньої програми дисципліна забезпечує наступні **компетентності**:

Загальні компетентності

ЗК02 Здатність застосовувати знання у практичних ситуаціях.

Фахові компетентності

СК03 Здатність розробляти архітектури, модулі та компоненти програмних систем.

СК11 Здатність реалізовувати фази та ітерації життєвого циклу програмних систем та інформаційних технологій на основі відповідних моделей і підходів розробки програмного забезпечення.

СК13 Здатність обґрунтовано обирати та освоювати інструментарій з розробки та супроводження програмного забезпечення.

2.4. **Програмні результати навчання** освітньої програми, яким відповідає дисципліна:

ПР06 Уміння вибирати та використовувати відповідну задачі методологію створення програмного забезпечення.

ПР07 Знати і застосовувати на практиці фундаментальні концепції, парадигми і основні принципи функціонування мовних, інструментальних і обчислювальних засобів інженерії програмного забезпечення.

ПР15 Мотивовано обирати мови програмування та технології розробки для розв'язання завдань створення і супроводження програмного забезпечення.

ПР17 Вміти застосовувати методи компонентної розробки програмного забезпечення.

В результаті вивчення дисципліни здобувачі повинні **знати:**

- об'єктно-орієнтований підхід до програмування;
- основні поняття ООП;
- синтаксис і семантику мови C#;
- теорію типів і систему типізації в .NET;
- подієво-орієнтоване програмування;
- застосування концепції .NET для реалізації ООП;
- компонентний підхід до програмування;

вміти:

- використовувати основні синтаксичні та семантичні конструкції мови C#;
- обґрунтовувати необхідність використання об’єктно-орієнтованої технології або компонентного підходу для вирішення конкретної задачі;
- створювати консольні прикладні застосування мовою C# на основі сучасних методів об'єктно-орієнтованого та компонентного програмування;
- використовувати в професійній діяльності сучасні інтегровані середовища програмування (Visual Studio, Rider);
- самостійно опановувати нові методи та технології розробки програм.

2.5. Міждисциплінарні зв’язки

При вивченні дисципліни використовуються знання здобувачів з дисциплін «Основи програмування», «Вища математика», «Алгоритми та структури даних».

Знання, одержані здобувачами при вивченні дисципліни, використовуються при вивченні дисциплін «Об’єктно-орієнтоване програмування» з курсовою роботою, «Бази даних» з курсовою роботою.

Вимоги до знань та умінь визначаються галузевими стандартами вищої освіти України.

3. ПРОГРАМА НАВЧАЛЬНОЇ ДИСЦИПЛІНИ

Заліковий модуль 1. Розробка програм на платформі .NET

Тема 1. Вступ. Основні поняття мови C#

Що таке .NET, CLR та IL. Синтаксис, семантика, типізація у мові C#. Структури вводу, виводу, управління та повторення. Масиви та рядки у C#. Методи, параметри методів. Правила DRY та KISS.

Тема 2. Обробка виключних ситуацій

Конструкція try…catch. Фільтрація виключень. Операція throw. Блок finally. Використання конструкції using та інтерфейсу IDisposable для безпечної роботи з некерованими ресурсами.

Тема 3. Колекції у мові C#

Узагальнення та їх обмеження. Інтерфейс IEnumerable<T>, ключове слово yield. CRUD для об'єкта, що ітерується. Колекція IList<T>, List<T>, індексація по об'єкту, динамічний масив в контексті C#, основні методи. Множина ISet<T>, HashSet<T>, основні методи та властивості. Стек Stack<T>. Черга Queue<T>. Інтерфейс IDictionary<TKey, TValue>, KeyValuePair<TKey, TValue>. Словник Dictionary<TKey, TValue>, основні методи та особливості використання. Декоратор ReadOnlyCollection<T>.

Тема 4. Об'єктно-орієнтоване програмування у мові C#

Клас, структура, запис. Модифікатори доступу. Поля. Властивості. Конструктори. Ключове слово static. Абстрактні класи. Спадкування в C#. Інтерфейси. Спадкування конструкторів. Ключове слово base. Перетворення типів, поліморфізм подтипів. Віртуальні члени класу, ключові слова virtual, override, new. Інтерфейс vs абстрактний клас. Делегати Action<T>, Func<T>, лямбда-функція. Події, обробка подій, клас Timer.

Тема 5. Мова інтерпретованих запитів LINQ

Призначення. Проекція колекції в іншу колекцію. Фільтрація колекції. Пошук елементів за умовою: First, FirstOrDefault, Last, LastOrDefault, Single, SingleOrDefault, Any, All, Contains. Операції з властивостями або полями об'єктів колекції: Min, Max, Sum, Average. Методи Aggregate, Take, TakeWhile, Skip, SkipWhile. Сортування за умовою. Багаторівневе сортування. Методи Concat, Distinct, Union, Except, Intersect, Group By.

Тема 6. Потоки. Робота з файлами, мережею. HTTP-клієнт

Stream як основний клас для роботи з потоками. FileStream для передачі даних файлів. Декоратори для роботи з потоками StreamReader та StreamWriter. Робота з віддаленими серверами за допомогою HttpClient. Отримання результатів як потоків.

Тема 7. Принципи SOLID. Основні патерни проектування

Принцип єдиного обов'язку. Принцип відкритості/закритості. Принцип підстановки Лісков. Принцип розділення інтерфейсу. Принцип інверсії залежностей. Породжуючі патерни GoF. Поведінкові та структурні патерни.

Тема 8. Основи реляційних баз даних

Базові поняття про бази даних. Концепція реляційної бази даних. Сутності, первинні ключі, зв'язки між сутностями, зовнішній ключ, індекси в реляційних базах даних.

4. СТРУКТУРА НАВЧАЛЬНОЇ ДИСЦИПЛІНИ

Назви змістових модулів і тем	Кількість годин							
	Денна/денна скорочена форма				Заочна/заочна скорочена форма			
	усього	у тому числі			усього	у тому числі		
		лекції	лабораторні	самостійна робота		лекції	лабораторні	самостійна робота
1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
Заліковий модуль 1								
Тема 1. Вступ. Основні поняття мови C#	15	2	4	9	15	-	-	15
Тема 2. Обробка виключних ситуацій	8	1	2	5	8	-	-	8
Тема 3. Колекції у мові C#	15	2	4	9	15	2	-	13
Тема 4. Об’єктно-орієнтоване програмування у мові C#	34	4	10	20	34	2	2	30
Тема 5. Мова інтерпретованих запитів LINQ	15	2	4	9	15	1	2	12
Тема 6. Потоки. Робота з файлами, мережею. HTTP-клієнт	23	3	6	14	23	1	2	20
Тема 7. Принципи SOLID. Основні патерни проектування	7	1	2	4	7	-	-	7
Тема 8. Основи реляційних баз даних	3	1	-	2	3	-		3
Усього годин за модуль 1:	**120**	**16**	**32**	**73**	**120**	**6**	**6**	**108**

5. ТЕМИ ПРАКТИЧНИХ ЗАНЯТЬ

Не передбачено.

6. ТЕМИ ЛАБОРАТОРНИХ ЗАНЯТЬ

№ з/п	Назва теми	Кількість годин	
		Денна/денна скорочена ф. н.	Заочна/заочна скорочена ф. н.
1.	Лабораторна робота №1. Основи синтаксису мови C#. Робота з масивами. Робота з рядками.	4	
2.	Лабораторна робота №2. Тренування роботи з класами	2	
3.	Лабораторна робота №3. Класи: властивості, методи, спадкування, інтерфейси	4	1
4.	Лабораторна робота №4. Класи. Робота з архітектурою проекту, навички роботи з існуючим кодом	8	1
5.	Лабораторна робота №5. Методи розширення, LINQ	4	2
6.	Лабораторна робота №6. Потоки. Робота з файлами.	4	1
7.	Лабораторна робота №7. Робота з мережею, HTTP-клієнт.	6	1
РАЗОМ		**32**	**6**

7. САМОСТІЙНА РОБОТА

На самостійну роботу здобувачам денної форми навчання відведено 72 години, заочної - 108 годин.

Самостійна робота здобувачів при вивченні дисципліни «Розробка програм на платформі .NET» залучає такі складові:

- опрацювання лекційного матеріалу з кожної теми;
- опрацювання додаткової літератури по темі;
- вивчення окремих тем або питань, що передбачені для самостійного опрацювання;
- підготовка до виконання, а також до захисту лабораторних робіт;
- виконання індивідуального завдання, зокрема підготовка матеріалу до розробки та захисту проєкту;
- підготовка до проведення контрольних заходів.

Розподіл годин самостійної роботи

№	Назва теми	Кількість годин			
		денна	денна скороч.	заочна	заочна скороч.
1.	Огляд архітектурного рішення .NET, CLR	1	1	1	1
2.	Поняття про збірку та маніфест в .NET	1	1	1	1
3.	Простори імен в .NET	1	1	1	1
4.	Статичні класи	1	1	1	1
5.	Часткові типи і методи	1	1	1	1
6.	Робота з датами, особливості обробки часових проміжків	1	1	1	1
7.	Використання анонімних функцій. Поняття чистої функції	1	1	1	1
8.	Підходи до організації валідації: винятки або об’єкт-результат	1	1	1	1
9.	Проблема вибору необхідних патернів проектування	1	1	1	1
10.	Серіалізація. Способи зберігання даних у файлах. CSV, JSON, XML	1	1	1	1
11.	Опрацювання матеріалу лекцій, робота з рекомендованою літературою	8	8	24	24
12.	Виконання лабораторних робіт	32	32	40	40
13.	Виконання індивідуального завдання (проєкту), зокрема підготовка матеріалу до розробки та захисту проекту	18	18	30	30
14.	Підготовка до контрольних заходів	4	4	4	4
	Разом	**72**	**72**	**108**	**108**

Приклади тем індивідуальних завдань (проєктів) для самостійного опрацювання

1. Облік родинного бюджету
2. Облік успішності академічної групи
3. Менеджмент закладу харчування
4. Менеджмент паркінгу
5. Виробничий цех
6. Ветеринарна клініка
7. Магазин промислових товарів
8. Книжковий каталог
9. Страхова компанія
10. Поштове відділення

8. МЕТОДИ НАВЧАННЯ

Використовуються наступні методи навчання**:** лекції, лабораторні заняття, самостійна робота.

Навчальна лекція – це логічне, послідовне викладання змісту навчання, яке характеризується судженнями, висновками, підсумком. Вона охоплює основний теоретичний матеріал однієї або кількох тем навчальної дисципліни. Призначенням лекції є формування у здобувачів фундаментальних знань з дисципліни, а також визначає основний зміст і характер усіх інших навчальних занять та самостійної роботи здобувачів із цієї дисципліни.

Лабораторне заняття - форма організації навчання, яку проводять за завданням і під керівництвом НПП. Основні дидактичні цілі – експериментальне підтвердження вивчених теоретичних положень навчальної дисципліни та формування вмінь й навичок їх практичного застосування. Проведення лабораторного заняття ґрунтується на попередньо підготовлених наборах завдань різної складності для розв'язання на занятті. Лабораторне заняття проводиться у навчальних лабораторіях з використанням пристосованого до умов навчального процесу устаткування.

Самостійна робота здобувача є основним способом оволодіння навчальним матеріалом у час, вільний від обов'язкових аудиторних занять. Мета виконання самостійної роботи – поглиблення, узагальнення й закріплення теоретичних знань і практичних умінь здобувачів із дисципліни шляхом вироблення вміння самостійної роботи з навчальною і фаховою літературою та інформацією в мережі Інтернет.

Самостійна робота здобувачів здійснюється у формі: підготовки до лекцій і лабораторних занять, виконанні самостійних проектів. Самостійну роботу здобувач може виконувати у бібліотеці, комп'ютерних класах, а також у домашніх умовах.

Підготовка до лекцій передбачає самостійне опрацювання теоретичного матеріалу. При цьому необхідно звернути увагу на необхідність чіткого засвоєння основних термінів та визначень, розуміння їх змісту, обов'язкового аналізу використання теоретичних положень для розв'язання наданих прикладів.

Самоперевірку засвоєння навчального матеріалу здобувач здійснює за контрольними запитаннями, що надано після кожної теми у конспекті лекцій та іншій літературі, та після кожного лабораторного заняття у відповідних методичних вказівках. Якщо на деякі запитання здобувач не може надати відповіді, то необхідно повторити вивчення навчального матеріалу, або визначити правильну відповідь з викладачем на консультації.

Під час вивчення даної дисципліни використовуються:

− мультимедійні освітні технології: інтерактивні лекції (презентації) із використанням програми MS Power Point у поєднанні з анімацією та звуковим супроводом; перегляд відеороликів за окремими пунктами тем занять, використання електронних посібників;

− діалогові технології: організація групових обговорень, використання «мозкового штурму».

Лекції проводяться з використанням технічних засобів навчання й супроводжуються демонстрацією презентацій за допомогою проектора.

У разі виникнення необхідності забезпечення навчального процесу в дистанційному режимі супровід та контроль знань реалізовується за допомогою дистанційного курсу, розробленого в Google ClassRoom. Онлайн лекції, консультації та усні відповіді на питання, захист проєктів проводиться за допомогою Google Meet або Zoom.

9. МЕТОДИ КОНТРОЛЮ

Основними завданнями контролю знань здобувачів вищої освіти з дисципліни є оцінювання засвоєння теоретичних знань і практичних навичок, отриманих під час навчання.

Контрольні заходи мають виконувати наступні функції:

- стимулювати систематичну самостійну роботу над навчальним матеріалом;
- забезпечувати закріплення та реалізацію набутих теоретичних знань при підготовці до практичних занять;
- прищеплювати навички відповідального ставлення до своїх обов'язків, самостійного цілеспрямованого пошуку потрібної інформації, чіткої організації свого робочого дня.

Оцінювання знань здобувачів складається з поточного та підсумкового контролю.

Поточний контроль знань здобувачів вищої освіти передбачає оцінювання за наступними основними напрямами:

- перевірка теоретичних знань;
- перевірка підготовки до лабораторних занять;
- перевірка виконання індивідуального завдання (проєкту).

З даних компонентів складаються загальні бали, які фіксуються в журналі викладача.

Оцінювання рівня засвоєння теоретичних знань здобувачів вищої освіти проводиться під час усної співбесіди зі здобувачами по теоретичним матеріалам, за результатами захисту проєкту й виконання самостійних робіт. Підсумковим контролем є залік.

10. РОЗПОДІЛ БАЛІВ, ЯКІ ОТРИМУЮТЬ ЗДОБУВАЧІ

Використовується модульно-рейтингова система оцінювання, яка передбачає розподіл балів за виконання всіх запланованих видів робіт. При цьому максимальна кількість балів за модуль при умові його бездоганного виконання дорівнює 100. Ця сума складається з балів, що накопичив студент у ході поточного контролю, та балів за виконання індивідуального завдання (проєкту).

Успішність студентів-заочників оцінюється аналогічно.

Лабораторні роботи у модулі відображують оволодіння навичками та вміння застосовувати знання на практиці і сукупно відповідають 70-ти відсоткам ваги. При зниженні якості виконання тієї чи іншої лабораторної роботи, знижується і кількість балів, якою вона оцінюється.

Розподіл балів за видами робіт

№ модуля	№ зан.	Вид роботи	Тема	Максимальна кількість балів	
				ден./ден. скорочена	заоч./заоч. скорочена
1	1	Лабораторна робота № 1	Основи синтаксису мови C#. Робота з масивами. Робота з рядками.	10	10
	2	Лабораторна робота № 2	Тренування роботи з класами	5	5
	3	Лабораторна робота № 3	Класи: властивості, методи, спадкування, інтерфейси	10	10
	4	Лабораторна робота № 4	Класи. Робота з архітектурою проекту, навички роботи з існуючим кодом	15	15
	5	Лабораторна робота № 5	Методи розширення, LINQ	10	10
	6	Лабораторна робота № 6	Потоки. Робота з файлами.	10	10
	7	Лабораторна робота №7	Робота з мережею, HTTP-клієнт.	10	10
		Самостійна робота	Індивідуальний проєкт за темою	30	30
			Разом за модуль	**100**	**100**

Оцінювання кожної лабораторної роботи ведеться за наступними показниками:

1. Своєчасність практичного виконання лабораторної роботи (у тиждень згідно із графіком робіт) (0-1 бали).
2. Теоретичний захист виконаної лабораторної роботи (у тиждень наступний за тижнем планового виконання роботи) (0-3 бали).
3. Якість знайдених студентом рішень (ефективність алгоритму, доречність використання елементів інтерфейсу, тощо) (6-11 балів).

Якість знайдених студентом рішень (ефективність алгоритму, доречність використання структур даних, елементів інтерфейсу тощо) оцінюється наступним чином:

- робота виконана без зауважень - максимальний бал;

- робота виконана достатньо повно з деякими зауваженнями –75% від максимального бала;
- робота виконана не повністю – 50% від максимального бала.

Оцінка всієї лабораторної роботи знаходиться підсумовуванням балів за кожний з показників.

Оцінювання індивідуального проєкту відбувається за наступними складовими частинами (наведено максимально можливий бал):

- використання інтерфейсів (1 бал);
- використання делегатів/подій (2 бали);
- використання узагальнень (2 бали);
- використання конструкторів (2 бали);
- відповідність коду конвенціям про стилізацію коду C# (4 бали);
- відповідність коду SOLID, KISS, DRY (4 бали);
- введення-виведення даних в програму оформлено у вигляді окремих модулів (2 бали);
- ваявність валідації даних (1 бал);
- загальний архітектурний підхід (3 бали);
- відповідність обсягів функціоналу розробленої програми вимогам до програм такого типу (9 балів).

Загальна оцінка індивідуального проєкту знаходиться підсумовуванням балів за кожну складову частину.

Для допуску до підсумкового контролю студент повинен виконати графік навчального процесу, усі види запланованих завдань і протягом семестру отримати в сумі не менше 50 балів.

Семестровий контроль здійснюється у формі заліку в третьому семестрі для денної та заочної форм навчання, у першому семестрі для денної скороченої форми навчання.

У разі виконання студентом усіх видів поточних контрольних заходів залік виставляється студенту на підставі зарахованих балів протягом семестру. Результати заліку оцінюються за 100-бальною шкалою. У відомість оцінка проставляється як у балах національної шкали, так і за шкалою ECTS:

При наявності у здобувачів **результатів неформального навчання** за освітнім компонентом «Розробка програм на платформі .NET» у повному обсязі, визнання та оцінювання результатів здійснюється відповідно до «Положення про порядок визнання у Криворізькому національному університеті результатів навчання, отриманих в умовах неформальної освіти». У випадку, якщо за підсумками визнання результатів неформального навчання визнається тільки частина результатів навчання, заявнику зараховуються окремі види навчальної роботи за освітнім компонентом «Розробка програм на платформі .NET».

Нижче наведені окремі види навчальної роботи, які можуть бути зараховані здобувачеві при наявності сертифікату про успішне проходження рекомендованих онлайн курсів.

Тема	Посилання на рекомендовані курси
Вступ. Основні поняття мови C#	https://training.epam.ua/Training/Details/3508?lang=ua https://www.coursera.org/learn/introduction-programming-unity https://foxminded.ua/cs-start-1/
Обробка виключних ситуацій	https://training.epam.ua/Training/Details/3508?lang=ua https://www.coursera.org/learn/introduction-programming-unity https://foxminded.ua/cs-start-1/
Колекції у мові C#	https://training.epam.ua/Training/Details/3508?lang=ua https://www.coursera.org/learn/introduction-programming-unity https://www.coursera.org/learn/data-structures-design-patterns https://foxminded.ua/cs-start-1/
Принципи SOLID. Основні патерни проектування	https://training.epam.ua/Training/Details/3508?lang=ua https://www.coursera.org/learn/data-structures-design-patterns https://foxminded.ua/cs-start-1/
Об’єктно-орієнтоване програмування у мові C#	https://training.epam.ua/Training/Details/3508?lang=ua https://www.coursera.org/learn/csharp-class-development https://www.coursera.org/learn/oo-development-using-c-sharp https://foxminded.ua/cs-start-1/
Мова інтерпретованих запитів LINQ	https://training.epam.ua/Training/Details/3508?lang=ua https://www.coursera.org/learn/oo-development-using-c-sharp https://foxminded.ua/cs-start-1/

Шкала оцінювання

Національна шкала успішності	**Оцінка ECTS**	**Визначення ECTS**	**100-бальна система оцінювання**
відмінно/ зараховано	**A**	ВІДМІННО - відмінне виконання лише з незначними помилками	90...100
добре/ зараховано	**B**	ДУЖЕ ДОБРЕ - вище середнього рівня з кількома помилками	80...89
	C	ДОБРЕ - у цілому правильно робота з певною кількістю помилок і недоліків	71...79
задовільно/ зараховано	**D**	ЗАДОВІЛЬНО - непогано, але зі значною кількістю грубих помилок	61...70
	E	ДОСТАТНЬО - виконання задовольняє мінімальні потреби	50...60
незадовільно/ не зараховано	**FX**	НЕЗАДОВІЛЬНО - із можливістю повторного складання	30...49
	F	НЕЗАДОВІЛЬНО - з обов'язковим повторним вивчення дисципліни	0...29

11. ПЕРЕЛІК ПИТАНЬ ДЛЯ ПІДСУМКОВОГО КОНТРОЛЮ ЗНАНЬ

1. Переваги платформи.NET над аналогами.
2. Призначення та особливості віртуальної машини.
3. Призначення основних просторів імен бібліотеки класів платформи.NET.
4. Способи використання директиви using.
5. Форматування даних при їх виведенні.
6. Використання засобів SDK для розроблення програм.
7. Призначення та використання перечислень.
8. Призначення та використання структур.
9. Використання списків аргументів змінної довжині.
10. Розробка рекурсивних методів.
11. Призначення та елементи класу Array.
12. Чим відрізняються value від reference type?
13. Що таке Generics? Які проблеми вони вирішують?
14. Що робить оператор yield?
15. Призначення та використання елементів класу System.Exception.
16. Призначення ключових слів try, catch, finaly.
17. Використання вкладених блоків try.
18. Розробка власних класів виключень.
19. Генерування користувальницьких виключень.
20. Перехоплення користувальницьких виключень.
21. Принципи об’єктно-орієнтованого підходу.
22. Синтаксис опису класу.
23. Особливості статичних елементів класу.
24. Специфікатори доступу до елементів класу у C#.
25. Порядок ініціалізації об'єкта класу.
26. Призначення та використання посилання this.
27. Агрегація й спадкування.
28. Синтаксис спадкування у C#.
29. Порядок виклику конструкторів при спадкуванні.
30. Чи дозволено множинне спадкування в C#?
31. У чому різниця між ключовими словами new та override при перевизначенні методу?
32. Принцип поліморфізму.
33. Переваги концепції поліморфізму.
34. Поняття про абстрактні класи та їх призначення.
35. Правила використання абстрактних класів.
36. Поняття про інтерфейси та їх призначення.
37. Правила використання інтерфейсів.
38. Що таке делегат?
39. Чи відрізняється Delegate від Action?
40. Що таке LINQ і для чого використовується? Наведіть кілька прикладів застосування LINQ.

12. ПРИКЛАДИ ТЕМ ІНДИВІДУАЛЬНИХ ПРОЕКТІВ

1. Облік родинного бюджету
2. Облік успішності академічної групи
3. Менеджмент закладу харчування
4. Менеджмент паркінгу
5. Виробничий цех
6. Ветеринарна клініка
7. Магазин промислових товарів
8. Книжковий каталог
9. Страхова компанія
10. Поштове відділення

13. НАВЧАЛЬНО-МЕТОДИЧНІ МАТЕРІАЛИ З ДИСЦИПЛІНИ

13.1 Навчальна та довідкова література

1. Платформа .NET та мова програмування C# 8.0: навч. посіб. / Коноваленко І.В., Марущак П.О. Тернопіль: ФОП Паляниця В.А., 2020. 320 с.

2. Ровінський В.А. Програмування мовою C#: навч. посіб. Івано-Франківськ: Сімик, 2016. 603 с.

3. Об’єктно-орієнтоване програмування. Лабораторний практикум: навч. посіб. / Бойко Б.І., Омельчук Л.Л., Русіна Н.Г. Київ: Київський національний університет ім. Тараса Шевченка, 2016. 90 с.

4. Бублик В.В. Об’єктно-орієнтоване програмування: підруч. Київ: ІТ-книга, 2015. 624 с.

5. Troelsen A., Japikse Ph. Pro C# 10 with .NET. Foundational Principles and Practices in Programming. Apress Berkeley, 2019. 1328 p.

6. Albahari J, Albahari B. C# 7.0 in a Nutshell: The Definitive Reference. O'Reilly Media, 2017. 1024 p.

7. Skeet J. C# in Depth. 3rd Edition. Manning Publications Co, 2016. 608 p.

8. Albahari J. C# 8.0. Pocket Reference. O'Reilly Media, 2019. 240 p.

9. Price M. J. C# 7 and .NET Core: Modern Cross-Platform Development. Packt Publishing, 2017. 640 p.

10. Larman C. Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and Iterative Development. Pearson, 2017. 736 p.

11. Martin R. C. Clean Code: A Handbook of Agile Software Craftsmanship. 2 Edition. Pearson, 2015. 478 p.

13.2. Методична література

1. Рибальченко О.Г., Білашенко С.В. Методичні вказівки до виконання лабораторних робіт з дисципліни «Розробка програм на платформі .NET» для студ. всіх форм навч. за спец. 121 «Інженерія програмного забезпечення». Кривий Ріг, 2023. 36 с.

14. ІНФОРМАЦІЙНІ РЕСУРСИ

До складу інформаційних ресурсів навчальної дисципліни входять:

1. Бібліотека Криворізького національного університету. URL: http://lib.knu.edu.ua/ (дата звернення 20.12.2024).

Internet-ресурси:

1. Рибальченко О.Г. Розробка програм на платформі .NET: консп. лекцій. URL: https://classroom.google.com/u/0/c/Mzg4MTQzMTI2MzMy?hl (дата звернення 20.12.2024).
2. MilanJovanovicTech. https://youtube.com/@MilanJovanovicTech (дата звернення 20.12.2024).
3. Raw Coding. https://youtube.com/@RawCoding (дата звернення 20.12.2024).
4. Інструкція з C# - Частина 1. https://professorweb.com/my/csharp/charp_theory/level1/index.php/ (дата звернення 20.12.2024).
5. Інструкція з C# - Частина 2. https://professorweb.com/my/csharp/charp_theory/level1/index1.php/ (дата звернення 20.12.2024).
6. LINQ to Objects https://professorweb.com/my/LINQ/base/level1/linq_index.php/ (дата звернення 20.12.2024).
7. Повна інструкція з мови програмування C# 10 та платформі .NET 6 https://metanit.com/sharp/tutorial/ (дата звернення 20.12.2024).

15. ТЕРМІНОЛОГІЧНИЙ СЛОВНИК

Об'єктно-орієнтоване програмування (Object-Oriented Programming) - A programming paradigm that emphasizes the use of objects, which encapsulate data and behavior, and provides a way to model complex systems and processes in a modular and reusable way, using principles such as abstraction, encapsulation, inheritance, and polymorphism.

Об'єкт (Object) - An instance of a class that encapsulates data and behavior, and represents a real-world entity or process. Stores state of entity

Клас (Class) - A blueprint for creating objects that defines a set of data and behavior shared by all instances of the class.

Інкапсуляція (Encapsulation) – Encapsulation is a fundamental concept in object-oriented programming (OOP) that involves bundling data and the methods (or functions) that operate on that data into a single unit, called a class. The purpose of encapsulation is to ensure that the object's state remains consistent and that changes to it are made in a controlled and predictable way. By allowing access to the object's internal data only through specific methods, encapsulation enables the object to enforce constraints on the data, such as ensuring that values are within a certain range or that relationships between different data fields are maintained. One of instruments to get good encapsulation is hiding details of class by using access modifiers.

Успадкування (Inheritance) - Inheritance is a mechanism that allows a new class, called the derived class or subclass, to be based on an existing class, called the base class or superclass. The derived class inherits the properties and behavior of the base class, which it can then modify, extend, or override as needed. This enables code reuse and allows for a more organized and modular code structure, as common properties and behavior can be defined in the base class and reused in the derived classes.

Поліморфізм (Polymorphism) - Polymorphism refers specifically to the ability of an object or function to behave in different ways based on the context in which it is used. This can include subtype polymorphism, as in OOP, but can also refer to other forms of polymorphism such as parametric polymorphism, where a function or data type can operate on values of different types, or ad-hoc polymorphism, where a function can behave differently based on the types of its arguments.

Абстракція (Abstraction) - Abstraction is a fundamental concept in computer science and programming that involves focusing on essential features or properties of an object or system, while ignoring details that are not relevant to its function or behavior. In programming, abstraction can refer to the process of defining and using abstract classes or interfaces that specify the essential features or behaviors of a group of related objects, without specifying their implementation details. Abstraction is important for creating reusable, modular, and maintainable code, as it allows programmers to work with higher-level concepts and hide implementation details that may change over time.

Модифікатори доступу (Access modifiers) - Access modifiers are keywords in object-oriented programming that determine the level of accessibility of classes, fields, methods, properties, and other members of a class or object. Access modifiers control which parts of a program can access or modify a given member, and provide a mechanism for encapsulating and hiding implementation details. In C#, access modifiers include public, private, protected, and internal, as well as more specialized modifiers such as protected internal or private protected. The choice of access modifier depends on the intended use and visibility of a given member, and is an important consideration in designing and implementing classes and objects in an OOP system.

Поле (Field) - Field is a variable that is defined within a class or struct and represents the state or data of an object. Fields can be accessed and modified by the methods or properties of the class, and can be declared with access modifiers such as public, private, or protected, to control their visibility

and accessibility, but having open fields is bad way to write your code. Fields are often used to encapsulate data within an object and provide a way to store and retrieve data in a structured manner, as part of the encapsulation and information hiding principles of OOP.

Властивість (Property) - Property is a member of a class or struct that provides access to a private field or other underlying data source, and allows for reading and/or writing of the value through getter and/or setter methods. Properties are often used as an abstraction layer to provide controlled access to an object's state or data, and can enforce validation, calculation, or other behaviors when the value is accessed or modified. Properties are declared with a get and/or set accessor, which define the logic for getting or setting the value, and can be declared with access modifiers such as public, private, or protected, to control their visibility and accessibility.

Метод (Method) - Method is a collection of statements or instructions that perform a specific task or action. Methods are associated with a class or object and can be called to perform a specific action on that class or object. Methods can take input parameters and return output values. Methods are often used to encapsulate reusable code and provide a way to perform a specific operation on an object or system, as part of the abstraction and modularity principles of OOP.

Конструктор (Constructor) - constructor is a special method that is called when an object of a class is created or instantiated. The constructor is used to initialize the object's state or data, and can take parameters or use default values. Constructors are typically used to set the initial values of an object's fields or properties, and can perform initialization tasks such as memory allocation or resource acquisition. Constructors are declared with the same name as the class, and can have different signatures to support different parameter lists or overloading. The use of constructors is a fundamental aspect of object-oriented programming and supports the encapsulation and abstraction principles.

Абстрактний клас (abstract class) - abstract class is a class that cannot be instantiated directly, but serves as a template or blueprint for other classes to derive from. An abstract class may contain one or more abstract methods, which are declared but not implemented in the abstract class itself, but must be implemented in derived classes. Abstract classes are used to define common behavior or characteristics that can be shared by multiple subclasses, while allowing for variation and customization in the implementation of specific methods or properties. Abstract classes are often used to implement the abstraction and inheritance principles of OOP, and base behavior, that must be extended by subclasses.

Інтерфейс (Interface) - an interface is a collection of abstract methods and properties that define a contract or set of behaviors that a class or object must implement. An interface defines a set of rules or guidelines for implementing a specific feature or behavior, but does not contain any implementation details itself. Interfaces are often used to enable polymorphism and provide a way for objects of different types to interact with each other in a consistent way. In C#, interfaces are declared using the interface keyword and may contain methods, properties, indexers, and events. Implementing an interface requires a class or struct to provide concrete implementations for all the methods and properties defined in the interface.

Виключення (Exception) - an exception is an object that represents an error or unexpected condition that occurs during the execution of a program. Exceptions can be thrown by a method or statement when an error occurs, and are used to signal that an operation has failed or could not be completed. Exceptions can contain information about the type and location of the error, as well as any relevant data or context. Exception handling is the process of detecting, responding to, and recovering from exceptions, and is an important aspect of robust and reliable software design. In C#, exceptions are typically handled using try-catch blocks or other structured error handling mechanisms.

Колекція (Collection) - a collection is an object that groups together other objects or values into a single unit, typically for the purpose of storing, organizing, or processing them in a convenient

way. Collections can be used to represent a wide range of data structures, from simple arrays and lists to more complex structures like sets, maps, and trees. Collections can be generic or non-generic, depending on whether they are strongly typed or not, and can provide a variety of methods and properties for manipulating and querying the contents of the collection. Collections are an essential part of many programming tasks, from simple data processing to complex algorithms and data analysis. In C#, collections are often implemented using the ICollection or IEnumerable interfaces (but not only them), and may be provided by the standard library or custom data structures.

Узагальнення (Generic) - generic refers to a feature that allows classes, methods, and other constructs to be parameterized by one or more types, rather than being limited to a fixed set of types. Generics provide a way to write reusable and type-safe code that can work with a wide variety of data types, without having to create separate implementations for each type. By defining classes or methods that can accept any type of data, generics enable greater flexibility and code reuse, while reducing the potential for errors and inefficiencies. Generics are commonly used for collections, algorithms, and other data structures that need to be generic and type-safe. In C#, generics are implemented using type parameters and are declared using angle brackets (< >). Also, generics can filter types in it by using word “where”.

Делегат (Delegate) - a delegate is a type that represents a reference to a method with a specific signature or set of parameters. Delegates can be used to create and pass around references to methods as if they were objects, allowing methods to be passed as arguments, stored as variables, and invoked dynamically at runtime. Delegates provide a way to encapsulate and pass behavior as data, which is a key aspect of many programming patterns and techniques. Delegates are commonly used for event handling, asynchronous programming, and functional programming, among other things. In C#, delegates are declared using the delegate keyword and can be instantiated using a lambda expression or a method reference. Also, there are 2 pre-defined generic delegates: Func<T>, Action<T>, which can cover 99% of delegates use cases.

Подія (Event) - an event is a mechanism that allows objects to notify other objects or components about changes or actions that occur within them. Events are commonly used for communication and coordination between objects, and can be used to trigger specific behaviors or reactions in response to changes in the system or user interactions. Events are typically implemented using a publisher-subscriber model, where one object (the publisher) raises an event, and one or more other objects (the subscribers) register to receive and handle the event. Events can be used to implement a wide range of behaviors, from simple notifications to complex workflows and state machines. In C#, events are declared using the event keyword and are typically implemented using delegates or event handlers.

LINQ (Language Integrated Query) is a feature in .NET Framework that provides a uniform way of querying data from different data sources using a consistent syntax and programming model. LINQ enables developers to write expressive and powerful queries that can work with various data types, including objects, databases, XML, and more. With LINQ, developers can use a single query syntax to query and manipulate data from different sources, without having to learn multiple query languages or APIs. LINQ supports a wide range of operations, including filtering, sorting, grouping, joining, and aggregation, and provides a flexible and extensible framework for building complex data-driven applications. In C# and other .NET languages, LINQ is implemented using query expressions, lambda expressions, and extension methods.

Типи значень (Value types) - In C#, a value type is a data type that directly stores its value in memory, rather than storing a reference to an object that contains the value. Examples of value types in C# include integral types (such as int, long, and byte), floating-point types (such as float and double), and the bool, char, and decimal types. Value types are typically smaller and more efficient than

reference types, as they do not require memory allocation on the heap and do not need to be garbage collected. In C#, value types are passed by value, meaning that a copy of the value is passed to a method or assigned to a variable, rather than a reference to the original value. However, value types can also be boxed, which means that they can be wrapped in a reference type and treated as objects.

Типи посилань (reference type) - In C#, a reference type is a data type that stores a reference to an object in memory, rather than storing the object directly. Examples of reference types in C# include class types, interface types, delegate types, and array types. Reference types are allocated on the heap and are garbage collected by the CLR (Common Language Runtime), which means that they can be dynamically allocated and deallocated at runtime. In C#, reference types are passed by reference, meaning that a reference to the original object is passed to a method or assigned to a variable, rather than a copy of the object's value. This allows multiple variables or methods to access and modify the same object. However, this also means that reference types can lead to issues such as memory leaks and unintended modifications if not used carefully.

Нульові значення (null) - In programming, null is a special value that represents the absence of a value or the lack of an object reference. It is typically used to indicate that a variable or object does not currently hold a valid value or reference. In C#, null is the default value for reference types, meaning that a variable that has not been initialized will hold a null reference. Attempting to access a member or method of a null reference will result in a NullReferenceException at runtime. In contrast, value types cannot hold null as a value, but can be made nullable using the ? operator or the Nullable<T> structure. Handling null values is an important consideration when writing robust and reliable code.

Принцип єдиної відповідальності (The Single Responsibility Principle) - The Single Responsibility Principle (SRP) is a design principle in object-oriented programming that states that a class should have only one reason to change. In other words, a class should have only one responsibility or job, and it should not be responsible for multiple unrelated tasks. This principle helps to improve code readability, maintainability, and reusability, as it allows classes to be focused and specialized, rather than having a mix of unrelated functions. By separating concerns and responsibilities into different classes, changes can be made more easily and with less risk of introducing bugs or unintended consequences.

Принцип відкритості-закритості (The Open-Closed Principle) - The Open-Closed Principle (OCP) is a design principle in object-oriented programming that states that software entities (such as classes, modules, and functions) should be open for extension, but closed for modification. This means that the behavior of an entity should be modifiable without changing its source code, by adding new code or modules that extend its functionality. This approach promotes software reuse and minimizes the impact of changes to the system, by reducing the need for modification and retesting of existing code. The OCP can be achieved through techniques such as inheritance, composition, and dependency injection.

Принцип підстановки Ліскова (The Liskov Substitution Principle) - The Liskov Substitution Principle (LSP) is a design principle in object-oriented programming that states that objects of a superclass should be replaceable with objects of its subclasses, without affecting the correctness of the program. In other words, any derived class should be able to be used in place of its parent class without causing errors or unexpected behavior. This principle helps to ensure that code is modular and extensible, and that behavior is consistent and predictable across different implementations. Violating the LSP can lead to code that is hard to understand, maintain, and test, and can cause unexpected errors or bugs.

Принцип сегрегації інтерфейсів (The Interface Segregation Principle) - The Interface Segregation Principle (ISP) is a design principle in object-oriented programming that states that a

client should not be forced to depend on methods it does not use. In other words, interfaces should be designed to be as small and specific as possible, and clients should only depend on the methods that they actually need to use. This principle helps to improve code modularity, maintainability, and flexibility, by reducing the coupling between different components of a system. By separating interfaces into smaller, more focused units, it becomes easier to reuse and replace components, and to write tests that cover specific behavior.

Принцип інверсії залежностей (The Dependency Inversion Principle) - The Dependency Inversion Principle (DIP) is a design principle in object-oriented programming that states that high-level modules should not depend on low-level modules, but both should depend on abstractions. Additionally, abstractions should not depend on details, but details should depend on abstractions. This principle helps to promote loose coupling between components of a system, which makes the system more modular, flexible, and maintainable. By relying on abstractions rather than concrete implementations, it becomes easier to change components without affecting other parts of the system. DIP can be implemented using techniques such as dependency injection, inversion of control, and interface-based programming.

Додаток до робочої програми

Робочий план

з дисципліни **«**Розробка програм на платформі .NET**»**
для здобувачів вищої освіти першого (бакалаврського) рівня вищої освіти
за спеціальністю 121 «Інженерія програмного забезпечення»

Вид навчальної роботи	Годин у семестрі/кредитів	Розподіл годин по тижнях																Вид підсумкового контролю
		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	
Лекційні заняття	16	2 ПК	-	2 ПК	-	2 ПК	-	2 ПК	-	2 ПК	-	2 ПК	-	2 ПК	-	2 ПК	-	залік
Лабораторні заняття	32	2 ПК	2 ПК	2 ПК	2 ПК	2 ПК	2 ПК	2 ПК	2 ПК	2 ПК	2 ПК	2 ПК	2 ПК	2 ПК	2 ПК	2 ПК	2 ПК	
Самостійна робота	72	4	5	4	5	4	5	4	5	4	5	4	5	4	5	4	5	
Всього годин/ кредитів	120/4	8	7	8	7	8	7	8	7	8	7	8	7	8	7	8	7	

Позначки:

ПК - поточний контроль
ЗМ - складання змістових модулів