МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

**Український державний університет
науки і технологій**

Кафедра «Автоматика та телекомунікації»

В авторській редакції

КОМП'ЮТЕРНІ ТЕХНОЛОГІЇ В СИСТЕМАХ ЗАЛІЗНИЧНОЇ АВТОМАТИКИ

Навчально-методичні рекомендації
до лабораторних занять
(частина 3)

Електронне видання

ДНІПРО
2024

УДК 004.432:656.25(076.5)
К 63

Упорядники:
Р. В. Рибалка, В. В. Маловічко,
Н. В. Маловічко, К. В. Гончаров

Електронне видання

Схвалено Групою забезпечення якості освітньої програми
174 "Автоматика та автоматизація на транспорті"
Протокол № 7 від 24.04.2024
Схвалено Групою забезпечення якості освітньої програми
273 "Системи керування рухом поїздів"
Протокол № 7 від 18.04.2024

К 63 Комп’ютерні технології в системах залізничної автоматики : навчально-методичні рекомендації до лабораторних занять (частина 3) / упоряд. Р. В. Рибалка, В. В. Маловічко, Н. В. Маловічко, К. В. Гончаров ; Укр. держ. ун-т науки і технологій. – Електрон. вид. – Дніпро : УДУНТ, 2024. – 58 с.

Навчально-методичні рекомендації призначено для використання здобувачами вищої освіти, які здобувають освітній ступінь «бакалавр» за освітньо-професійними програмами «Автоматика та автоматизація на транспорті» та «Системи керування рухом поїздів» під час виконання лабораторних робіт з навчальної дисципліни «Комп’ютерні технології в системах залізничної автоматики». Навчально-методичні рекомендації містять основні теоретичні відомості, порядок виконання лабораторних робіт та питання для самоконтролю.

Іл. 12. Табл. 4. Бібліогр.: 7 назв.

ЗМІСТ

ВСТУП

Характеристика місця та значення навчальної дисципліни для підготовки фахівця. Ці навчально-методичні рекомендації (далі – НМР) до лабораторних занять складено для опанування навчальної дисципліни «Комп'ютерні технології в системах залізничної автоматики» (далі – Дисципліна), яка викладається під час першого року навчання відповідно до освітньо-професійних програм (далі – ОПП) «Автоматика та автоматизація на транспорті» (далі – ААТ) спеціальності 174 «Автоматизація, комп'ютерно-інтегровані технології та робототехніка» та «Системи керування рухом поїздів» (далі – СКРП) спеціальності 273 «Залізничний транспорт».

Опанування Дисципліни дозволяє здобувачу, який здобуває освіту за ОПП ААТ, отримати навики використання інформаційних і комунікаційних технологій, здатність вільно користуватись сучасними комп'ютерними та інформаційними технологіями для вирішення професійних завдань, програмувати та використовувати прикладні та спеціалізовані комп'ютерно-інтегровані середовища для вирішення задач автоматизації, робототехніки та зв'язку. Опанування Дисципліни дозволяє здобувачу, який здобуває освіту за ОПП СКРП, отримати навики використання інформаційних і комунікаційних технологій; розвинути здатність до абстрактного мислення, аналізу та синтезу; створити основу для застосування сучасних програмних засобів для розробки проектно-конструкторської та технологічної документації зі створення, експлуатації, ремонту та обслуговування систем керування рухом поїздів, пристроїв залізничної автоматики та їх елементів, а також для організації дії системи звітності та обліку (управлінського, статистичного, технологічного) роботи систем керування рухом поїздів та пристроїв залізничної автоматики, здійснювати діловодство, документування та управління якістю згідно нормативно-правових актів, інструкцій та методик.

Мета лабораторних занять з Дисципліни для цих НМР – сформувати у здобувача освіти (далі – Здобувач) навики: відповідно очікуваним результатам навчання, визначеним в робочій програмі Дисципліни, розробляти програму мовою програмування C# шляхом застосування основ об'єктно-орієнтованого програмування; проводити експериментальні дослідження за напрямом застосування комп'ютерних технологій, у т.ч. порівнювати результати дослідження з теоретичними положеннями Дисципліни.

Вимоги до попередніх знань та умінь: знання типових операцій з файлами та папками (створення, перейменування, копіювання тощо), навики програмування мовою C# з використанням структурованого програмування, основ роботи у Visual Studio. Використання технічних засобів: лабораторні роботи (далі – ЛР) виконуються за допомогою комп'ютера, наданого університетом, або комп'ютера Здобувача.

Вимоги до оформлення протоколу / звіту з ЛР (далі – Звіт):

– Допустимі види оформлення Звіту: електронна, паперова.

- Формат аркушу Звіту: А4.
- У разі оформлення Звіту:
  - З використанням засобів комп'ютерної техніки: гарнітура основного тексту – Times New Roman; гарнітура тексту програм – Consolas, Cascadia або інша моноширинна; розмір шрифту – мінімум 10 пт.
  - Власноруч: допускається прикріплення додатків (рисунок, таблиця, текст програми тощо), які може бути надруковано за допомогою комп'ютерної техніки.
- Звіт повинен містити інформацію, яка однозначно ідентифікує його виконавця: прізвище та ім'я Здобувача, номер академічної групи, шифр Здобувача (номер індивідуального навчального плану), номер варіанту (якщо вимагається в ЛР).
- Наповнення Звіту повинне відповідати вимогам розділу «Зміст звіту» відповідної ЛР.
- Записи відповідно порядку виконання ЛР в тексті Звіту позначаються номером відповідного пункту порядку виконання ЛР цих НМР.
- Аналіз результатів експериментів (досліджень) в ЛР повинен містити порівняння цих результатів з теоретичними положеннями Дисципліни.

Порядок проведення ЛР: Здобувач виконує експериментальні дослідження відповідно цих НМР, оформлює та захищає Звіт. Здобувач допускається до захисту Звіту, якщо наповнення Звіту відповідає розділу «Зміст звіту».

Розподіл роботи над виданням між співавторами:

- Рибалка Р. В. – вступ, розробка ЛР № 2 (сумісно з Маловічко Н. В.), 3 (сумісно з Маловічко В. В.), 5, 8;
- Маловічко В. В. – розробка ЛР № 3, 4;
- Маловічко Н. В. – розробка ЛР № 1, 2;
- Гончаров К. В. – розробка ЛР № 6, 7.

Визначення номера варіанту завдання

Номер варіанту завдання (якщо вимагається в ЛР) позначається цілим числом і дорівнює сумі двох останніх цифр у шифрі Здобувача (номер індивідуального навчального плану), якщо не вказано іншого.

Якщо чисельне значення номера варіанту завдання перевищує максимальне значення у множині номерів варіантів певної ЛР, то номером варіанту вважати число, яке відображене на цю множину за принципом «mod N». *Приклад*: мінімальний номер варіанта в певній ЛР – один, максимальний номер – 15; номер варіанта Здобувача – 17 (або, наприклад, 32); тоді за номер варіанта Здобувача прийняти значення 17–15=2 (або 32–2×15=2, відповідно).

Якщо чисельне значення номера варіанту завдання менше за мінімальне значення у множині номерів варіантів певної ЛР, то номером варіанту вважати число, яке дорівнює мінімальному значенню множини номерів варіантів цієї ЛР. *Приклад*: мінімальний номер варіанта в певній ЛР – один; номер варіанта Здобувача – нуль; тоді за номер варіанта Здобувача прийняти значення один.

ВИМОГИ З ОХОРОНИ ПРАЦІ ПІД ЧАС ВИКОНАННЯ ЛАБОРАТОРНИХ РОБІТ

Загальні положення

Якщо ЛР виконуються в спеціально призначеному приміщенні університету, то Здобувач зобов'язаний виконувати вимоги правил техніки безпеки та поводження у цьому приміщенні (ці вимоги повідомляються Здобувачу перед початком або на початку першого лабораторного заняття).

Якщо ЛР виконуються на комп'ютері, наданому університетом, то:

- Здобувачу *заборонено* змінювати налаштування програмної та технічної частини комп'ютера, окрім випадків, визначених цими НМР та вказівками відповідального працівника університету (чергового по аудиторії, викладача тощо, далі – Відповідальний).
- У разі непередбаченої роботи обладнання Здобувач зобов'язаний терміново повідомити про це Відповідальному.

Обладнання робочого місця користувача комп'ютера (далі – Користувач):

- Основне: монітор, клавіатура, робочий стіл, стілець (крісло);
- Допоміжне: підставка для ніг, шафи, полиці та інше.

Взаємне розташування елементів робочого місця Користувача не повинно заважати виконанню необхідних рухів та переміщень для експлуатації комп'ютера; сприяти оптимальному режиму праці й відпочинку, зниженню втоми Користувача.

Поверхню екрана монітора потрібно розташувати в оптимальній зоні інформаційного поля в площині, перпендикулярній нормальній лінії погляду Користувача, який знаходиться в робочій позі. Допускається:

- відхилення від цієї площини не більше 45°;
- відхилення лінії погляду від нормальної не більше 30°.

Розташувати монітор на робочому місці необхідно так, щоб поверхня екрана знаходилась на відстані 500…600 мм від очей Користувача, залежно від розміру екрана.

Клавіатуру потрібно розташовувати на робочому столі (не допускаючи її хитання) або на окремому столі (якщо клавіатуру виконано як окремий пристрій) на відстані 100…300 мм від краю, що є ближчим до Користувача. Положення клавіатури та кут її нахилу (в межах 5…15°) повинні відповідати побажанням Користувача.

Крісло повинно забезпечувати підтримування раціональної робочої пози під час виконання основних операцій. Поверхня сидіння має бути плоскою, передні краї – закругленими.

Раціональна поза Користувача: розташування тіла, при якому ступні Користувача розташовані на площині підлоги або на підставці для ніг, стегна зорієнтовані у горизонтальній площині, верхні частини рук – вертикальні, кут ліктьового суглоба коливається у межах 70…90°, зап'ястя зігнуті під ку-

том не більше ніж 20°, нахил голови – у межах 15…20°, також виключені часті її повороти.

Вимоги безпеки перед початком роботи

Оглянути робоче місце щодо відсутності сторонніх предметів. Якщо комп'ютер виконано у версії, якою передбачено під'єднання периферійного обладнання за допомогою з'єднувальних шнурів (кабелів), то перевірити, чи все необхідне обладнання з'єднано відповідно.

Перевірити надійність встановлення апаратури на робочому столі. Монітор повинен розміщуватись *не* на краю стола. Повернути монітор так, щоб було зручно дивитися на екран – під прямим кутом (а не збоку) і трохи зверху вниз; при цьому екран має бути трохи нахиленим – нижній його край ближче до Користувача. Оглянути та перевірити загальний стан апаратури, справність електропроводки, з'єднувальних шнурів (кабелів), штепсельних вилок, розеток, заземлення захисного екрана.

У разі виявлення будь-яких несправностей чи невідповідностей – роботу *не* розпочинати і повідомити про це Відповідальному.

Вимоги безпеки під час роботи

Під час роботи на клавіатурі Користувач повинен сидіти прямо, не напружуватися.

Заборонено:

- самостійно ремонтувати та очищувати апаратуру на робочому місці;
- перевищувати тривалість безперервної роботи за монітором, що складає 2 години без регламентованої перерви.

Вимоги безпеки після закінчення роботи

У разі потреби – зберегти файли, які є відкритими та перебувають в режимі редагування. Забрати з робочого місця особисті речі Користувача.

Вимоги безпеки в аварійних ситуаціях

У випадку раптового припинення постачання електроенергії вимкнути комп'ютер у такій послідовності: периферійні пристрої (у т.ч. монітор), процесор, стабілізатор напруги (якщо є), витягнути штепсельні вилки з розеток.

У разі виявлення ознак горіння (дим, запах гару):

1. Вимкнути апаратуру, повідомити Відповідальному (у разі його відсутності – знайти джерела займання і вжити заходів щодо ліквідації займання).

2. Якщо немає можливості швидкого відключення електропроводів від джерел постачання, частину що горить, потрібно тушити тільки вуглекислотним вогнегасником або сухим піском.

Якщо стався нещасний випадок, потрібно:

1. Надати потерпілому першу медичну допомогу.

2. Повідомити Відповідальному.
3. У разі потреби – викликати «швидку допомогу».

ОПИС ЕКСПЕРИМЕНТАЛЬНОЇ УСТАНОВКИ

Усі ЛР виконуються з використанням комп'ютера. Вимоги до комп'ютера – установлено:

- інтегроване середовище розробки Visual Studio версії не раніше 2008 року з англійською локалізацією. Visual Studio Community 2022 доступно для завантаження за посиланням https://visualstudio.microsoft.com/thank-you-downloading-visual-studio/?sku=Community&channel=Release&version=VS2022&source=VSLandingPage&cid=2030&passive=false;
- .NET Framework версії не раніше 3.5 (доступно для завантаження на сторінці за посиланням https://dotnet.microsoft.com/en-us/download/dotnet-framework) або .NET (доступно для завантаження на сторінці за посиланням https://dotnet.microsoft.com/en-us/download).

ПОЧАТКОВІ НАЛАШТУВАННЯ

Якщо ЛР виконується на комп'ютері, наданому університетом, то для збереження результатів виконання ЛР Здобувач повинен створити папку (далі – Робоча папка):

1. Уточнити у викладача розташування, призначене для збереження файлів Здобувачів, та перейти до цього розташування (наприклад, за допомогою файлового менеджера).

2. Створити Робочу папку з назвою, яка має структуру «НомерГрупи»-«_»-«Прізвище», де «НомерГрупи» та «Прізвище» – номер групи та прізвище Здобувача відповідно. *Приклад*: «АБ1234_Шевченко» без подвійних лапок. В подальшому результати всіх ЛР зберігати лише у даній Робочій папці (якщо не вказано іншого).

3. Створити папку з назвою, яка має структуру «ЛР»-«_»-«номер», де «номер» – номер поточної ЛР. *Приклад*: «ЛР_1» без подвійних лапок.

4. Перейти до папки «ЛР_номер». В подальшому результати всіх ЛР зберігати у відповідних папках (якщо не вказано іншого).

ЛАБОРАТОРНА РОБОТА № 1
ДОСЛІДЖЕННЯ ОСНОВ БУДОВИ КЛАСУ НА ПРИКЛАДІ МОДЕЛЮВАННЯ ЛАМПИ ТА ЛІНЗОВОГО КОМПЛЕКТУ СВІТЛОФОРУ

Мета роботи: дослідити основи будови класу (константи, поля, властивості, методи) та звертання до об’єктів на прикладі моделювання лампи та лінзового комплекту світлофору.

Основні теоретичні відомості

На теперішній час існує декілька способів («парадигм») розроблення програмного забезпечення, наприклад, процедурне програмування (procedural programming), об’єктно-орієнтоване програмування (ООП, object-oriented programming або ООР англійською). Вони відрізняються підходами до структурування та організації тексту програм. Як правило, процедурне програмування використовують у порівняно невеликих проектах, в яких дані та підпрограми, що обробляють ці дані, зазвичай зберігаються окремо. Дані передаються між підпрограмами як аргументи.

ООП створено для більш ефективного керування складністю процесу розроблення програмного забезпечення ніж процедурне програмування. Деякі переваги ООП:

- Ефективність під час моделювання складних систем з багатьма сутностями, що взаємодіють між собою. Сутності представляються як *об’єкти*, а їх відношення, зокрема, як *єрархія класів* (класи узгоджено за рівнями підпорядкованості для обмінювання даними згідно з конкретними правилами).
- Можливість приховувати інформацію щодо конструкцій мови програмування (мовних конструктів) – *інкапсулювати* (encapsulate). Тобто об’єднати дані (стан) та підпрограми (поведінка або функційні елементи) в одному класі. Доступ до даних із програми поза межами класу (ділянки визначення – scope), можна організувати не безпосередньо до даних, а через певну підпрограму в класі (метод, властивість тощо). Це дозволить реалізувати контрольований доступ до даних класу. Наприклад, перед записом значення до даних класу перевіряти виконання певної умови.

Метод чи мова програмування, що підтримує об’єкти, класи та успадковування, називається об’єктно-орієнтованими (object-oriented) [1]. *Об’єктом* (object) у мові програмування називається набір операцій та даних, які зберігають і підтримують ефект операцій [1]. А *класом* (class) у мові програмування називається *шаблон* (тип даних) для об’єктів, що визначає внутрішню структуру й набір операцій для екземплярів таких об’єктів [1]. Об’єкт можна розуміти як *екземпляр класу*.

Клас у C# декларується використовуючи ключове слово `class`. Тип, який декларовано як `class`, являється типом *посиланням* (reference type). Тому, якщо декларувати змінну цього типу, то змінна міститиме значення `null` (літерал, який представляє порожнє посилання, що не посилається ні на один об'єкт [7]) доки не створити екземпляр класу (об'єкт) явно.

У кожного об'єкта (в загальному випадку – мовної конструкції) є частина тривалості виконання, протягом якої цей об'єкт існує. Ця частина тривалості виконання називається *життєвий цикл* і складається з таких етапів:

1. Створення. *Приклад*: екземпляр класу створюється використовуючи оператор `new`.

2. Використання. *Приклад*: звертання до елементів екземпляра класу.

3. Знищення. Для *керованих* (managed) об'єктів виконується автоматично під час роботи «збирача сміття» (garbage collector), який входить до складу загальномовного середовища виконання (common language runtime – CLR).

Клас C# – структура даних, яка може містити декларації таких елементів:

- Елементи дані (стан або атрибути): константи, поля (змінні).
- Функційні елементи (поведінка): методи, властивості, конструктори тощо.
- Вкладені типи: класи, інтерфейси, структури, типи переліки.

У даній ЛР використано такі елементи класу C#:

- *Поле* (field) – змінна будь-якого типу, яку декларовано безпосередньо у класі чи структурі (`struct`) [7]. Поля являються елементами типів, які містять у собі ці поля.
- *Властивість* (property) – елемент, який надає гнучкий механізм читати, записувати або обчислювати значення *приватних* полів [7] (які безпосередньо не є доступними користувачеві). Властивість дозволяє класу надати публічну можливість отримання (get) та задання (set) значень, приховуючи реалізацію коду, який виконує певні перевірки. Містить *методи доступу* (accessors):
  - `get` – ключове слово, яке визначає метод доступу, що *повертає* значення властивості [7].
  - `set` – ключове слово, яке визначає метод доступу, що *присвоює* значення властивості [7]. Посилання на значення, яке присвоюється властивості у викликаючому коді, зберігається у контекстному ключовому слові `value`.
  - `init` – ключове слово, яке визначає метод доступу, що присвоює нове значення тільки під час створення об'єкта. Використовує ключове слово `value` аналогічно методу доступу `set`.
- *Метод* (method) – блок коду, який містить послідовність операторів (statements) [7]. Програма спричиняє виконання операторів шляхом виклику методу та визначення необхідних аргументів методу.

Всі типи та їх елементи мають певний рівень повноважень, що його вимагають від об'єкта для отримання доступу до захищеного ресурсу, тобто *pi-*

вень доступу (access level) [1]. Рівень доступу визначає чи може бути надано доступ з іншого коду у *збірці* (DLL або EXE файл, які створено шляхом компіляції одного або більше CS файлів) або інших збірках.

У даній ЛР використано такі модифікатори доступу до елементів класу C#:

- `public` – ключове слово, яке є модифікатором доступу для типів та елементів типів, і є найбільш дозвільним рівнем доступу [7]. Елемент класу може бути доступним з *будь-якого* іншого коду у тій самій збірці, або іншій збірці, яка посилається на цю збірку.
- `private` – ключове слово, яке є модифікатором доступу для елемента типів, і є найменш дозвільним рівнем доступу [7]. Елемент класу може бути доступним тільки з коду у тому самому класі.

Коли додаток C# запускається на виконання, то першим методом, який викликається, є метод `Main`. Тобто він є вхідною точкою додатку C#. Метод `Main` повинен бути *статичним* (`static`).

`static` – ключове слово, яке є модифікатором, що декларує *статичний* елемент, який належить типу, а *не екземпляру* цього типу [7].

У даній ЛР досліджуються основи будови класу та звертання до об’єктів на прикладі моделювання лампи та лінзового комплекту світлофору з використанням консольного додатку.

Об’єкт моделювання

На мережі залізниць, як постійні сигнальні прилади, застосовуються світлофори [4]. *Світлофор* – оптичний прилад залізничної сигналізації, за допомогою якого у будь-який час доби кольором передають на відстань накази, що прямо відносяться до руху поїздів [6].

Лінзовий світлофор [2] – світлофор, в якому для кожного сигнального показання є окрема оптична система – лінзовий комплект (ЛК), який складається із зовнішньої ступеневої безбарвної лінзи та внутрішнього світлофільтру-лінзи [3]. Для лінзових світлофорів застосовуються одноніткові та двоніткові лампи. Схему взаємодії лампи та лінзового комплекту лінзового світлофору показано на рис. 1.1.

Можливості функційної моделі лампи (рис. 1.2 *а*):

- задавати (за абсолютним значенням) та отримувати (вимірювати) величину електричної напруги (В) на лампі;
- отримувати відомості про справність лампи («справна», «несправна»);
- пошкоджувати лампу (тільки один раз).

Можливості функційної моделі ЛК (рис. 1.2 *б*):

- отримувати відомості про справність лінзи («справна», «несправна»);
- пошкоджувати лінзу (тільки один раз).

Постановка задачі

Розробити консольний додаток, який:

- реалізує можливості функційних моделей лампи та ЛК;

– виводить на екран: інформацію про справність лампи та ЛК, напругу на лампі.

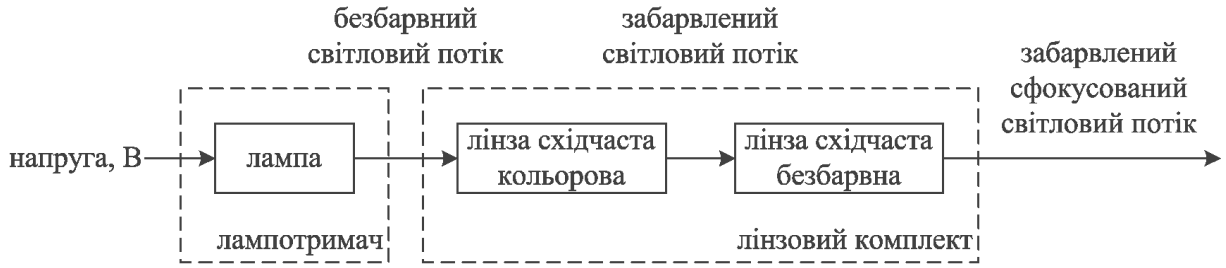

Рис. 1.1. Схема взаємодії лампи та лінзового комплекту лінзового світлофору

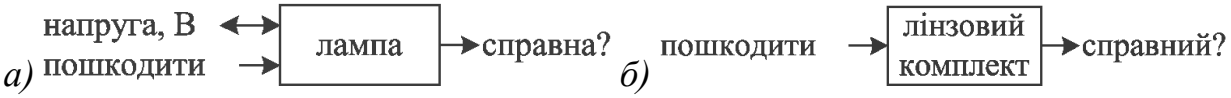

Рис. 1.2. Функційна схема моделей лампи (*а*)
та лінзового комплекту (*б*) лінзового світлофору

Порядок виконання лабораторної роботи

1. В Робочій папці створити папку для поточної ЛР.

2. Створити у Visual Studio (далі – VS) рішення (solution) `RailwayLightSignal` з проектом `ConsoleApp` (розташування та назва елементів інтерфейсу користувача VS залежить від версії VS), яке розташовано (поле `Location`) в папці поточної ЛР:

2.1. Запустити VS.

2.2. Натиснути лівою кнопкою миші (далі – ЛКМ) по кнопці `Create a new project`.

2.3. З випадних списків обрати мову `C#`, платформу – `Windows`, тип проекту – `Console`.

2.4. ЛКМ по `Console Application` (або `Console App`). Натиснути кнопку `Next` (`OK` або `Create`).

2.5. В полі `Project Name` ввести `ConsoleApp`, а в полі `Solution name` ввести `RailwayLightSignal`.

2.6. Задати розташування (поле `Location`) ідентичне папці поточної ЛР.

2.7. З випадного списку обрати .NET Framework 3.5 (або більш нову версію), якщо доступно (залежить від обраної реалізації .NET та версії VS).

2.8. Натиснути кнопку `Next` (`OK` або `Create`).

2.9. Якщо VS надає можливість *не* використовувати «оператори верхнього рівня» (“do not use top-level statements”), то активувати цю можливість (залежить від обраної реалізації .NET та версії VS). Натиснути кнопку `Next` (`OK` або `Create`).

3. Виконати початкові налаштування VS: відобразити вікна (меню View) Solution Explorer, Error List, Toolbox, Properties Window.

4. До проекту ConsoleApp додати заготовки класу Lamp (моделює роботу лампи) та класу LensKit (моделює роботу ЛК) в окремих файлах:

4.1. У вікні Solution Explorer натиснути правою кнопкою миші (далі – ПКМ) по назві проекту ConsoleApp, Add, New Item (якщо запропоновано опцію Show All Templates, то скористатись нею), натиснути лівою кнопкою миші (ЛКМ) на Class, задати в текстовому полі Name назву файлу Lamp.cs, натиснути на кнопку Add.

4.2. У вікні Solution Explorer ПКМ по назві проекту ConsoleApp, Add, New Item, ЛКМ по Class, задати в текстовому полі Name назву файлу LensKit.cs, натиснути на кнопку Add.

5. У файловому менеджері операційної системи відкрити папку проекту ConsoleApp. Назви файлів, які мають розширення CS – до Звіту.

6. Отримати у викладача файл CodeFile1.cs. Скопіювати його до папки проекту ConsoleApp та відкрити.

7. З файлу CodeFile1.cs доповнити тіло класу LensKit:

- isGoodState – поле типу bool. *Призначення*: зберігати інформацію щодо справності ЛК.
- IsGoodState – властивість типу bool. *Призначення*: повертати інформацію щодо справності ЛК:
  - true – ЛК справний,
  - false – в іншому випадку.
- Damage() – метод, тип значення, що повертається – void. *Призначення*: моделювати пошкодження ЛК.

8. З файлу CodeFile1.cs доповнити тіло класу Lamp:

- voltage – поле типу double. *Призначення*: зберігати напругу на лампі, В.
- Voltage – властивість типу double. *Призначення*: повертати та задавати напругу (за абсолютним значенням) на лампі, В.
- isGoodState – поле типу bool. *Призначення*: зберігати інформацію щодо справності лампи.
- IsGoodState – властивість типу bool. *Призначення*: повертати інформацію щодо справності лампи:
  - true – лампа справна,
  - false – в іншому випадку.
- Damage() – метод, тип значення, що повертається – void. *Призначення*: моделювати пошкодження лампи.

9. Закрити файл CodeFile1.cs.

10. Створити лампу (об’єкт класу Lamp) з іменем lamp1; вивести значення електричної напруги на лампі до консольного вікна. Для цього – внести зміни до тіла методу Main (файл Program.cs). Текст програми вводити між

фігурними дужками { }, які обмежують тіло методу Main. Ввести код, після виконання якого:

10.1. В одному рядку коду буде декларовано змінну lamp1 класу Lamp і одразу ініційовано створеним об'єктом класу Lamp

```
Lamp lamp1 = new Lamp(); // створити лампу lamp1
```

10.2. До консольного вікна буде виведено значення електричної напруги (читання з властивості Voltage), яке міститься в лампі lamp1 за промовчанням (тобто одразу після створення)

```
Console.WriteLine("Напруга на lamp1 (за промовчанням), В:"
    + lamp1.Voltage); // вивести напругу до консолі
```

11. Скомпілювати рішення: меню Build\ Build solution. Повідомлення про успішність компіляції (і помилки, якщо є) – до Звіту.

12. Задати напругу на лампі lamp1 рівною 11,2 В:

12.1. Доповнити тіло методу Main (файл Program.cs) надрукувавши з нового рядку після раніше введеного коду

```
lamp1.Voltage = 11.2;
```

12.2. Аналогічно п. 10.2 дописати код для виведення до консольного вікна значення електричної напруги (властивість Voltage) на лампі lamp1. Цього разу, значення буде зміненим, тобто не тим, що було за промовчанням.

12.3. Для того, щоб консольний додаток очікував натиснення клавіші Enter, а потім закрився (у т.ч. залежить від обраної реалізації .NET), в кінці тіла методу Main доповнити

```
Console.ReadLine();
```

В подальшому доповнювати тіло методу Main розміщуючи код *вище* даного рядка.

13. Запустити проект в режимі налагодження: меню Debug\ Start Debugging. Значення напруги на лампі lamp1 за промовчанням та після зміни – до Звіту. Закрити консольне вікно.

14. Створити ЛК з іменем lensKit1, «пошкодити» його (метод Damage()), вивести до консольного вікна інформацію про справність lensKit1:

14.1. Доповнити тіло методу Main

```
LensKit lensKit1 = new LensKit();
lensKit1.Damage();
```

14.2. Самостійно дописати код для виведення до консольного вікна інформації про справність lensKit1 (властивість IsGoodState, аналогічно п. 10.2).

15. Запустити проект в режимі налагодження. Повідомлення про справність ЛК lensKit1 після пошкодження – до Звіту. Закрити консольне вікно.

16. У методі `Main` самостійно виконати таке:

16.1. Створити лампу `lamp2`, задати їй довільне числове значення електричної напруги x, де x – конкретне число, що не було використано в тексті програми раніше.

16.2. Присвоїти лампі `lamp1` лампу `lamp2`.

16.3. Задати лампі `lamp1` довільне числове значення електричної напруги y, де y – конкретне число, що не було використано в тексті програми раніше.

16.4. Заповнити табл. 1.1 (до Звіту) значеннями властивості `Voltage` ламп `lamp1` та `lamp2` *після* виконання відповідних рядків програми. Наприклад, використовуючи точки переривання і покрокові операції.

Таблиця 1.1

Значення напруги після виконання відповідних рядків програми

Опис рядку методу `Main`	Значення напруги, В	
	`lamp1`	`lamp2`
створення `lamp2`		
запис напруги x в `lamp2`		
`lamp1 = lamp2;`		
запис напруги y в `lamp1`		

17. Зберегти проект. Остаточний текст програми – до Звіту. Закрити поточне рішення.

Зміст звіту

1. Номер, тема, мета ЛР.
2. Пункти порядку виконання лабораторної роботи: 5, 11, 13, 15, 16.4, 17.
3. Рішення в папці `RailwayLightSignal` з проектом `ConsoleApp`.
4. Висновки.

Контрольні запитання

1. ООП. Клас. Будова класу.
2. ООП. Об’єкт. Створення екземпляра класу.
3. ООП. Клас. Поле класу.
4. ООП. Клас. Властивість класу.
5. ООП. Модифікатори `public`, `private`, `static`.

ЛАБОРАТОРНА РОБОТА № 2
ДОСЛІДЖЕННЯ СПІВВІДНОШЕННЯ «АГРЕГАЦІЯ» НА ПРИКЛАДІ МОДЕЛЮВАННЯ СИГНАЛЬНОГО ВОГНЮ ЛІНЗОВОГО СВІТЛОФОРУ

Мета роботи: дослідити роботу з елементами класу для співвідношення «агрегація» на прикладі моделювання сигнального вогню лінзового світлофору; отримати поняття про конструктори класу.

Основні теоретичні відомості

Створення екземпляра класу є початком життєвого циклу цього екземпляра. Під час створення екземпляра класу C# викликається його *конструктор* (їх може бути декларовано декілька в одному класі). Серед іншого, конструктори дозволяють розробнику

- встановити значення *за промовчанням* (значення без задання або default), наприклад, ініціалізувати елементи дані екземпляра класу;
- реалізувати певну поведінку («логіку») під час створення екземпляра класу.

Конструктор в C# це метод, що на відміну від звичайних методів має сигнатуру методу, яка включає тільки модифікатор доступу (опціонально), ім’я методу та список параметрів, і *не* включає тип, що повертається. При цьому ім’я конструктора ідентичне імені типу, в якому декларовано цей конструктор.

В C# існують різні види конструкторів. Залежно від приналежності до екземпляра класу чи безпосередньо класу:

- *Конструктор екземпляру* (instance constructor) – елемент класу, в якому визначено код, що виконується під час створення нового екземпляра класу з використанням виразу `new` [7].
- *Статичний конструктор* (static constructor) – елемент класу, в якому визначено код, що повинен бути виконаний тільки *один* раз [7]. Наприклад, ініціалізувати *статичні* дані. Викликається автоматично перед тим як створено перший екземпляр, або було звернення до будь-якого статичного елемента.

Конструктор, який не приймає жодних параметрів, називається *конструктором без параметрів* (parameterless constructor). Якщо клас не являється `static` і не містить жодного конструктора, то такому класу компілятором C# надається публічний (`public`) конструктор без параметрів, щоб забезпечити створення екземпляра класу. Інколи такий конструктор називають *конструктор за промовчанням* (default constructor).

Якщо відсутня потреба реалізовувати якусь додаткову поведінку властивості, то для зменшення обсягу коду, окрім, наприклад, конструктора за промовчанням, у C# передбачено інші засоби, зокрема, *автоматично реалізована властивість* (auto-implemented property). *Автоматично реалізована властивість* робить декларацію властивості більш стислою (менше коду), якщо в

її методах доступу не потрібно реалізовувати додаткову поведінку. Після декларування такої властивості компілятор створює `private` поле, яке доступне тільки через методи доступу цієї властивості (`get`, `set`, `init`).

Серед найкращих практик розроблення програмного забезпечення є реалізація того, щоб у разі потреби внести зміни до існуючого коду (наприклад, до певного модуля) зменшити вплив цих змін на інший (як правило вже працюючий і перевірений) код (інші модулі). Тобто замість проекту з *сильним* сполученням (tightly coupled design) між модулями розробити проект зі *слабким* сполученням (loosely coupled design).

Серед способів досягти слабкого сполучення між модулями, окрім інкапсуляції, є відношення «агрегація» (aggregation). Якщо між класами є відношення «агрегація», то один клас (агрегатор) містить в собі екземпляр *іншого* класу. При цьому життєві цикли цих класів *не* залежать один від одного. Тобто після знищення екземпляру агрегатора екземпляр іншого класу (того, що входив до складу агрегатора) *не* буде автоматично знищено (і навпаки).

У даній ЛР досліджується співвідношення «агрегація» на прикладі моделювання сигнального вогню лінзового світлофору з використанням додатку `Windows Forms`. При цьому використано класи та елементи керування `Windows Forms`, що подано нижче.

`CheckBox` – клас (простір імен `System.Windows.Forms`), представляє поле для помітки у Windows [7]. Деякі властивості:

- `Checked` – властивість (типу `bool`), отримує чи задає значення, яке визначає, чи знаходиться `CheckBox` у поміченому стані [7].
- `Enabled` – властивість (типу `bool`), отримує чи задає значення, яке визначає, чи може елемент керування реагувати на дії користувача [7].

Об’єкт моделювання

Лінзовий комплект (ЛК) з *лампотримачем* під двониткову лампу (рис. 1.1) призначений для встановлення в головках світлофорів. При *денному* режимі живлення напруга на затискачах лампотримача лінзових світлофорів повинна бути 11.5 В (допустиме відхилення +0.5 В та −1.0 В) [5].

Лінзи для ЛК надаються лише у вигляді сфокусованих ЛК і окремо не надаються. Зібрані ЛК фокусуються, після чого фіксуються лампотримачі для забезпечення надійного фокусування лампи.

Функційна модель сигнального вогню лінзового світлофору (ЛК з лампотримачем) (рис. 2.1):

- складається з лампи та ЛК;
- містить нижнє (10.5 В) та верхнє (12 В) гранично допустимі значення напруги на лампі для денного режиму живлення;
- можливості:
  - отримувати відомості про видимість вогню: «в межах норми» – значення напруги на лампі знаходиться в границях 10.5…12 В, лампа та ЛК є справними; «поза межами норми» – в іншому випадку;

◦ замінювати лампу.

Постановка задачі

Розробити додаток Windows Forms, який:

– реалізує можливості функційної моделі сигнального вогню лінзового світлофору в умовах денного режиму живлення;
– дозволяє користувачу: задавати напругу на лампі та справність лампи (імітувати пошкодження лампи); замінювати лампу;
– виводить на екран: напругу на лампі, відомості про справність лампи та видимість сигнального вогню.

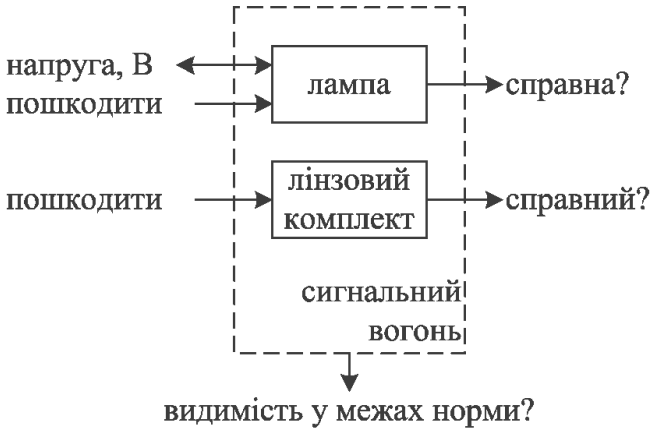

Рис. 2.1. Функційна схема моделі сигнального вогню лінзового світлофору

Порядок виконання лабораторної роботи

1. Отримати у викладача папку з проектом WindowsFormsApp.

2. Додати проект WindowsFormsApp до рішення RailwayLightSignal:

2.1. За допомогою файлового менеджера операційної системи скопіювати папку з проектом WindowsFormsApp до папки рішення RailwayLightSignal, яке створено в ЛР № 1.

2.2. Відкрити рішення RailwayLightSignal (файл з розширенням SLN).

2.3. У вікні Solution Explorer ПКМ по назві рішення, ЛКМ по Add, Existing Project, перейти до папки WindowsFormsApp і обрати файл проекту (файл з розширенням CSPROJ), натиснути на кнопку Open.

2.4. У VS встановити проект WindowsFormsApp *стартовим* проектом: у вікні Solution Explorer ПКМ по назві проекту WindowsFormsApp, Set As Startup Project.

2.5. Назви проектів у рішенні RailwayLightSignal (вікно Solution Explorer) – до Звіту.

3. Ознайомитися з формою проекту WindowsFormsApp (рис. 2.2) в режимі конструктора. Для цього: у вікні Solution Explorer розгорнути вміст проекту, ПКМ по Form1.cs, у контекстному меню ЛКМ по View De-

signer; почергово ЛКМ по кожному компоненту на формі, під час цього у вікні Properties, на вкладці Properties ( ) переглядати значення властивості Name.

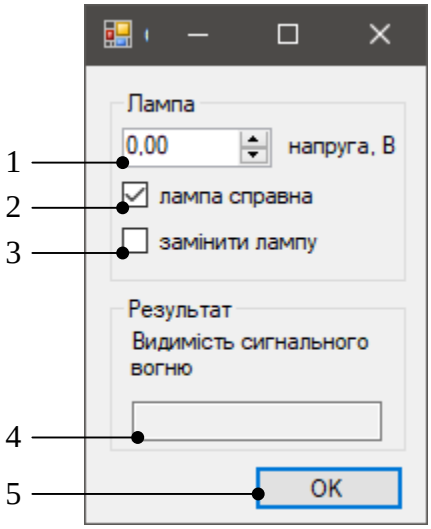

Рис. 2.2. Вікно проекту WindowsFormsApp з елементами керування:
1 – NumericUpDown; *2*, *3* – CheckBox; *4* – TextBox; *5* – Button

4. В режимі конструктора доповнити форму проекту WindowsFormsApp елементами CheckBox (*2* та *3* на рис. 2.2). Задати їх властивостям Name (вікно Properties, вкладка Properties) такі значення: chkBxLampIsGoodState та chkBxReplaceLamp, відповідно.

5. До Звіту занести:

– ескіз (або скріншот) форми (подібно до рис. 2.2);
– перелік відповідностей «елемент керування на ескізі (скріншоті) форми» – «властивість Name цього елемента» (подібно до рис. 2.2).

Примітки: на рис. 2.2 подано перелік відповідностей «елемент керування» – «ім'я *класу* цього елемента», а не «*властивість* Name цього елемента».

6. Скопіювати файли Lamp.cs та LensKit.cs з проекту ConsoleApp до проекту WindowsFormsApp:

6.1. У вікні Solution Explorer ПКМ по назві проекту WindowsFormsApp, Add\Existing Item.

6.2. Перейти до папки проекту ConsoleApp, обрати файли Lamp.cs та LensKit.cs, натиснути на кнопку Add.

7. Дозволити у файлі Form1.cs використовувати класи Lamp та LensKit. Для цього – доповнити перелік using на початку файлу Form1.cs рядком

```
using ConsoleApp;
```

8. До проекту WindowsFormsApp додати заготовку класу LightSignal (моделює роботу сигнального вогню лінзового світлофору) в окремому файлі: у вікні Solution Explorer ПКМ по назві проекту, Add, New Item (якщо

запропоновано опцію Show All Templates, то скористатись нею), ЛКМ по Class, задати в текстовому полі Name назву файлу LightSignal.cs, натиснути на кнопку Add.

9. Отримати у викладача файл CodeFile2.cs. Скопіювати його до папки проекту WindowsFormsApp та відкрити.

10. З файлу CodeFile2.cs доповнити тіло класу LightSignal такими елементами:

- Lamp – автоматично реалізована властивість типу Lamp. *Призначення*: зберігати екземпляр лампи.
- LensKit – автоматично реалізована властивість типу LensKit. *Призначення*: зберігати екземпляр ЛК.
- voltageNominalDayMode – константа типу double. *Призначення*: зберігати значення номінальної електричної напруги на лампі у денному режимі живлення, В.
- voltageMinDayMode – константа типу double. *Призначення*: зберігати нижню границю допустимого діапазону електричної напруги на лампі (денний режим живлення), В.
- voltageMaxDayMode – константа типу double. *Призначення*: зберігати верхню границю допустимого діапазону електричної напруги на лампі (денний режим живлення), В.
- IsVisible – властивість, яка *повертає* значення типу bool: true – «видимість в межах норми»; false – в іншому випадку.
- ReplaceLamp(double) – метод, тип значення, що повертається – double. *Призначення*: «замінити» поточну лампу на новий екземпляр лампи із заданою електричною напругою, В.
- LightSignal() – конструктор. *Призначення*: створити екземпляр сигнального вогню лінзового світлофору з екземплярами лампи та ЛК.

11. Закрити файл CodeFile2.cs. Самостійно (аналогічно п. 7) дописати код, який дозволяє у файлі LightSignal.cs використовувати класи Lamp та LensKit (простір імен ConsoleApp).

12. Відкрити файл Form1.cs в режимі редактора коду: в Solution Explorer ПКМ по Form1.cs, ЛКМ по View Code.

Клас Form1 містить методи користувача:

- ParseDataUpdateForm(LightSignal). *Призначення*: читати дані з форми, змінювати параметри сигнального вогню, оновити стан елементів керування на формі відповідно параметрам сигнального вогню.
- OutputOfResult(LightSignal). *Призначення*: виводити на форму відомості про видимість сигнального вогню.

13. Доповнити тіло класу Form1 полем lightSignal типу LightSignal. Ініціювати поле конструктором типу LightSignal за промовчанням.

14. Доповнити тіло обробника події btnOK_Click кодом, після виконання якого буде викликано:

14.1. Метод `ParseDataUpdateForm` з аргументом `lightSignal`.

14.2. Метод `OutputOfResult` з аргументом `lightSignal`.

15. Скомпілювати та запустити проект в режимі налагодження.

16. Дослідити функціонування додатку, заповнити табл. 2.1 (до Звіту):

16.1. Задати значення напруги, що виходить за межі діапазону для денного режиму живлення. Натиснути кнопку `OK`, результат – до колонки «Напруга поза нормою» (табл. 2.1).

Таблиця 2.1

Параметр	Напруга поза нормою	Напруга в нормі	Лампу пошкоджено	Лампу замінено
Напруга, В				
Лампа справна				
Видимість вогню				

16.2. Задати значення напруги в межах діапазону для денного режиму живлення. Натиснути кнопку `OK`, результат – до колонки «Напруга в нормі» (табл. 2.1).

16.3. За нормальної напруги пошкодити лампу: зняти відмітку «лампа справна» у вікні додатку (*2* на рис. 2.2), натиснути кнопку `OK`, результат – до колонки «Лампу пошкоджено» (табл. 2.1).

16.4. Замінити лампу: встановити відмітку «замінити лампу» у вікні додатку (*3* на рис. 2.2), натиснути кнопку `OK`, результат – до колонки «Лампу замінено» (табл. 2.1). Закрити додаток.

17. У Звіті привести алгоритм функціонування методів `btnOK_Click`, `ParseDataUpdateForm` та `OutputOfResult` з коментарями.

18. Зберегти проект. Остаточний текст програми – до Звіту. Закрити поточне рішення.

Зміст звіту

1. Номер, тема, мета ЛР.

2. Пункти порядку виконання лабораторної роботи: 2.5, 5, 16–18.

3. Рішення в папці `RailwayLightSignal` з проектами `WindowsFormsApp` та `ConsoleApp`.

4. Висновки.

Контрольні запитання

1. Клас. Конструктор класу.
2. Клас. Конструктор екземпляру класу.
3. Клас. Конструктор за промовчанням.
4. Клас. Автоматично реалізована властивість.
5. Співвідношення між класами виду «агрегація».

ЛАБОРАТОРНА РОБОТА № 3
ДОСЛІДЖЕННЯ СПІВВІДНОШЕННЯ «УСПАДКОВУВАННЯ» НА ПРИКЛАДІ МОДЕЛЮВАННЯ ІМПУЛЬСНОГО РЕЙКОВОГО КОЛА

Мета роботи: дослідити роботу з елементами класу для співвідношення «успадковування» на прикладі моделювання імпульсного рейкового кола; сформувати уявлення про роботу імпульсного рейкового кола.

Основні теоретичні відомості

Серед типових вимог до ООП, окрім, наприклад, інкапсуляції (приховування інформації), є успадковування, поліморфізм й абстракція даних. У мовах програмування *успадковування властивостей* (inheritance) – копіювання всієї або частини внутрішньої структури й набору операцій з одного класу (базовий або base клас) до підлеглого класу (похідний або derived клас) [1]. Переваги успадковування:

- Можливість повторно використовувати код, тобто виключити з програми фрагменти коду, які повторюються. В модульному програмуванні це також можна досягти використовуючи підпрограми, але в ООП ці підпрограми можна розмістити у класах разом з елементами даними (у разі потреби).
- Можливість реалізувати *поліморфну* поведінку, тобто змінити реалізацію певних елементів класів (які належать до однієї єрархії класів) залежно від того, до якого класу належить об’єкт.

Недолік успадковування – воно призводить до більш сильно сполучених класів. Якщо похідний клас повинен успадковувати *не* всі елементи базового класу, то застосовується «агрегація».

Під час розроблення програмного забезпечення досить часто зустрічається задача, коли потрібно змоделювати певну поведінку декількох *схожих* між собою сутностей певного домену (конкретна галузь знань чи експертизи).

Приклад:

- домен – пристрої залізничної автоматики;
- схожі сутності – різні види рейкових кіл (див. розділ «Об’єкт моделювання»), наприклад, імпульсне рейкове коло (РК), тональне РК тощо;
- певна поведінка – процес визначення присутності рухомого складу в межах дії певного РК.

Доповнення до задачі: реалізація поведінки у кожній з вказаних сутностей повинна бути *різною*. Для прикладу, поданого вище, у різних РК передається сигнал різної форми і визначення присутності рухомого складу в межах дії РК відбувається по різному. У разі використання ООП для вирішення подібної задачі розробляється єрархія класів, де кожна зі схожих сутностей представляється відповідним класом на одному й тому ж рівні єрархії.

Для реалізації *поліморфної* поведінки:

1. Додатково створюється базовий клас (розміщено в єрархії на один рівень вище за класи моделювання схожих сутностей). Наприклад, РК. Від цього базового класу успадковуються класи схожих сутностей (різні види РК), які в єрархії розміщено рівнем нижче.

2. Якщо під час розроблення єрархії класів з'ясовано, що визначати реалізацію якогось елемента (наприклад, методу, що визначає присутність рухомого складу) у базовому класі не має смислу (метод залежить від виду РК і не може мати універсальної реалізації для всіх випадків), а всі похідні класи *зобов'язані* перевизначити (`override`) реалізацію цього елемента базового класу, то:

2.1. Такий елемент базового класу помічається модифікатором `abstract` (означає, що реалізація елементу є неповною чи відсутньою), тоді елемент називається *абстрактним*.

2.2. Оскільки декларацію абстрактних елементів дозволено тільки в класах `abstract`, то і сам базовий клас помічається модифікатором `abstract`.

2.3. Відповідні елементи похідних класів (наприклад, методи) помічаються модифікатором `override` (означає, що елемент розширює чи модифікує абстрактну або віртуальну реалізацію елемента, успадкованого від базового класу).

Якщо клас помічено модифікатором `abstract`, то такий клас називається *абстрактним класом* (abstract class). Створення екземплярів абстрактного класу *не* допускається. Ці класи призначено для того, щоб інші класи успадковувались від них.

Поліморфну поведінку буде помітно, якщо

1. Створити екземпляр якогось похідного класу (з тих, що моделюють схожі сутності), наприклад, імпульсне РК.

2. Привести його до базового типу (присвоїти змінній базового типу, або передати як аргумент параметру, що має базовий тип), тобто до РК.

3. Звернутись до елементу, який перевизначено (наприклад, визначення присутності рухомого складу в межах дії певного РК), через *базовий* тип.

Якщо ж поліморфна поведінка непотрібна і прийнято рішення, що у базовому класі доцільно розмісити реалізацію якогось елемента (наприклад, методу), то

- базовий клас не обов'язково повинен бути абстрактним;
- цей елемент у базовому класі обов'язково повинен бути *не* абстрактним;
- відповідні елементи у похідних класах можуть бути помічені модифікатором `new`, який явно *приховує* елемент, який успадковано від базового класу, і *заміняє* його елементом похідного класу. При цьому зникає попередження компілятора щодо приховування елементів.

У даній ЛР використано новий модифікатор доступу до елементів класу C#: `protected` – ключове слово, яке є модифікатором доступу для елемента

типів, і такий елемент є доступним в межах свого класу та з об’єктів класів, що походять від цього класу [7]. У даній ЛР досліджується співвідношення «успадковування» на прикладі моделювання імпульсного рейкового кола з використанням додатку `Windows Forms`.

Об’єкт моделювання

Одним з основних засобів інтервального регулювання руху поїздів є *автоматичне блокування* (АБ). За умови обладнання АБ перегін між станціями поділяється на окремі блок-ділянки (БД) і на їх границях розташовані прохідні світлофори. БД обладнуються *рейковими колами* (РК) – електричними колами, провідником в яких є рейкові нитки, що утворюють рейкову лінію (РЛ) [3]. Серед функцій РК є визначення присутності рухомого складу в межах дії РК, щоб не допустити в’їзду в межі одного РК більше однієї рухомої одиниці. Для того, щоб забезпечити електричну ізоляцію між суміжними РК їх розділено *ізолюючими стиками*. Структурну схему РК приведено на рис. 3.1.

Рис. 3.1. Структура рейкового кола

До одного кінця РК підключається джерело електричної напруги (передавач), до іншого – колійне реле (приймач). За вільної БД електричний струм (інформаційний сигнал) від джерела по одній рейковій нитці проходить через обмотку колійного реле і повертається до джерела живлення по іншій рейковій нитці. Колійне реле під дією струму в обмотці спрацьовує (замикає фронтовий контакт), що свідчить про відсутність поїзда в межах дії даного РК. Форма інформаційного сигналу залежить від різновиду РК.

Під час перебування поїзда на БД рейкові нитки електрично з’єднуються (шунтуються) колісними парами (поїзний шунт). Струм від джерела живлення не доходить до колійного реле (сигнал відсутній), а повертається через колісні пари (шунтується) до джерела живлення. Колійне реле розмикає свій фронтовий контакт, що свідчить про зайнятість даної БД.

На роботу РК впливає багато факторів, серед них – опір баласту (ізоляції) – опір між рейковими нитками. З його зниженням все більше струму не доходить від джерела живлення до колійного реле. Значення опору баласту, Ом × км:

- 20 – приймається як нескінченно велике;
- 1 та більше – нормальне функціонування РК, шунт відчувається;

– менше за 1 – РК не відчуває шунта, тобто РК є зайнятим незалежно від присутності рухомого складу на даній БД.

Можливості функційної моделі передавача РК (рис. 3.2 *а*): задавати та отримувати відомості про наявність сигналу на виході передавача.

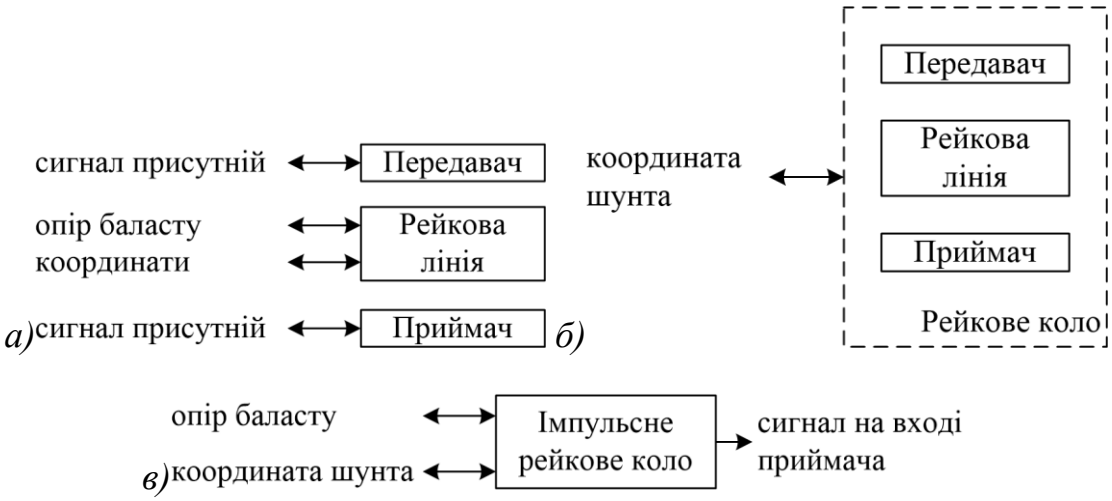

Рис. 3.2. Функційні схеми таких моделей:
а) елементи РК; *б)* абстрактне РК; *в)* імпульсне РК

Можливості функційної моделі приймача РК (рис. 3.2 *а*): задавати та отримувати відомості про наявність сигналу на вході приймача.

Можливості функційної моделі РЛ (рис. 3.2 *а*):

– задавати та отримувати величину опору баласту (невід'ємна величина), Ом × км;
– задавати (під час створення об'єкта) та отримувати значення координат початку та кінця РЛ (координата початку менша за координату кінця), м.

Функційна модель абстрактного РК (рис. 3.2 *б*):

– складається з передавача, РЛ та приймача;
– можливості: задавати та отримувати координату поїзного шунта, м.

Функційна модель імпульсного РК постійного струму (рис. 3.2 *в*):

– походить від абстрактного РК;
– містить величини мінімально допустимого опору баласту (1 Ом × км) та максимально допустимої довжини РК (2600 м);
– можливості:
  ◦ задавати та отримувати величину опору баласту (невід'ємна величина), Ом × км;
  ◦ задавати та отримувати координату поїзного шунта, м;
  ◦ отримувати відомості про наявність сигналу на приймачі: сигнал такий самий, як на передавачі (тобто «присутній»), якщо опір баласту неменший за мінімальний та координата шунта знаходиться поза межами РК; сигнал «відсутній» – в іншому випадку.

Постановка задачі

Розробити додаток Windows Forms, який:

– реалізує можливості функційної моделі імпульсного РК;
– дозволяє користувачу: задавати опір баласту, координату поїзного шунта в метрах відносно нуля (початок межі дії РК);
– виводить на екран: опір баласту, координату шунта, сигнал на вході приймача.

Порядок виконання лабораторної роботи

1. В Робочій папці створити папку для поточної ЛР.

2. Отримати у викладача папку з рішенням RailwayAutomatics та проектом RailwayTrackCircuit.

3. Ознайомитися з формою проекту RailwayTrackCircuit (рис. 3.3) в режимі конструктора. Для цього: відкрити проект RailwayTrackCircuit (розширення файлу CSPROJ) або відповідне рішення (розширення файлу SLN) у VS; у вікні Solution Explorer ПКМ по Form1.cs, у контекстному меню ЛКМ по View Designer; почергово ЛКМ по кожному компоненту на формі, під час цього у вікні Properties, на вкладці Properties () переглядати значення властивості Name.

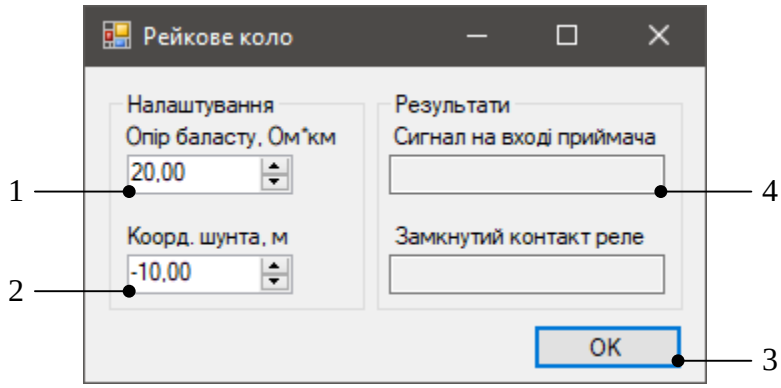

Рис. 3.3. Вікно проекту RailwayTrackCircuit
(після виконання ЛР 5) з елементами керування:
1, *2* – NumericUpDown; *3* – Button; *4* – TextBox

4. До Звіту занести:

– ескіз (або скріншот) форми (подібно до рис. 3.3);
– перелік відповідностей «елемент керування на ескізі (скріншоті) форми» – «властивість Name цього елемента» (подібно до рис. 3.3).

Примітки: на рис. 3.3 подано перелік відповідностей «елемент керування» – «ім'я *класу* цього елемента», а не «*властивість* Name цього елемента».

5. До проекту RailwayTrackCircuit додати заготовки класів в окремих файлах:

– `Transmitter` – клас, який моделює роботу передавача РК.
– `Receiver` – клас, який моделює роботу приймача РК.
– `RailLine` – клас, який моделює роботу РЛ.
– `TrackCircuit` – клас, який моделює роботу узагальненого РК. Позначити клас як *абстрактний*.
– `TrackCircuitDCImpulse` – клас, який моделює роботу імпульсного РК постійного струму. Задати в якості базового класу клас `TrackCircuit`.

Для цього для кожного класу: у вікні `Solution Explorer` ПКМ по назві проекту, `Add`, `New Item` (якщо запропоновано опцію `Show All Templates`, то скористатись нею), ЛКМ по `Class`, задати в текстовому полі `Name` назву файлу, що відповідає назві класу, натиснути на кнопку `Add`.

6. Доповнити тіло класу `Transmitter` публічною (`public`) автоматично реалізованою властивістю `IsSignalAvailable` типу `bool`. *Призначення*: моделює отримання та задання сигналу на виході передавача. Значення `true` відповідає присутності сигналу на виході передавача, `false` – відсутності сигналу.

7. Доповнити тіло класу `Receiver` публічною (`public`) автоматично реалізованою властивістю `IsSignalAvailable` типу `bool`. *Призначення*: моделює отримання та задання сигналу на вході приймача. Значення `true` відповідає присутності сигналу на вході приймача, `false` – відсутності сигналу.

8. Доповнити тіло абстрактного класу `TrackCircuit`:
– `transmitter` – поле `protected` типу `Transmitter`. *Призначення*: зберігати передавач.
– `railLine` – поле `protected` типу `RailLine`. *Призначення*: зберігати РЛ.
– `receiver` – поле `protected` типу `Receiver`. *Призначення*: зберігати приймач.
– `ShuntCoordinate` – абстрактна властивість типу `double`. *Призначення*: повертати та задавати координату шунта, м.

9. Отримати у викладача файл `CodeFile3.cs`. Скопіювати його до папки проекту `RailwayTrackCircuit` та відкрити.

10. З файлу `CodeFile3.cs` доповнити тіло класу `RailLine` такими елементами:
– `ballastResistance` та `BallastResistance` – поле та властивість типу `double`, відповідно. *Призначення*: повертати або задавати величину опору баласту (за абсолютним значенням), Ом × км .
– `BeginCoordinate` – автоматично реалізована властивість типу `double`. *Призначення*: повертати координату початку РК, м.
– `EndCoordinate` – автоматично реалізована властивість типу `double`. *Призначення*: повертати координату кінця РК, м.

– `RailLine(double, double, double)` – конструктор. *Призначення*: задати опір баласту та координати границь РК.

11. З файлу `CodeFile3.cs` доповнити тіло класу `TrackCircuitDCImpulse` такими елементами:

– `ballastResistanceMin` – константа типу `double`. *Призначення*: зберігати мінімально допустимий опір баласту, Ом × км.

– `railLengthMax` – константа типу `double`. *Призначення*: зберігати значення максимальної довжини РК, м.

– `BallastResistance` – властивість типу `double`. *Призначення*: повертати або задавати величину опору баласту РЛ (Ом × км).

– `shuntCoordinate` та `ShuntCoordinate` – поле та властивість (*перевизначає* відповідну властивість базового класу) типу `double`, відповідно. *Призначення*: повертати або задавати координату шунта (м).

– `UpdateSignalAtReceiver()` – метод, тип значення, що повертається – `void`. *Призначення*: оновлювати сигнал на приймачі.

– `IsSignalAvailableAtReceiver` – властивість типу `bool`. *Призначення*: повертати відомості про присутність сигналу на приймачі.

– `TrackCircuitDCImpulse(double, double)` – конструктор. *Призначення*: ініціювати передавач, РЛ (задати координати початку та кінця РЛ) та приймач.

12. Закрити файл `CodeFile3.cs`.

13. Відкрити `Form1.cs` в режимі редактора коду.

Клас `Form1` містить методи користувача:

– `ParseDataUpdateForm(TrackCircuitDCimpulse)` – метод, тип значення, що повертається – `void`. *Призначення*: читати дані з форми, змінювати параметри РК, оновлювати стан елементів керування на формі відповідно параметрам РК.

– `OutputOfResult(TrackCircuitDCimpulse)` – метод, тип значення, що повертається – `void`. *Призначення*: виводити відомості про присутність сигналу на вході приймача РК на форму.

14. Доповнити тіло класу `Form1` полем `trackCircuitDCImpulse` типу `TrackCircuitDCImpulse`. Ініціювати поле, викликавши конструктор класу `TrackCircuitDCImpulse` з параметрами, які задають координати початку та кінця імпульсного РК – 0 м та 2000 м, відповідно.

15. Доповнити тіло обробника події `btnOK_Click` кодом, після виконання якого буде викликано:

15.1. Метод `ParseDataUpdateForm` з аргументом `trackCircuitDCImpulse`.

15.2. Метод `OutputOfResult` з аргументом `trackCircuitDCImpulse`.

16. Скомпілювати та запустити проект в режимі налагодження.

17. Дослідити функціонування додатку, заповнити табл. 3.1 (до Звіту). Порожні комірки заповнити результуючим сигналом на вході приймача.

18. Порівняти дані табл. 3.1 з логікою роботи функційної моделі імпульсного РК постійного струму. Результат порівняння – до Звіту.

19. У Звіті привести алгоритм функціонування методу `UpdateSignalAtReceiver()` класу `TrackCircuitDCImpulse`.

Таблиця 3.1

Сигнал на вході приймача рейкового кола

Опір баласту, Ом×км	Координата шунта, м							
	–10	0	10	500	1000	1500	2000	2100
20								
5								
0,5								

20. Зберегти проект. Остаточний текст програми – до Звіту. Закрити поточне рішення.

Зміст звіту

1. Номер, тема, мета ЛР.
2. Пункти порядку виконання лабораторної роботи: 4, 17–20.
3. Рішення в папці `RailwayAutomatics` з проектом `RailwayTrackCircuit`.
4. Висновки.

Контрольні запитання

1. Співвідношення «успадковування».
2. Модифікатор `abstract`.
3. Модифікатор `override`.
4. Модифікатор `new`.
5. Модифікатор `protected`.

ЛАБОРАТОРНА РОБОТА № 4
ДОСЛІДЖЕННЯ ОСНОВ ВИКОРИСТАННЯ ІНТЕРФЕЙСІВ НА ПРИКЛАДІ МОДЕЛЮВАННЯ ЕЛЕКТРОМАГНІТНОГО РЕЛЕ

Мета роботи: дослідити взаємодію з інтерфейсами на прикладі моделювання електромагнітного реле; сформувати уявлення про роботу електромагнітних реле.

Основні теоретичні відомості

Під час розроблення програмного забезпечення з використанням ООП часто зустрічається задача, коли потрібно, щоб певний клас містив поведінку, визначену в *декількох* інших сутностях. Можливі способи вирішення:

- Використати множинне успадковування, тобто коли похідний клас може успадковуватись від більше ніж одного базового класу (наприклад, в C++). В C# множинне успадковування *не* підтримується.
- Використати *інтерфейс* C# (interface) – визначає контракт [7]. Будь-який клас (`class`), запис (`record`) чи структура (`struct`), які реалізують цей контракт (інтерфейс), повинні надати реалізацію елементів, які визначено в інтерфейсі.

У даній ЛР досліджується взаємодія з інтерфейсами на прикладі моделювання електромагнітного реле з використанням додатку `Windows Forms`. При цьому використано класи та елементи керування `Windows Forms`, що подано нижче.

`RadioButton` – клас (простір імен `System.Windows.Forms`), дозволяє користувачу обрати єдиний варіант з певної групи варіантів, коли використовується разом з іншими елементами керування `RadioButton` [7]:

- `Checked` – властивість (типу `bool`), отримує чи задає значення, яке визначає, чи знаходиться `RadioButton` у відміченому стані [7].

Об’єкт моделювання

Для реалізації системи автоматики, яка містить дискретні (у т.ч. двійкові) елементи, може бути використано комутацію електричних кіл шляхом замикання (розмикання) електричних контактів. Одним з поширених пристроїв, які це виконують, є електромагнітні реле. *Електромагнітне реле* – пристрій, який перетворює електричну величину (струм, напруга) на механічну (переміщення *якоря*).

Електромагнітне *комбіноване* реле – трипозиційне з нейтральною та поляризованою системою, що має один нейтральний та один поляризований якір. В якості комбінованого реле в даній ЛР обрано КШ1-280, функційну схему якого приведено на рис. 4.1.

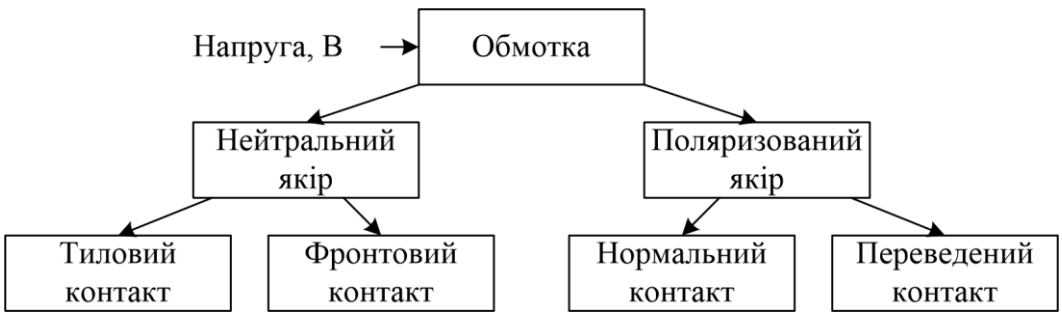

Рис. 4.1. Функційна схема електромагнітного комбінованого реле

Якір являється металевою рухомою частиною реле, на яку діє електромагнітне поле, створене електричним струмом в обмотці реле. Залежно від виду якоря (нейтральний або поляризований) та значення напруги на обмотці реле замкнутими є два контакти:

- «Тиловий» / «Фронтовий» – якщо *сили* електричного струму в обмотці недостатньо / достатньо для притягання нейтрального якоря. Напрям струму не має значення.
- «Нормальний» / «Поляризований» – якщо сили струму *певного напряму* в обмотці недостатньо / достатньо для притягання поляризованого якоря.

Функційна модель комбінованого реле КШ1-280:

- містить:
  - значення напруги на обмотці реле для нейтрального якоря: відпускання 1,4 В, спрацьовування 6,5 В;
  - значення напруги на обмотці для поляризованого якоря: переведення –3,9 В, повернення до нормального положення 3,9 В.
- можливості:
  - задавати та отримувати (вимірювати) електричну напругу на обмотці реле;
  - отримувати відомості про те, який контакт нейтрального якоря замкнутий: «тиловий» – абсолютне значення електричної напруги на обмотці не перевищує значення електричної напруги відпускання; «фронтовий» – абсолютне значення напруги на обмотці не менше за значення електричної напруги спрацьовування;
  - отримувати відомості про те, який контакт поляризованого якоря замкнутий: «переведений» – напруга на обмотці не перевищує напруги перекидання; «нормальний» – напруга на обмотці не менша напруги переведення до нормального положення.

Комбіноване реле містить нейтральний і поляризований якорі. Тому в даній ЛР єрархію створено з використанням інтерфейсів (рис. 4.2).

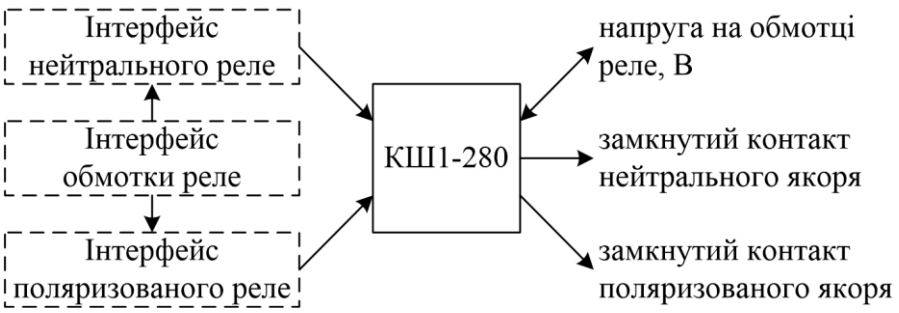

Рис. 4.2. Структура єрархії моделі реле

Постановка задачі

Розробити додаток Windows Forms, який:

– реалізує можливості функційної моделі реле КШ1-280;
– дозволяє користувачу: задавати напругу на обмотці реле;
– виводить на екран: відомості про те, які контакти замкнуті.

Порядок виконання лабораторної роботи

1. Отримати у викладача папку з проектом ElectromagneticRelay.

2. Додати проект ElectromagneticRelay до рішення RailwayAutomatics, яке створено в ЛР № 3:

2.1. За допомогою файлового менеджера операційної системи скопіювати папку з проектом ElectromagneticRelay до папки рішення RailwayAutomatics.

2.2. Відкрити рішення RailwayAutomatics (файл з розширенням SLN).

2.3. У вікні Solution Explorer ПКМ по назві рішення, Add, Existing Project, перейти до папки ElectromagneticRelay і обрати файл проекту (файл з розширенням CSPROJ), натиснути на кнопку Open.

2.4. У VS встановити проект ElectromagneticRelay *стартовим* проектом: у вікні Solution Explorer ПКМ по назві проекту ElectromagneticRelay, Set As Startup Project.

3. Ознайомитися з формою проекту ElectromagneticRelay (рис. 4.3) в режимі конструктора. Для цього: у вікні Solution Explorer розгорнути вміст проекту, ПКМ по Form1.cs, у контекстному меню ЛКМ по View Designer; почергово ЛКМ по кожному компоненту на формі, під час цього у вікні Properties, на вкладці Properties () переглядати значення властивості Name.

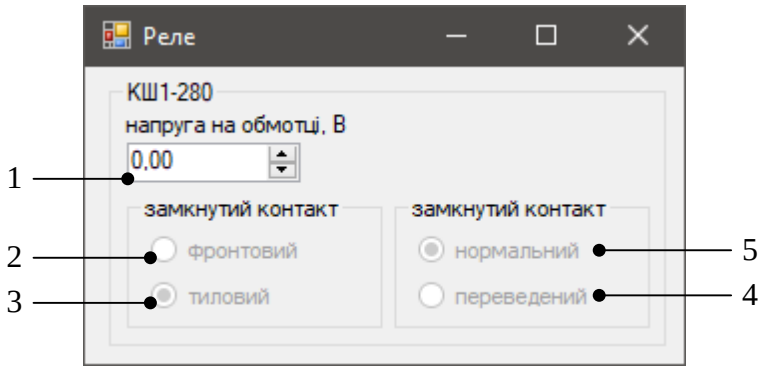

Рис. 4.3. Вікно проекту ElectromagneticRelay з елементами керування:
1 – NumericUpDown; *2–5* – RadioButton

4. До Звіту занести:

– ескіз (або скріншот) форми (подібно до рис. 4.3);

– перелік відповідностей «елемент керування на ескізі (скріншоті) форми» – «властивість Name цього елемента» (подібно до рис. 4.3).

Примітки: на рис. 4.3 подано перелік відповідностей «елемент керування» – «ім'я *класу* цього елемента», а не «*властивість* Name цього елемента».

5. До проекту ElectromagneticRelay додати заготовки інтерфейсів в окремих файлах:

– IRelayCoil – інтерфейс, який представляє напругу на обмотці реле.
– IRelayNeutral – інтерфейс, який представляє нейтральне реле. Задати в якості базового інтерфейсу інтерфейс IRelayCoil.
– IRelayPolarized – інтерфейс, який представляє поляризоване реле. Задати в якості базового інтерфейсу інтерфейс IRelayCoil.

Для цього для кожного інтерфейсу: у вікні Solution Explorer ПКМ по назві проекту, Add, New Item (якщо запропоновано опцію Show All Templates, то скористатись нею), ЛКМ по Interface, задати в текстовому полі Name назву файлу, що відповідає назві інтерфейсу, натиснути на кнопку Add.

6. До проекту ElectromagneticRelay додати заготовку класу RelayKSh1_280 (моделює роботу реле КШ1-280) в окремому файлі. Задати в якості інтерфейсів, які реалізує даний клас, інтерфейси IRelayNeutral та IRelayPolarized.

Для цього: у вікні Solution Explorer ПКМ по назві проекту, Add, New Item (якщо запропоновано опцію Show All Templates, то скористатись нею), ЛКМ по Class, задати в текстовому полі Name назву файлу RelayKSh1_280.cs, натиснути на кнопку Add. Доповнити декларацію класу інтерфейсами IRelayNeutral та IRelayPolarized, які реалізуватиме клас.

7. У файлі з інтерфейсом IRelayNeutral доповнити тіло простору імен (ElectromagneticRelay) переліком можливих замкнутих контактів нейтрального якоря.

```
enum ClosedContactNeutral {Тиловий, Фронтовий};
```

8. У файлі з інтерфейсом IRelayPolarized доповнити тіло простору імен (ElectromagneticRelay) переліком можливих замкнутих контактів поляризованого якоря.

```
enum ClosedContactPolarized {Нормальний, Переведений};
```

9. Отримати у викладача файл CodeFile4.cs. Скопіювати його до папки проекту ElectromagneticRelay та відкрити.

10. З файлу CodeFile4.cs доповнити тіло інтерфейсу IRelayCoil елементом CoilVoltage – властивість типу double. *Призначення*: повертати та задавати величину напруги на обмотці реле, В.

11. З файлу CodeFile4.cs доповнити тіло інтерфейсу IRelayNeutral елементом ClosedContact – властивість типу ClosedContactNeutral. *Призначення*: повертати відомості про те, який контакт замкнутий.

12. З файлу `CodeFile4.cs` доповнити тіло інтерфейсу `IRelayPolarized` елементом `ClosedContact` – властивість типу `ClosedContactPolarized`. *Призначення*: повертати відомості про те, який контакт замкнутий.

13. З файлу `CodeFile4.cs` доповнити тіло класу `RelayKSh1_280` такими елементами:

- `dropOutVoltageNeutral`, `pickUpVoltageNeutral` – константи типу `double`. *Призначення*: зберігати значення електричної напруги (В) відпускання та спрацьовування нейтрального якоря, відповідно.
- `flipVoltagePolarized`, `normalVoltagePolarized` – константи типу `double`. *Призначення*: зберігати значення електричної напруги (В) перекидання та повернення до нормального положення поляризованого якоря, відповідно.
- `coilVoltage`, `CoilVoltage` – поле та властивість типу `double`, відповідно. *Призначення*: повертати та задавати величину електричної напруги на обмотці реле, В.
- `ClosedContactNeutral` – властивість типу `ClosedContactNeutral`. *Призначення*: повертати інформацію про те, який контакт нейтрального якоря замкнутий. Використовується разом з властивістю `IRelayNeutral.ClosedContact` та полем `closedContactNeutral`.
- `ClosedContactPolarized` – властивість типу `ClosedContactPolarized`. *Призначення*: повертати інформацію про те, який контакт поляризованого якоря замкнутий. Використовується разом з властивістю `IRelayPolarized.ClosedContact` та полем `closedContactPolarized`.

14. Закрити файл `CodeFile4.cs`.

15. Відкрити `Form1.cs` в режимі редактора коду.

Клас `Form1` містить обробник події `nmrcUpDnKSh1_280_ValueChanged`. Цей обробник викликається у разі зміни значення властивості `Value` в `nmrcUpDnKSh1_280Voltage`.

16. Доповнити тіло класу `Form1` полем `kSh1_280` типу `RelayKSh1_280`, ініціювати його викликавши конструктор типу `RelayKSh1_280` за промовчанням.

17. Скомпілювати та запустити проект в режимі налагодження.

18. Дослідити функціонування додатку: побудувати графік релейної характеристики (залежність стану контакту від значення електричної напруги на обмотці реле, див. рис. 4.4) реле КШ1-280 для нейтрального та поляризованого якоря. Результат – до Звіту.

Примітки: замкнутим контактам нейтрального якоря «Тиловий» та «Фронтовий» ставити у відповідність числа 0 та 1, відповідно. Замкнутим контактам поляризованого якоря «Нормальний» та «Переведений» ставити у відповідність числа 1 та –1, відповідно.

19. У Звіті привести алгоритм функціонування методу `nmrcUpDnKSh1_280_ValueChanged` з коментарями.

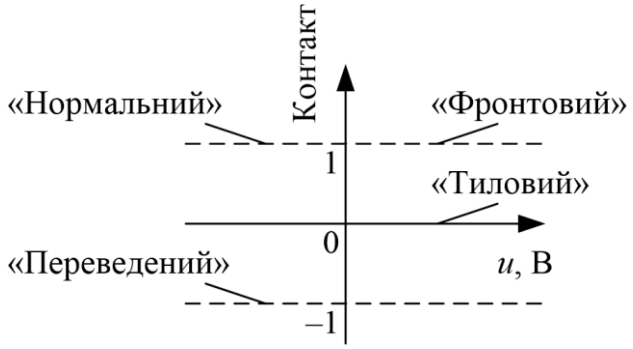

Рис. 4.4. Шаблон графіку для побудови релейної характеристики

20. Зберегти проект. Остаточний текст програми – до Звіту. Закрити поточне рішення.

Зміст звіту

1. Номер, тема, мета ЛР.
2. Пункти порядку виконання лабораторної роботи: 4, 18–20.
3. Рішення в папці `RailwayAutomatics` з проектом `ElectromagneticRelay`.
4. Висновки.

Контрольні запитання

1. Інтерфейс. Призначення та спосіб декларування.
2. Абстрактний клас. Призначення та спосіб декларування.
3. Порівняння інтерфейсів та абстрактних класів.
4. Множинне успадковування.
5. Функційна модель комбінованого реле КШ1-280.

ЛАБОРАТОРНА РОБОТА № 5
ДОСЛІДЖЕННЯ ДЕЛЕГУВАННЯ НА ПРИКЛАДІ МОДЕЛЮВАННЯ ВЗАЄМОДІЇ РЕЙКОВОГО КОЛА З КОЛІЙНИМ РЕЛЕ

Мета роботи: дослідити механізм делегування на прикладі моделювання взаємодії рейкового кола з колійним реле; отримати поняття про використання делегатів та подій; сформувати уявлення про взаємодію рейкового кола з колійним реле.

Основні теоретичні відомості

Серед типових вимог до ООП, зокрема, є *пересилання повідомлень*. *Повідомлення* (message) у мовах програмування – запит до об'єкта для виконання однієї з його операцій [1]. Пересилання повідомлень дозволяє забезпечити комунікацію між об'єктами. Це може бути реалізовано шляхом *делегування* (delegation), тобто засобу, що дає змогу об'єктові призначити *інший* об'єкт для обслуговування повідомлення [1].

Події (events) в C# є способом для об'єкта поширити серед усіх «зацікавлених» («підписаних» на подію) компонентів повідомлення про те, що щось відбулось (власне, подія) [7]. Будь-які інші компоненти можуть «підписатись» на подію, та бути сповіщеними, коли вона виникне. Після отримання цього сповіщення, компоненти можуть виконати певну дію, наприклад, викликати якийсь метод. Таким чином у C# досягається делегування, тобто виклик одного методу (або декількох) після виникнення певної події.

Делегат (delegate) в C# – тип, який представляє посилання на методи з певним списком параметрів та типом, який повертається [7]. Після створення екземпляра делегата можна асоціювати його з будь-яким методом зі *сумісною* сигнатурою та типом значення, що повертається [7]. Обробники подій є методами, які викликано через делегати.

У даній ЛР досліджується механізм делегування на прикладі моделювання взаємодії рейкового кола з колійним реле з використанням додатку `Windows Forms`.

Об'єкт моделювання

Відповідно до рис. 3.1, з РК на колійне реле надходить сигнальний струм (корисний сигнал). В попередніх ЛР змодельовано функціонування РК та електромагнітного реле як незалежних елементів. В даній ЛР моделюється узгоджена робота імпульсного РК та колійного реле НМШ1-500.

Функційна модель реле НМШ1-500:

- містить значення електричної напруги на обмотці реле: відпускання 2,5 В, спрацьовування 7,5 В;
- можливості
  - задавати та отримувати (вимірювати) електричну напругу на обмотці реле;
  - отримувати відомості про те, який контакт замкнутий: «тиловий» – абсолютне значення електричної напруги на обмотці не перевищує напруги відпускання; «фронтовий» – абсолютне значення електричної напруги на обмотці не менше напруги спрацьовування;
  - реагувати на зміну сигналу на вході приймача РК шляхом подання відповідної електричної напруги на обмотку.

Доповнення функційної моделі імпульсного РК. Можливості: генерувати подію у разі зміни сигналу на вході приймача РК. В параметрах події передається новий сигнал на вході приймача.

Постановка задачі

Розробити додаток `Windows Forms`, який узгоджує роботу імпульсного РК та колійного реле НМШ1-500:

- дозволяє користувачу: задавати ті ж значення, що й в проекті `RailwayTrackCircuit`;
- виводить на екран:
  - ту ж інформацію, що й в проекті `RailwayTrackCircuit`;
  - додатково – виводить інформацію про те, який саме контакт колійного реле замкнуто.

Порядок виконання лабораторної роботи

1. Відкрити рішення `RailwayAutomatics`, яке створено в ЛР № 3 та змінено в ЛР № 4. Встановити проект `RailwayTrackCircuit` стартовим проектом: у вікні `Solution Explorer` ПКМ по назві проекту `RailwayTrackCircuit`, `Set As Startup Project`.

2. Відкрити форму проекту `RailwayTrackCircuit` в режимі конструктора. Додати до частини форми (рис. 3.3) «Результати» елементи керування:

- `Label` з текстом (властивістю `Text`) «Замкнутий контакт реле».
- `TextBox` з властивістю `Name`, що дорівнює `txtBxClosedContact` – представлення того, який контакт замкнуто.

3. До проекту `ElectromagneticRelay` додати заготовку класу `RelayNMSh1_500` (моделює роботу реле НМШ1-500) в окремому файлі. Задати інтерфейс `IRelayNeutral` як інтерфейс, який даний клас реалізує.

Для цього: у вікні `Solution Explorer` ПКМ по назві проекту, `Add`, `New Item` (якщо запропоновано опцію `Show All Templates`, то скористатись нею), ЛКМ по `Class`, задати в текстовому полі `Name` назву файлу `RelayNMSh1_500.cs`, натиснути на кнопку `Add`. Доповнити декларацію класу інтерфейсом `IRelayNeutral`, який реалізуватиме клас.

4. Отримати у викладача файл `CodeFile5.cs`. Скопіювати його до папки рішення `RailwayAutomatics` та відкрити.

5. З файлу `CodeFile5.cs` доповнити тіло класу `RelayNMSh1_500` такими елементами:

- `dropOutVoltage`, `pickUpVoltage` – константи типу `double`. *Призначення*: зберігати значення електричної напруги (В) відпускання та спрацьовування нейтрального якоря, відповідно.
- `coilVoltage`, `CoilVoltage` – поле та властивість типу `double`, відповідно. *Призначення*: повертати або задавати величину електричної напруги на обмотці реле, В.

– `ClosedContact` – властивість типу `ClosedContactNeutral`. *Призначення*: повертати інформацію про те, який контакт нейтрального якоря замкнутий. Використовується разом з полем `closedContact`.
– `UpdateCoilVoltage(bool)` – метод – обробник події «зміна сигналу на вході приймача», тип значення, що повертається – `void`. *Призначення*: оновлювати значення напруги на обмотці реле відповідно до таких умов – якщо на вході приймача РК сигнал змінився на
  ◦ «Присутній», то задати на обмотці реле напругу 1.2×`pickUpVoltageNeutral`;
  ◦ «Відсутній», то задати 0.01×`dropOutVoltageNeutral`.

6. З файлу `CodeFile5.cs` доповнити тіло класу `TrackCircuitDCimpulse` такими елементами:
– `SignalAvailableAtReceiverChangedHandler(bool)` – делегат. *Призначення*: декларувати подію «зміна сигналу на вході приймача».
– `SignalAvailableAtReceiverChanged` – поле типу `SignalAvailableAtReceiverChangedHandler`. *Призначення*: декларувати подію «зміна сигналу на вході приймача».
– `OnSignalAvailableAtReceiverChanged(bool)` – метод, тип значення, що повертається – `void`. *Призначення*: генерувати подію «зміна сигналу на вході приймача». Параметр `isSignalAvailableAtReceiver` приймає інформацію чи присутній сигнал на вході приймача.

7. З файлу `CodeFile5.cs` замінити тіло методу доступу `set` властивості `IsSignalAvailableAtReceiver` класу `TrackCircuitDCimpulse` для генерації події «зміна сигналу на вході приймача».

8. Закрити файл `CodeFile5.cs`.

9. Скопіювати файли `RelayNMSh1_500.cs`, `IRelayCoil.cs` та `IRelayNeutral.cs` з проекту `ElectromagneticRelay` до проекту `RailwayTrackCircuit` використовуючи вікно `Solution Explorer`: виділити вказані файли в області елементів проекту `ElectromagneticRelay`, затиснути на них ЛКМ, перетягнути до області елементів проекту `RailwayTrackCircuit`, відпустити ЛКМ.

10. Дозволити у файлі `Form1.cs` проекту `RailwayTrackCircuit` використовувати типи простору імен `ElectromagneticRelay` за допомогою директиви `using`.

11. Доповнити тіло класу `Form1` проекту `RailwayTrackCircuit` полем `NMSh1_500` типу `RelayNMSh1_500`. Ініціювати його викликавши конструктор типу `RelayNMSh1_500` за промовчанням.

12. Доповнити тіло методу `Form1_Load` (клас `Form1` проекту `RailwayTrackCircuit`) кодом, після виконання якого буде здійснено підписання на подію та задано значення `true` властивості `ReadOnly` елемента `txtBxClosedContact`:

```
trackCircuitDCImpulse.SignalAvailableAtReceiverChanged
    += NMSh1_500.UpdateCoilVoltage;
txtBxClosedContact.ReadOnly = true;
```

13. Доповнити тіло методу `OutputOfResult` (клас `Form1` проекту `RailwayTrackCircuit`) кодом, після виконання якого до `txtBxClosedContact` буде виведено найменування замкнутого контакту:

```
txtBxClosedContact.Text =
    NMSh1_500.ClosedContact.ToString();
```

14. Скомпілювати та запустити проект в режимі налагодження.

15. Дослідити функціонування додатку. Заповнити таблицю виду табл. 3.1 (до Звіту). Порожні комірки заповнити найменуванням замкнутого контакту колійного реле.

16. Порівняння даних таблиць у поточній ЛР та ЛР № 3 – до Звіту.

17. Зберегти проект. Остаточний текст програми – до Звіту. Закрити поточне рішення.

Зміст звіту

1. Номер, тема, мета ЛР.
2. Пункти порядку виконання лабораторної роботи: 15–17.
3. Рішення в папці `RailwayAutomatics` з проектами `RailwayTrackCircuit` та `ElectromagneticRelay`.
4. Висновки.

Контрольні запитання

1. Делегат. Призначення та спосіб декларування.
2. Делегат. Групова адресація та групові (multicast) делегати.
3. Подія. Призначення та спосіб декларування.
4. Подія. Взаємодія видавця події та підписника на подію.
5. Подія. Передача параметрів у подію.

ЛАБОРАТОРНА РОБОТА № 6
ДОСЛІДЖЕННЯ ЗВЕРТАННЯ ДО ФАЙЛУ НА ПРИКЛАДІ МОДЕЛЮВАННЯ ЖУРНАЛУ ОГЛЯДУ ФОРМИ ДУ-46

Мета роботи: дослідити читання з та запис до текстового файлу на прикладі моделювання журналу огляду форми ДУ-46; сформувати уявлення про журнали обліку робіт з технічного обслуговування пристроїв сигналізації, централізації та блокування.

Основні теоретичні відомості

Для того, щоб забезпечити можливість відстежити стан системи (процесу) у часі, як правило, використовується *реєстрування* (log out) або запис до журналу обліку (journalize). Це дозволяє виявити відхилення (порушення) у функціонуванні системи (процесу) та потенційні області для покращення її продуктивності. Реєстрування, як правило, виконується до файлу, що розміщено на матеріальному носії. *Файл* (file) – пойменований набір записів, що зберігаються чи обробляються як єдиний блок (модуль) [1]. У C# для читання з та запису до файлів чи інших джерел використовуються *потоки* даних. *Потік* (stream) – послідовність байт, яку можна використовувати для запису до та читання із запам'ятовуючого пристрою [7]. У файла є постійне місце зберігання (на відміну від потоку).

Журнал обліку, як правило, зберігає багато порівняно невеликих записів. У C# для представлення порівняно невеликих сукупностей даних зручно застосовувати *структуру*. *Структура* C# (`struct`) – значимий тип (value type), який подібний на клас (може містити елементи дані та функціональні елементи) [7]. Оскільки на відміну від класу структура є значимим типом, а не типом посилання, то структура не вимагає виділення пам'яті в динамічно розподіленій ділянці пам'яті – *купі* (heap), що збільшує швидкодію програми.

У даній ЛР досліджується читання з та запис до текстового файлу на прикладі моделювання журналу огляду форми ДУ-46. При цьому використано елементи, що подано нижче.

`OpenFileDialog` – клас (простір імен `System.Windows.Forms`), який відображає стандартне діалогове вікно, що запитує користувача відкрити файл [7].

`System.IO` – простір імен, який містить типи, що дозволяють читати та записувати до файлів та потоків даних, а також типи, що надають базову підтримку файлів та папок [7].

- `Stream` – клас (простір імен `System.IO`), надає універсальне (generic) представлення послідовності байтів [7].
- `FileStream` – клас (простір імен `System.IO`), надає `Stream` для файлу, підтримуючи синхронні та асинхронні операції читання та запису [7].
- `StreamReader` – клас (простір імен `System.IO`), реалізує `TextReader` (представляє читача, який може читати послідовні серії символів), що читає символи з потоку байтів у певному кодуванні [7].
- `StreamWriter` – клас (простір імен `System.IO`), реалізує `TextWriter` (представляє те, що може записувати послідовні серії символів) для запису символів до потоку в певному кодуванні [7].

`using` – оператор, який забезпечує коректне використання об'єктів `IDisposable` [7]. `IDisposable` – інтерфейс, який надає механізм для звільнення некерованих ресурсів [7].

Об’єкт моделювання

Одним з обов’язків працівників служби сигналізації, централізації та блокування (СЦБ) є забезпечення робіт з технічного обслуговування (ТО) пристроїв. На даний момент на залізницях діє планово-попереджувальний вид ТО. Для кожної дільниці старшого електромеханіка СЦБ (бригади) складаються та затверджуються *чотиритижневий* та *річний* плани-графіки ТО пристроїв [5]. Факт виконання робіт, передбачених планами-графіками, виконавці підтверджують підписом у відповідній графі оперативного плану, а результати перевірок реєструють в *журналах* (*картках*) за формою, що вказана в переліку робіт з ТО.

У даній ЛР моделюється внесення записів (у спрощеному вигляді) до журналу огляду колій, стрілочних переводів, пристроїв СЦБ, зв’язку і контактної мережі ДУ-46 [5], шляхом доповнення існуючого текстового файлу.

Функційна модель запису до журналу огляду:

- містить:
  - дату та час внесення запису;
  - ім’я працівника, який вносить запис;
  - текстове повідомлення.
- можливості: доповнювати текстовий файл рядковим представленням запису.

Функційна модель журналу огляду:

- містить: список записів.
- можливості:
  - читати з текстового файлу до списку записів журналу;
  - перетворювати записи журналу на рядок.

Постановка задачі

Розробити додаток `Windows Forms`, який відповідно до описаного функціоналу моделює внесення та читання записів з журналу огляду:

- дозволяє користувачу: доповнювати текстовий файл записами;
- виводить на екран: вміст текстового файлу із записами.

Порядок виконання лабораторної роботи

1. Отримати у викладача папку з рішенням `InspectionLog` та проектом `InspectionLog`.

2. Ознайомитися з формою проекту `InspectionLog` (рис. 6.1) в режимі конструктора. Для цього: відкрити проект `InspectionLog` (розширення файлу `CSPROJ`) або відповідне рішення (розширення файлу `SLN`) у VS; у вікні `Solution Explorer` ПКМ по `Form1.cs`, у контекстному меню ЛКМ по `View Designer`; почергово ЛКМ по кожному компоненту на формі, під час цього у вікні `Properties`, на вкладці `Properties` () переглядати значення властивості `Name`.

3. Ескіз (або скріншот) форми – до Звіту.

4. До проекту `InspectionLog` додати заготовку *структури* `EntryDU46` (моделює запис в журналі огляду) в окремому файлі.

Для цього: у вікні `Solution Explorer` ПКМ по назві проекту, `Add`, `New Item` (якщо запропоновано опцію `Show All Templates`, то скористатись нею), ЛКМ по `Struct` (у разі відсутності – по `Class`), задати в текстовому полі `Name` назву файлу `EntryDU46.cs`, натиснути на кнопку `Add`. Якщо додано код декларації *класу*, то замінити ключове слово `class` на `struct`.

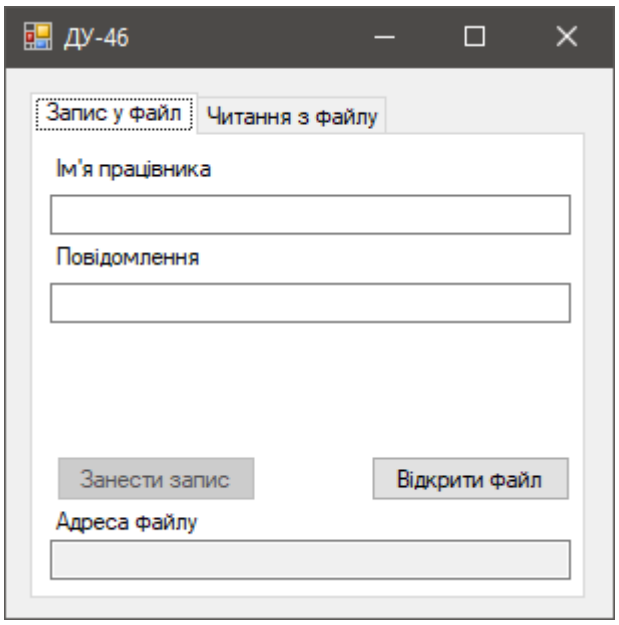

Рис. 6.1. Вікно проекту `InspectionLog`

5. До проекту `InspectionLog` додати заготовку класу `LogDU46` (моделює журнал огляду) в окремому файлі.

6. Доповнити тіло структури `EntryDU46` такими елементами:

– `TimeStamp` – автоматично реалізована властивість (`public`) типу `DateTime`. *Призначення*: повертати та задавати дату та час запису до журналу.
– `Name` – автоматично реалізована властивість (`public`) типу `string`. *Призначення*: повертати та задавати ім’я працівника, який вносить запис.
– `Message` – автоматично реалізована властивість (`public`) типу `string`. *Призначення*: повертати та задавати текст повідомлення у записі.

7. Отримати у викладача файл `CodeFile6.cs`. Скопіювати його до папки проекту `InspectionLog` та відкрити.

8. З файлу `CodeFile6.cs` доповнити тіло структури `EntryDU46` такими елементами:

– `AddToFile(EntryDU46, Stream)` – метод (`static`), тип значення, що повертається – `void`. *Призначення*: доповнювати текстовий файл (параметр `stream`) рядковим представленням запису (параметр `entry`) в журналі огляду.
– `EntryDU46(DateTime, string, string)` – конструктор. *Призначення*: створити запис з датою, іменем працівника та повідомленням.

9. З файлу `CodeFile6.cs` доповнити тіло класу `LogDU46` такими елементами:
– `EntryList` – автоматично реалізована властивість (`public`) типу `List<EntryDU46>`. *Призначення*: повертати або задавати список записів в журналі.
– `ReadFileToLog(List<EntryDU46>, Stream)` – метод (`static`). *Призначення*: читати вміст текстового файлу (параметр `stream`) та записувати його до журналу (параметр `entryList`).
– `ToString()` – перевизначення методу `ToString()`. *Призначення*: перетворювати журнал записів на рядкове представлення.
– `LogDU46()` та `LogDU46(EntryDU46)` – конструктори.

10. Закрити файл `CodeFile6.cs`.

11. На форму в режимі конструктора додати елемент керування `OpenFileDialog`. Задати його властивості `Name` значення `openFileDialog`.

12. Відкрити `Form1.cs` в режимі редактора коду. Дозволити використання типів простору імен `System.IO`. Доповнити тіло класу `Form1` приватними полями типу `string`:
– `fNameWrite` – ім’я файлу, до якого буде виконано запис.
– `fNameRead` – ім’я файлу, з якого буде виконано читання.

13. У файловому менеджері операційної системи перейти до папки з `InspectionLog.exe` (за промовчанням – у папці проекту …`\bin\Debug`).

14. За допомогою будь-якого текстового редактору створити порожній текстовий файл `ДУ-46.txt`. Розмір файлу – до Звіту. Закрити текстовий редактор.

15. Скомпілювати та запустити проект в режимі налагодження.

16. Дослідити зміну розміру файлу після доповнення новим записом:

16.1. В додатку `InspectionLog` перейти на вкладку «Запис у файл», натиснути кнопку «Відкрити файл», в діалозі «Відкриття файлу» обрати файл `ДУ-46.txt`, натиснути кнопку «Відкрити».

16.2. До текстових полів «Ім’я працівника» та «Повідомлення» занести довільний текст (наприклад, довільна літера), натиснути кнопку «Занести запис». Розмір файлу (у файловому менеджері операційної системи) – до Звіту.

16.3. Повторити п. 16.2 (до Звіту).

17. Дослідити вміст файлу:

17.1. В додатку `InspectionLog` перейти на вкладку «Читання з файлу», натиснути кнопку «Читати з файлу», в діалозі «Відкриття файлу» обрати файл `ДУ-46.txt`, натиснути кнопку «Відкрити».

17.2. Виконати п. 13, відкрити файл `ДУ-46.txt` іншим текстовим редактором. Вміст файлу – до Звіту.

17.3. Порівняти відображення вмісту файлів у вікні додатку `InspectionLog` (п. 17.1) та текстовому редакторі (п. 17.2). Результат – до Звіту.

17.4. Закрити вікно іншого текстового редактору.

18. За допомогою додатку `InspectionLog` аналогічно п. 16.2 (вкладка «Запис у файл») доповнити файл `ДУ-46.txt` новим записом:

- Текстове поле «Ім’я працівника» – вказати довільне ім’я.
- Текстове поле «Повідомлення» – вказати без подвійних лапок «17.06.2024 виконуватиметься заміна лінзового комплекту світлофору».

19. Виконати п. 13, створити копію файлу `ДУ-46.txt` з назвою `ДУ-46 (2).txt`.

20. Дослідити результат зміни вмісту файлу:

20.1. У будь-якому текстовому редакторі відкрити файл `ДУ-46 (2).txt`. В рядку з датою другого запису в журналі додати слово «дата», зберегти зміни та закрити файл.

20.2. Виконати п. 17.1 з такими змінами: в діалозі «Відкриття файлу» обрати файл `ДУ-46 (2).txt`. Реакцію програми – до Звіту.

21. Зберегти проект. Остаточний текст програми – до Звіту. Закрити поточне рішення.

Зміст звіту

1. Номер, тема, мета ЛР.
2. Пункти порядку виконання лабораторної роботи: 3, 14, 16.2, 16.3, 17.2, 17.3, 20.2, 21.
3. Рішення в папці `InspectionLog` з проектом `InspectionLog`.
4. Висновки.

Контрольні запитання

1. Файл та файлові потоки.
2. Клас `Stream`.
3. Клас `FileStream`.
4. Класи `StreamWriter`, `StreamReader`.
5. Оператор `using`.

ЛАБОРАТОРНА РОБОТА № 7
ДОСЛІДЖЕННЯ РЕАЛІЗАЦІЇ ЗАПИТІВ НА ПРИКЛАДІ МОДЕЛЮВАННЯ ОТРИМАННЯ ВИТЯГУ З ФАЙЛУ АРМ ШН СИСТЕМИ МІКРОПРОЦЕСОРНОЇ ЦЕНТРАЛІЗАЦІЇ

Мета роботи: дослідити спосіб реалізації запитів на прикладі моделювання отримання витягу з файлу АРМ ШН системи мікропроцесорної централізації; отримати поняття про серіалізацію та десеріалізацію об'єктів; отримати уявлення про запити LINQ to Objects.

Основні теоретичні відомості

Під час розроблення програмного забезпечення інколи потрібно перетворити стан об'єкта на форму, яку може бути збережено чи переслано [7]. В .NET такий процес називається *серіалізація* (serialization). Відповідно *десеріалізація* – це процес протилежний серіалізації – процес перетворення потоку даних на об'єкт [7]. За допомогою серіалізації, наприклад, можна створити незалежну копію об'єкта типу посилання, тобто виконати глибоке копіювання. Серіалізація також може бути застосована до збереження об'єктів у записах баз даних.

Однією з поширених операцій з базами даних (в загальному випадку – колекціями) є отримання даних відповідно певному критерію. Як правило, для цього використовують *запити* (queries) – запити отримання даних безпосередньо чи з бази даних на основі вказаних умов [1]. До C# безпосередньо інтегровано можливості запитів, на основі яких створено множину технологій, що називається *мовно-інтегровані запити* (Language-Integrated Query – LINQ) [7]. LINQ дозволяє не витрачати час на вивчення різних мов запиту залежно від типу джерела даних (бази даних SQL, документи XML тощо). Поширеним способом написання запитів LINQ є вираз запиту (query expression) з використанням синтаксису запиту для виконання фільтрування, впорядкування та групування над джерелами даних [7].

Першим кроком у запиті LINQ є визначення джерела даних. Для цього використовується твердження (clause) `from` (у виразі запиту), що визначає:

- джерело даних, відносно яких виконуватиметься запит;
- локальну *змінну діапазону* (range variable) – представляє *кожен* елемент у вхідній послідовності.

Для того, щоб визначити які елементи з джерела даних потрібно повернути, у виразі запиту використовується твердження `where` [7]. Це твердження застосовує умову типу `bool` (або *предикат* – функцію, що повертає значення типу `bool`) до кожного елемента джерела і повертає ті елементи, для яких вказана умова є істинною. Для того, щоб впорядкувати послідовність (повернуто у виразі запиту) за зростанням чи спаданням, у виразі запиту використовується твердження `orderby` [7]. Для того, щоб вказати тип значень, які

буде створено після виконання запиту, у виразі запиту використовується твердження `select` [7]. Вираз запиту повинен бути завершеним або твердженням `select`, або твердженням `group`.

Об’єкт моделювання

Для забезпечення безпеки руху поїздів та підвищення пропускної здатності на залізниці використовуються різні системи залізничної автоматики. На станціях досить поширене використання систем електричної централізації. *Електрична централізація* (ЕЦ) стрілок і сигналів – система, яка забезпечує схемні взаємозалежності між стрілками і сигналами, можливість їх управління та контролю [3]. ЕЦ, яка побудована з використанням мікропроцесорних технологій, називається *мікропроцесорна централізація* (МПЦ). Результат функціонування МПЦ (діагностування стану об’єктів управління та контролю) може бути записано до файлу. Доступ до файлу забезпечується з автоматизованого робочого місця (АРМ) електромеханіка (ШН).

В даній ЛР моделюється: внесення спрощених записів до файлу АРМ ШН системи МПЦ, шляхом серіалізації кожного запису; отримання вибірки з файлу за певним запитом, шляхом використання десеріалізації та запитів LINQ.

Функційна модель запису в файлі АРМ ШН системи МПЦ:

- містить:
  - дату та час внесення запису;
  - тип та назву об’єкту діагностування;
  - назву та значення параметру об’єкта діагностування.
- можливості: перетворювати запис на рядок.

Функційна модель журналу в АРМ ШН МПЦ:

- містить: список записів;
- можливості: серіалізувати до та десеріалізувати з двійкового файлу.

Постановка задачі

Розробити консольний додаток, який відповідно до описаного функціоналу моделює внесення та читання записів з файлу АРМ ШН МПЦ, формує вибірку даних відповідно до заданого параметру.

Порядок виконання лабораторної роботи

1. Отримати у викладача папку з рішенням `MicroprocessorInterlockingSystem` та проектом `MPC`. Відкрити рішення `MicroprocessorInterlockingSystem` (файл з розширенням `SLN`).

2. До проекту `MPC` додати заготовки класів в окремих файлах:

- `EntryArmShnMpc` – клас, який представляє запис АРМ ШН МПЦ. Задати цьому класу атрибут `Serializable`.
- `LogArmShnMpc` – клас, який моделює журнал записів АРМ ШН МПЦ.

Для цього для кожного класу: у вікні Solution Explorer ПКМ по назві проекту, Add, New Item (якщо запропоновано опцію Show All Templates, то скористатись нею), ЛКМ по Class, задати в текстовому полі Name назву файлу, що відповідає назві класу, натиснути на кнопку Add.

3. У файлі, який містить клас LogArmShnMpc, за допомогою директиви using дозволити використовувати типи просторів імен System.IO та System.Runtime.Serialization.Formatters.Binary.

4. Доповнити тіло класу EntryArmShnMpc такими елементами:

– TimeStamp – автоматично реалізована властивість (public) типу DateTime. *Призначення*: повертати або задавати дату та час.
– ItemType – автоматично реалізована властивість (public) типу string. *Призначення*: повертати або задавати тип об'єкту діагностування.
– ItemName – автоматично реалізована властивість (public) типу string. *Призначення*: повертати або задавати назву об'єкту діагностування.
– ParameterName – автоматично реалізована властивість (public) типу string. *Призначення*: повертати або задавати назву параметру об'єкту діагностування.
– ParameterValue – автоматично реалізована властивість (public) типу double. *Призначення*: повертати або задавати значення параметру об'єкту діагностування.

5. Отримати у викладача файл CodeFile7.cs. Скопіювати його до папки проекту MPC та відкрити.

6. З файлу CodeFile7.cs доповнити тіло класу EntryArmShnMpc такими елементами:

– ToString() – перевизначення методу ToString(). *Призначення*: повертати рядок з інформацією у поточному записі.
– EntryArmShnMpc(DateTime, string, string, string, double) – конструктор. *Призначення*: створити запис з датою (параметр timestamp), типом (параметр itemType) та назвою (параметр itemName) об'єкту діагностування, назвою (параметр parameterName) та значенням (параметр parameterValue) параметру об'єкта діагностування.

7. З файлу CodeFile7.cs доповнити тіло класу LogArmShnMpc такими елементами:

– EntryList – автоматично реалізована властивість (public) типу List<EntryArmShnMpc>. *Призначення*: повертати або задавати список записів в журналі.
– SerializeLog(string) – метод, тип значення, що повертається – void. *Призначення*: серіалізовувати список записів у журналі до файлу (параметр fName).

– `DeserializeLog(string)` – метод, тип значення, що повертається – `void`. *Призначення*: десеріалізовувати список записів у журналі з файлу (параметр `fName`).
– `LogArmShnMpc()` – конструктор.

8. Закрити файл `CodeFile7.cs`.

9. Дослідити файл із серіалізованими об'єктами:

9.1. Доповнити тіло методу `Main` (файл `Program.cs`) кодом, після виконання якого журнал `logMPC` буде серіалізовано до файлу `Журнал.dat`:

```
logMPC.SerializeLog("Журнал.dat");
```

9.2. Скомпілювати та запустити проект в режимі налагодження. Закрити вікно консольного додатку.

9.3. У файловому менеджері операційної системи перейти до папки `…\bin\Debug`, яку розміщено в папці проекту `MPC`.

9.4. У будь-якому текстовому редакторі відкрити для перегляду файл `Журнал.dat`. Опис вмісту файлу – до Звіту. Закрити текстовий редактор.

10. Прочитати файл `Журнал.dat` та вивести його вміст до консольного вікна:

10.1. Доповнити тіло методу `Main` (файл `Program.cs`) кодом, після виконання якого:

10.1.1. Журнал з файлу `Журнал.dat` буде десеріалізовано до змінної `logMpcFromFile` типу `LogArmShnMpc`:

```
LogArmShnMpc logMpcFromFile = new LogArmShnMpc();
// Десеріалізувати журнал з файлу.
try
{ logMpcFromFile.DeserializeLog("Журнал.dat"); }
catch  (Exception exc)
{   Console.WriteLine(exc.Message);
    Console.ReadLine();
    return;
}
```

10.1.2. До консольного вікна буде виведено вміст журналу, який збережено у змінній `logMpcFromFile`:

```
Console.WriteLine("З файлу (повністю):");
foreach (EntryArmShnMpc x in logMpcFromFile.EntryList)
    Console.WriteLine(x.ToString());
```

10.2. Скомпілювати та запустити проект в режимі налагодження.

10.3. Опис результату – до Звіту. Закрити консольне вікно.

11. Вивести всі елементи журналу типу «рейкове коло», в яких параметр «напруга на колійному реле, В» має значення не більше за 0,42 В. Сортувати результуючу вибірку за назвою об'єкта діагностування:

11.1. Доповнити тіло методу `Main` (файл `Program.cs`), кодом після виконання якого:

11.1.1. Буде створено відповідний запит LINQ:

```
var q = from x in logMpcFromFile.EntryList
        where ( (x.ItemType == "рейкове коло") &&
            (x.ParameterName == "напруга на колійному реле, В")
            && (x.ParameterValue <= 0.42) )
        orderby x.ItemName
        select x;
```

11.1.2. До консольного вікна буде виведено вибірку даних відповідно до попередньо створеного запиту:

```
Console.WriteLine();
Console.WriteLine("Результат виконання запиту:");
foreach (EntryArmShnMpc x in q)
    Console.WriteLine(x.ToString());
```

11.2. Скомпілювати та запустити проект в режимі налагодження.

11.3. Опис результату – до Звіту. Закрити консольне вікно.

12. Дослідити результат зміни запиту:

12.1. У запиті LINQ (п. 11.1.1) замість рядка, з яким порівнюється значення властивості `ParameterName`, задати будь-який інший рядок. Наприклад, доповнити рядок довільною літерою.

12.2. Скомпілювати та запустити проект в режимі налагодження.

12.3. Опис результату – до Звіту. Закрити консольне вікно.

12.4. Відмінити зміни в коді, внесені відповідно п. 12.1.

13. Дослідити результат зміни вмісту файлу:

13.1. Виконати п. 9.3, створити копію файлу `Журнал.dat` з назвою `Журнал2.dat`.

13.2. У методі `Main` у фрагменті коду для десеріалізації з файлу (п. 10.1.1) замінити аргумент `“Журнал.dat”` методу `DeserializeLog` на `“Журнал2.dat”`.

13.3. Скомпілювати та запустити проект в режимі налагодження.

13.4. Опис результату – до Звіту. Закрити консольне вікно.

13.5. Виконати п. 9.3, у будь-якому текстовому редакторі відкрити для редагування файл `Журнал2.dat`, видалити довільний фрагмент вмісту файлу, зберегти файл. Закрити текстовий редактор.

13.6. Скомпілювати та запустити проект в режимі налагодження.

13.7. Опис результату – до Звіту. Закрити консольне вікно.

13.8. Відмінити зміни в коді, внесені відповідно п. 13.2.

14. Зберегти проект. Остаточний текст програми – до Звіту. Закрити поточне рішення.

Зміст звіту

1. Номер, тема, мета ЛР.
2. Пункти порядку виконання лабораторної роботи: 9.4, 10.3, 11.3, 12.3, 13.4, 13.7, 14.
3. Рішення в папці `MicroprocessorInterlockingSystem` з проектом `MPC`.
4. Висновки.

Контрольні запитання

1. Серіалізація та десеріалізація.
2. Запити LINQ. Оператор запиту LINQ `from`.
3. Запити LINQ. Оператор запиту LINQ `where`.
4. Запити LINQ. Оператор запиту LINQ `orderby`.
5. Запити LINQ. Оператор запиту LINQ `select`.

ЛАБОРАТОРНА РОБОТА № 8
ДОСЛІДЖЕННЯ ПОТОКІВ ВИКОНАННЯ НА ПРИКЛАДІ МОДЕЛЮВАННЯ ПРОЦЕСУ ПЕРЕВЕДЕННЯ СТРІЛОК ЗА МАРШРУТОМ

Мета роботи: дослідити особливості роботи з потоками виконання на прикладі моделювання процесу переведення стрілок за маршрутом; сформувати уявлення про процес переведення стрілок за маршрутом.

Основні теоретичні відомості

Серед способів підвищити продуктивність виконання програми та надати графічному інтерфейсу можливість відповідати на дії користувача під час виконання якоїсь задачі є багатопотоковість (multithreading). *Потік* (thread) – процес у рамках іншого процесу, який застосовує ресурси останнього процесу [1]. *Потік* є частиною виконуваного коду програми. За промовчанням програма .NET запускається з одним (первинним – primary) потоком. У разі потреби програма може створити додаткові (робочі – worker) потоки, щоб виконати код паралельно чи конкуренто з первинним потоком [7].

Деякі *недоліки* багатопотоковості: складність програмної реалізації через необхідність керувати взаємодією потоків, у т.ч. забезпечити коректний доступ до спільних ресурсів; складність налагодження порівняно з однопотоковими додатками через можливу наявність помилок, виникнення яких залежить від часу та певної ситуації під час взаємодії декількох потоків.

Сучасні операційні системи використовують *процеси* (processes) для того, щоб розділити додатки, які виконуються [7]. Т.ч. *процес* (process) – ізольова-

ний набір ресурсів, який використовується окремим екземпляром додатку. В контексті одного процесу може бути запущено декілька потоків. Потік є базовою сутністю, для якої операційна система виділяє суму часових проміжків, у яких процесори фактично виконують програму – *процесорний час* (processor time). Потік може виконати будь-яку частину тексту програми, включно з частинами, які були щойно виконані іншим потоком.

У даній ЛР досліджуються особливості роботи з потоками виконання на прикладі моделювання процесу переведення стрілок за маршрутом. При цьому використано елементи, що подано нижче.

`System.Threading` – простір імен, надає класи та інтерфейси для багатопоточного програмування [7].

– `Thread` – клас (простір імен `System.Threading`), створює і контролює потік, задає його пріоритет та повертає статус [7].

Після того, як потік створено, він знаходиться в одному з можливих станів (відповідно переліку `ThreadState`), поки цей потік не буде знищено [7]. Початково, потік, який створено у загальномовному середовищі виконання (CLR), знаходиться у стані «незапущений» (`Unstarted`). Потік переходить з цього стану до стану «запущений» (`Running`) шляхом виклику методу `Thread.Start`. Після цього потік ніколи не може бути повернутим до стану «незапущений». Потік може перейти до стану «зупинений» (`Stopped`), наприклад, після його знищення. Також існують інші стани потоку.

Об’єкт моделювання

Станційні пристрої СЦБ, серед іншого, дозволяють або забороняють приймання поїзда на станцію чи відправлення зі станції, контролюють положення стрілок, дозволяють здійснювати переведення стрілок [3]. Для впорядкування відомостей щодо маршрутів, які можуть бути реалізовані на станції, використовується таблиця маршрутизації.

Стрілочний перевід – пристрій, що служить для переведення рухомого складу з однієї колії на іншу. *Стрілка* є частиною стрілочного переводу. *Стрілочні приводи* призначені для переведення, замикання та контролю положень (станів) вістряків стрілочного переводу: нормальне (плюсове), переведене (мінусове), проміжне (середнє), взріз. У даній ЛР прийнято, що плюсове положення стрілки відповідає положенню, при якому поїзд прямує без відхилення по стрілці, мінусове положення – з відхиленням.

Функційна модель стрілочного переводу:

– містить: стани стрілки («плюс», «мінус», «переведення»).
– можливості:
  ◦ отримувати відомості про стан стрілки;
  ◦ задавати та отримувати назву стрілки;
  ◦ переводити стрілку впродовж 2 с (якщо поточний стан стрілки не являється «переведення» та кінцевий стан відрізняється від поточного). Можливе задавання переведення в *окремому* потоці.

Постановка задачі

Розробити консольний додаток, який:

– відповідно до описаного функціоналу моделює процес переведення стрілок за маршрутом;
– виводить на екран: стан стрілок за маршрутом.

Порядок виконання лабораторної роботи

1. Створити рішення `RailwayStation` з проектом `TurnoutOperation`.

2. До проекту `TurnoutOperation` додати заготовку класу `Turnout` (моделює стрілочний перевід) в окремому файлі: у вікні `Solution Explorer` ПКМ по назві проекту, `Add`, `New Item` (якщо запропоновано опцію `Show All Templates`, то скористатись нею), ЛКМ по `Class`, задати в текстовому полі `Name` назву файлу `Turnout.cs`, натиснути на кнопку `Add`.

3. У файлі, що містить клас `Turnout`, дозволити використовувати типи простору імен `System.Threading` за допомогою директиви `using`.

4. Отримати у викладача файл `CodeFile8.cs`. Скопіювати його до папки проекту `TurnoutOperation` та відкрити.

5. З файлу `CodeFile8.cs` доповнити тіло класу `Turnout` такими елементами:

– `MovingTime` – константа типу `int`. *Призначення*: зберігати час переведення стрілки, мс.
– `TurnoutState` – перелік. *Призначення*: представляти множину можливих станів стрілки («плюс», «мінус», «переведення»).
– `state` та `State` – поле та властивість типу `TurnoutState`, відповідно. *Призначення*: зберігати та повертати стан стрілки.
– `Name` – автоматично реалізована властивість (типу `string`). *Призначення*: повертати або задавати назву стрілки.
– `MoveToPlus(bool, bool)` – метод. *Призначення*: перевести стрілку до стану «плюс». Параметри:
  ◦ `isToPlus` – параметр типу `bool`: `true` – перевести до стану «плюс», `false` – до «мінус»;
  ◦ `isInThread` – параметр типу `bool`: `true` – перевести в окремому потоці, `false` – перевести *без* створення окремого потоку.
– `StartMoveToPlus()`, `StartMoveToMinus()` – методи, тип значення, що повертається – `void`. *Призначення*: починати переведення стрілки до стану «плюс» чи «мінус», відповідно.

6. Закрити файл `CodeFile8.cs`.

7. Дослідити роботу додатку під час моделювання переведення стрілок зі стану «плюс» до стану «мінус», виконуючи переведення кожної стрілки в *окремому* потоці:

7.1. Доповнити тіло методу `Main` (файл `Program.cs`) кодом, після виконання якого буде:

7.1.1. Створено два екземпляри класу `Turnout`: `turnout1_3` та `turnout5_7`.

7.1.2. Властивостям `Name` об'єктів `turnout1_3` та `turnout5_7` присвоєно рядки "`1/3`" та "`5/7`", відповідно.

7.1.3. До консольного вікна виведено назву та стан кожного з об'єктів `turnout1_3` та `turnout5_7`:

```
Console.WriteLine(turnout1_3.Name
    + ": "
    + turnout1_3.State);
Console.WriteLine(turnout5_7.Name
    + ": "
    + turnout5_7.State);
```

7.1.4. Ініційовано переведення стрілок до стану «мінус», виконуючи переведення в *окремих* потоках:

```
turnout1_3.MoveToPlus(false, true);
turnout5_7.MoveToPlus(false, true);
```

7.1.5. Призупинено виконання програми до очікування натиснення клавіші `Enter`:

```
Console.ReadLine();
```

7.2. Скомпілювати та запустити проект в режимі налагодження.

7.3. Почекати завершення виконання додатку приблизно 5 с. Вміст консольного вікна – до Звіту. Закрити програму.

8. Дослідити роботу додатку під час моделювання переведення стрілок зі стану «плюс» до стану «мінус», виконуючи переведення стрілок послідовно (*без* використання окремих потоків для переведення кожної зі стрілок):

8.1. Змінити виклики методу `MoveToPlus` для об'єктів `turnout1_3` та `turnout5_7` (рядки коду, додані в п. 7.1.4): задати параметру `isInThread` методу `MoveToPlus` значення `false`.

8.2. Скомпілювати та запустити проект в режимі налагодження.

8.3. Виконати п. 7.3 (до Звіту).

9. Змоделювати переведення стрілок за маршрутом відповідно варіанту (дод. 1):

9.1. Закоментувати або вилучити всі рядки в тілі методу `Main` (файл `Program.cs`) окрім рядка, доданого в п. 7.1.5.

9.2. Доповнити тіло методу `Main` *вище* рядка доданого в п. 7.1.5 кодом, після виконання якого:

9.2.1. Аналогічно пунктам 7.1.1 та 7.1.2 – буде створено стрілки (екземпляри класу `Turnout`) відповідно варіанту (враховувати тільки непорожні комірки таблиці дод. 1); властивостям `Name` створених об'єктів буде присвоєно рядки з відповідними назвами стрілок.

9.2.2. Для тих стрілок в дод. 1 за варіантом, в яких початковий стан («мінус») відрізняється від початкового стану стрілок, створених в п. 9.2.1 («плюс»), аналогічно п. 8.1 буде виконано *послідовне* переведення стрілок (*без* використання окремих потоків) до відповідного стану.

9.2.3. До консольного вікна буде виведено рядок «Стан стрілок за варіантом:».

```
Console.WriteLine("Стан стрілок за варіантом:");
```

9.2.4. Аналогічно п. 7.1.3 – назву та стан кожної стрілки буде виведено до консольного вікна.

10. Дослідити роботу додатку під час моделювання переведення усіх стрілок за варіантом (дод. 1), які створено в п. 9.2.1. Стрілки, які знаходяться у стані «плюс», буде переведено до стану «мінус». Інші стрілки (які знаходяться у стані «мінус») буде переведено до стану «плюс». Переведення кожної стрілки буде виконано в *окремому* потоці:

10.1. Доповнити тіло методу `Main` (файл `Program.cs`) кодом, після виконання якого:

10.1.1. До консольного вікна буде виведено рядок «Для початку переведення всіх стрілок натисніть Enter» та додаток буде очікувати на натиснення клавіші `Enter`, щоб продовжити виконання:

```
Console.WriteLine("Для початку переведення " +
    "всіх стрілок натисніть Enter");
Console.ReadLine();
```

10.1.2. Аналогічно п. 7.1.4 – ініційовано переведення стрілок зі стану «плюс» («мінус») до стану «мінус» («плюс»), виконуючи переведення в *окремих* потоках.

Примітки:

– початковий стан стрілок – див. результат виконання п. 9.2.4.

– значення параметру `isToPlus` методу `MoveToPlus` – див. п. 5.

10.2. Скомпілювати та запустити проект в режимі налагодження.

10.3. Виміряти час, який витрачено на переведення усіх стрілок за варіантом (дод. 1):

10.3.1. Після появи у консольному вікні рядка, визначеного в п. 10.1.1: підготувати секундомір (скинути значення в нуль секунд); одночасно натиснути на клавішу `Enter` та розпочати відлік часу секундоміром.

10.3.2. Після завершення переведення усіх стрілок за варіантом (дод. 1): зупинити відлік часу секундоміра.

10.3.3. Виміряний час – до рядка «З використанням потоків» табл. 8.1 (колонка «1»).

10.4. Послідовність, в якій стрілки перебували у стані «переведення» та їх остаточний стан – до Звіту.

10.5. Закрити додаток.

10.6. Виконати пункти 10.2, 10.3, 10.4, 10.5 ще чотири рази. При цьому заповнювати колонки «2» – «5» табл. 8.1.

Таблиця 8.1

Експеримент	1	2	3	4	5	Середній час переведення, с
З використанням потоків						
Без використання потоків						

10.7. Обчислити середній час переведення стрілок у п’яти експериментах та записати до колонки «Середній час переведення, с».

11. Дослідити роботу додатку під час моделювання переведення усіх стрілок за варіантом (дод. 1), які створено в п. 9.2.1. Стрілки, які знаходяться у стані «плюс», буде переведено до стану «мінус». Інші стрілки (які знаходяться у стані «мінус») буде переведено до стану «плюс». Переведення стрілок буде виконано *послідовно* (*без* використання окремих потоків для переведення кожної зі стрілок):

11.1. Аналогічно п. 8.1 – змінити виклики методу `MoveToPlus` в рядках створених в п. 10.1.2: задати параметру `isInThread` методу `MoveToPlus` значення `false`.

11.2. Виконати пункти 10.2, 10.3, 10.4 (до Звіту), 10.5, 10.6, 10.7 з такими змінами: виміряний час заносити до рядка «Без використання потоків» табл. 8.1.

12. Заповнену табл. 8.1 – до Звіту.

13. Порівняння послідовностей, в яких стрілки перебували у стані «переведення» та їх остаточний стан, у разі переведення кожної стрілки в *окремому* потоці (п. 10) та під час *послідовного* переведення (п. 11) – до Звіту.

14. Зберегти проект. Остаточний текст програми – до Звіту. Закрити поточне рішення.

Зміст звіту

1. Номер, тема, мета ЛР.
2. Пункти порядку виконання лабораторної роботи: 7.3, 8.3, 10.4, 11.2, 12, 13, 14.
3. Рішення в папці `RailwayStation` з проектом `TurnoutOperation`.
4. Висновки.

Контрольні запитання

1. Потік (thread). Процесорний час.
2. Порівняння процесу (process) та потоку (thread).
3. Клас `Thread`. Стани потоку.
4. Клас `Thread`. Створення та запуск потоків.
5. Потік (thread). Переваги та недоліки багатопотоковості.

ДОДАТОК 1

ПОЧАТКОВИЙ СТАН СТРІЛОК ЗА ВАРІАНТАМИ ДЛЯ ЛАБОРАТОРНОЇ РОБОТИ № 8

Варіанти	Назва стрілки									
	1\3	5\7	9\11	13\15	17	19	21	23\25	27	29
1		+	+		–		+			+
2		+	+		–		+			–
3		+	+		–		–			
4		+	+		+			+		
5			–	+				+		
6			–	–		+				
7			–	–		–			–	
8	+	–			–		+			+
9	+	–			–		+			–
10	+	–			–		–			
11	+	–			+			+		
12	+	+	+	+				+		
13	+	+	+	–		+				
14	+	+	+	–		–			–	
15			–	+				+		
16			–	–		+				

Примітки:

- *знак «плюс» – стрілка знаходиться у стані «плюс»;*
- *знак «мінус» – стрілка знаходиться у стані «мінус»;*
- *порожня комірка – стрілку ігнорувати.*

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. ДСТУ ISO/IEC 2382:2017. Інформаційні технології. Словник термінів. Чинний від 2019–01–01. Вид. офіц. Київ : УкрНДНЦ, 2020. 464 с.
2. Інструкція з сигналізації на залізницях України : Наказ М-ва транспорту та зв'язку України від 23.06.2008 №747. URL: https://ips.ligazakon.net/document/FIN40297 (дата звернення: 08.04.2024).
3. Корнійчук М. П., Липовець Н. В., Шамрай Д. О. Технологія галузі і технічні засоби залізничного транспорту : підручник. Київ : Дельта, 2007. Ч. 2 : (розділи 7-14). 424 с.
4. Про затвердження Правил технічної експлуатації залізниць України : Наказ М-ва транспорту України від 20.12.1996 р. № 411 : станом на 1 січ. 2004 р. URL: https://zakon.rada.gov.ua/laws/show/z0050-97#Text (дата звернення: 08.04.2024).
5. ЦШ 0060. Інструкція з технічного обслуговування пристроїв сигналізації, централізації та блокування (СЦБ). Київ : М-во транспорту та зв'язку України, 2009. 87 с.
6. ЦД 0058. Інструкція з руху поїздів і маневрової роботи на залізницях України. Київ : М-во транспорту та зв'язку України, 2005. 325 с.
7. Microsoft Learn: Build skills that open doors in your career. URL: https://learn.microsoft.com/en-gb (date of access: 05.04.2024).

Навчально-методичне видання

Рибалка Роман Володимирович,
Маловічко Володимир Володимирович,
Маловічко Наталія Валентинівна,
Гончаров Костянтин Вікторович,

КОМП'ЮТЕРНІ ТЕХНОЛОГІЇ В СИСТЕМАХ ЗАЛІЗНИЧНОЇ АВТОМАТИКИ

Навчально-методичні рекомендації до лабораторних занять
(частина 3)

Електронне видання

Експертний висновок склав канд. техн. наук, доц. Володимир Профатилов

Зареєстровано НМВ УДУНТ (№ 732 від 10.06.2024)

В авторській редакції
Комп'ютерна верстка Р. В. Рибалка

Формат 60х84 1/16. Ум. друк. арк. 3,37. Обл.-вид. арк. 3,41.
Зам. № 53

Видавець: Український державний університет науки і технологій
вул. Лазаряна, 2, ауд. 2216, м. Дніпро, 49010.
Свідоцтво суб'єкта видавничої справи ДК № 7709 від 14.12.2022

Адреса видавця та дільниці оперативної поліграфії:
вул. Лазаряна, 2, Дніпро, 49010