

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ  
НАЦІОНАЛЬНИЙ ЛІСОТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ  
Кафедра інженерії програмного забезпечення

**КУРСОВА РОБОТА**

з об'єктно-орієнтованого програмування  
на тему: «Розроблення програмного продукту «DigitalCounter»

Студента 2-го курсу КН-22 групи  
спеціальності «Комп'ютерні науки»  
Симовича Олександра

(прізвище та ініціали)

Керівник: к.т.н., доц. Яцишин С.І.  
(посада, вчене звання, науковий ступінь, прізвище та ініціали)

Національна шкала \_\_\_\_\_

Кількість балів: \_\_\_\_\_ Оцінка: ECTS \_\_\_\_\_

Члени комісії

\_\_\_\_\_  
(підпис) (прізвище та ініціали)

\_\_\_\_\_  
(підпис) (прізвище та ініціали)

\_\_\_\_\_  
(підпис) (прізвище та ініціали)

\_\_\_\_\_  
(підпис) (прізвище та ініціали)

м. Львів - 2024 рік

# ЗАВДАННЯ НА КУРСОВУ РОБОТУ

студенту 2-го курсу групи КН-22

Симовичу Олександрю

## Варіант 15

1. Визначити клас ЦИФРОВИЙ ЛІЧИЛЬНИК. Лічильник - це змінна з обмеженим діапазоном, який скидається у початковий стан, коли її цілочисельне значення досягає визначеного максимуму.
2. Визначити конструктори (ініціалізації, копіювання, переміщення), деструктор та методи встановлення і виведення значень полів даних. Забезпечити можливість встановлення максимального і мінімального значень, зчитування поточного значення.
3. Перевантажити операцію () для встановлення значень полів даних, операцію ++ для збільшення значення лічильника на 1, -- для зменшення значення лічильника на 1, += для збільшення значення лічильника на задану величину, -= для зменшення значення лічильника на задану величину, операцію присвоєння об'єктів = та переміщення, потокові операції введення » та виведення « об'єктів.
4. Визначити похідний клас СЕКУНДОМІР з додатковими полями даних, які позначають включення та виключення секундоміра. Визначити операторні методи зчитування та виведення значення заміряного часу.
5. У межах ієрархії класів побудувати поліморфічний кластер на основі віртуального методу виведення поточного значення лічильника або часу вимкнення секундоміра. Продемонструвати механізм пізнього зв'язування.
6. Розробити клас РЕЗУЛЬТАТИ спринтерського забігу, який містить масив об'єктів класу СЕКУНДОМІР. Визначити найменший та найбільший час забігу, середнє значення часу забігу, результати часу для трьох переможців змагання.
7. Для роботи з масивом об'єктів побудувати та використати клас-ітератор.
8. Реалізувати пункти 1-7 завдання мовою C++ та C# з використанням WinForms.
9. Вимоги до оформлення курсової роботи дивитись у методичних вказівках до виконання курсового проектування з ООП (диск Р)
10. Забезпечити читання і запис даних для формування масиву об'єктів згідно завдання із файлу.

### Вимоги:

1. Вхідні дані до роботи: літературні джерела, технічна документація щодо розробки програм, ДСТУ з оформлення документації.
2. Зміст пояснювальної записки: Вступ. Специфікація роботи. Програмна документація. Висновки. Додатки.
3. Перелік графічного матеріалу: діаграма класів додатку; схема алгоритму реалізації одного з розроблених методів (за узгодженням з керівником курсової роботи).

**Дата здачі студентом завершеної роботи 10.05.2024 р.**

Календарний план  
виконання курсової роботи

| № з/п | Назва етапу роботи  | Строк виконання | Відмітка про виконання |
|-------|---|-----------------|------------------------|
| 1     | З'ясування загальної постановки завдання; розробка чернетки першого розділу пояснювальної записки   |                 |                        |
| 2     | Програмна реалізація, налагодження та тестування додатка; розробка чернетки пояснювальної записки (другий розділ, висновки, список використаних джерел, додатки). |                 |                        |
| 3     | Остаточне налагодження та тестування додатка; розробка чернетки пояснювальної записки (вступ, другий розділ, висновки, список використаних джерел, додатки).      |                 |                        |
| 4     | Розробка остаточного варіанта пояснювальної записки (вступ, другий розділ, висновки, список використаних джерел, додатки)...                                      |                 |                        |

Дата видачі завдання: 2.11.2023 р

Керівник роботи: Яцишин С.І. \_\_\_\_\_  
(підпис)

Студент: Симович О. \_\_\_\_\_  
(підпис )

## Зміст

|  |    |
|--|----|
| Вступ  | 3  |
| Специфікація роботи  | 4  |
| Основні концепції ООП  | 5  |
| Об'єкти і класи  | 6  |
| Поняття класу об'єктів   | 6  |
| Основні характеристики стану класу   | 6  |
| Поняття інкапсуляції властивостей об'єкту                                  | 7  |
| Поняття наслідування   | 7  |
| Поліморфізм  | 7  |
| Поняття про перевантаження операторів                                      | 8  |
| Віртуальні та дружні функції   | 10 |
| Контейнери й Ітератори   | 11 |
| Програмна документація   | 13 |
| Проектування структури програми  | 13 |
| Проектування інтерфейсу користувача  | 15 |
| Графічний інтерфейс на C# з використанням Windows Forms                    | 16 |
| Розроблення програми та її опис  | 17 |
| Розроблення програми з графічним інтерфейсом з використанням Windows Forms | 25 |
| Аналіз результатів комп'ютерної реалізації програми                        | 31 |
| Висновки   | 40 |
| Список використаних джерел   | 41 |
| Додатки  | 42 |
| Додаток А (Діаграма класів)  | 42 |
| Додаток В (Код програмних застосунків)                                     | 43 |

## Вступ

**Мета роботи:** Продемонструвати основи ООП на прикладі програми розробки класу «РЕЗУЛЬТАТИ», що містить масив об'єктів класу «СЕКУНДОМІР».

**Ключові слова:** клас, поля, методи, контейнери, ітератори, конструктор, деструктор, C#, C++, пізнє зв'язування, перевантаження операторів.

- Об'єктно-орієнтоване програмування (ООП) - це парадигма програмування, яка орієнтується на організацію програмного коду навколо об'єктів, які взаємодіють один з одним. Ідея ООП полягає в тому, щоб моделювати реальний світ або абстрактні концепції програмного середовища за допомогою об'єктів, які мають стан (атрибути) і поведінку (методи).
- Природним способом моделювання такої галузі є виділення класів об'єктів, які мають однакові властивості і поведінку, і встановлення між ними родових співвідношень. У свою чергу, в ООП ми визначаємо класи об'єктів, які відображають модель предметної галузі, з урахуванням спільності їх властивостей через ієрархію родових зв'язків.
- Наприклад, у курсовій роботі демонструються основи ООП на прикладі програми для реєстрування часу, для подальших дій з ним під час змагань. У програмі створено клас "DigitalCounter", який містить основні дані про час (мінімальний, максимальний і поточний), дочірній клас "Stopwatch", який розширює цей клас, додаючи дані про ім'я учасника, перевірку запуску секундоміру і методи зчитування та виведення значення заміряного часу. Також є контейнерний клас "Result", який містить масив об'єктів класу "Stopwatch", а також клас-ітератор для роботи з цим контейнерним класом.
- Програма реалізована мовами C++ та C# у середовищі Visual Studio 2022, з інтерфейсом у консолі та використанням Windows Forms відповідно.

## Специфікація роботи

Об'єктно-орієнтоване програмування (ООП) базується на декількох основних концепціях, які допомагають організувати та структурувати програми:

- **Класи і об'єкти:** Класи в ООП виступають як шаблони або моделі, за допомогою яких створюються об'єкти. Класи визначають структуру, властивості та поведінку об'єктів. Об'єкти є конкретними екземплярами класів, які мають свій стан (значення властивостей) і можуть виконувати певні дії (методи).
- **Успадкування:** Успадкування дозволяє створювати нові класи на основі вже існуючих. Цей процес називається похідністю або успадкуванням. Підкласи успадковують властивості та методи батьківських класів, що дозволяє перевикористовувати код і створювати ієрархію класів з загальною функціональністю.
- **Інкапсуляція:** Інкапсуляція в ООП використовується для обмеження доступу до даних та методів класу. Це дозволяє забезпечити контроль над внутрішньою структурою класу і захистити дані від неправильного використання. Інкапсуляція дозволяє зберігати дані приватними і надавати доступ до них лише через публічні методи, що забезпечує контрольовану взаємодію з об'єктами.
- **Поліморфізм:** Поліморфізм означає здатність об'єктів виконувати різні дії в залежності від типу об'єкта, з яким вони взаємодіють. Це означає, що об'єкти одного класу можуть мати різну реалізацію методів, але вони можуть бути викликані за допомогою одного спільного інтерфейсу. Поліморфізм дозволяє зводити код до загальної логіки, спрощує розширення програми і полегшує роботу з багатими ієрархіями класів.
- **Абстракція:** Абстракція дозволяє сконцентруватись на важливих аспектах об'єктів, ігноруючи незначні деталі. Це досягається шляхом визначення абстрактних класів та інтерфейсів, які встановлюють спільний контракт між класами і об'єктами, не розголошуючи їх конкретну реалізацію. Абстракція дозволяє створювати загальні моделі, які можуть бути розширені та спеціалізовані в похідних класах.

Ці концепції ООП допомагають покращити модульність, підтримуваність та повторне використання коду. Вони сприяють розбиттю програми на окремі компоненти (класи), які можуть працювати разом, але знаходитись у відокремленому стані. Крім того, ООП спрощує розвиток програмного забезпечення шляхом розподілу ролей та відповідальностей між класами та об'єктами.

Загалом, ООП надає потужні інструменти для створення структурованих, гнучких і розширюваних програм. Використання цих концепцій дозволяє програмістам працювати на вищому рівні абстракції, спрощує розробку складних систем та забезпечує більш ефективне управління кодом і ресурсами програми.

**Об'єкти і класи.** Формальне визначення об'єктів у об'єктно-орієнтованому програмуванні (ООП) полягає в тому, що об'єкти є сутностями, які чітко демонструють свою поведінку. Кожен об'єкт складається з трьох основних частин: назви, стану (змінних) та методів (операцій).

У широкому визначенні об'єкт в ООП є набором змінних стану та пов'язаних з ними методів. Ці методи визначають, як об'єкт взаємодіє з навколишнім середовищем. Важливою особливістю є можливість контролювати стан об'єкта за допомогою виклику методів, які визначають його поведінку. Цей набір методів часто називають об'єктним інтерфейсом.

Клас визначає набір даних і методів, які використовуються для обробки цих даних. Клас можна розглядати як шаблон. Об'єкти, які мають однакові властивості, утворюють клас.

Об'єкти в програмуванні представляють конкретні реалізації або екземпляри класів. Ці об'єкти зазвичай відображають конкретні сутності або об'єкти з реального світу. Класи, натомість, є абстракціями, що визначають структуру та поведінку об'єктів. Для створення конкретного об'єкта потрібно мати відповідний клас, який виступає як шаблон.

Методи в класах визначають їхню поведінку. Це функції або процедури, які виконують різноманітні операції з даними об'єкта.

Поняття "повідомлення" в ООП є аналогією виклику функцій у звичайному програмуванні. У контексті ООП, замість того, щоб безпосередньо змінювати стан об'єкта, ми "надсилаємо" повідомлення об'єкту, яке вказує йому, яку операцію виконати. Об'єкт, у свою чергу, обробляє це повідомлення та змінює свій стан відповідно до нього. Такий підхід дозволяє забезпечити більшу контрольованість та гнучкість у взаємодії з об'єктами.

**Поняття класу об'єктів.** Клас об'єктів в ООП розглядається як абстрактний тип даних. Це означає, що як деяке єдине ціле в програмі виступає певна сукупність даних і набір операцій (процедур), які можуть бути виконані з цими даними. Клас є ключовим поняттям C++. За визначенням Б.Страуструпа "Клас - це тип, визначуваний користувачем". Класи забезпечують приховання даних, гарантовану ініціалізацію даних, неявне перетворення типів для типів, визначених користувачем, динамічне завдання типу, контрольоване користувачем управління пам'яттю і механізми перевантаження операцій.

**Основні характеристики стану класу.** Сукупність даних, які характеризують поточний стан об'єкту певного класу і його взаємозв'язку з іншими об'єктами, називають локальною (приватною) пам'яттю об'єкту, а визначені для об'єктів даного класу операції - методами об'єктів класу. Сукупність всіх методів, специфікованих для деякого класу, називається інтерфейсом класу.

**Інкапсуляція** властивостей об'єкту, що є ключовим поняттям у об'єктно-орієнтованому програмуванні (ООП), відображає принцип збереження та контролю доступу до внутрішньої структури об'єкту. Цей принцип, базуючись на концепції інформаційного сховища та розділення обов'язків, визначає можливість об'єктів зберігати дані (властивості) та виконувати операції над ними (методи) у внутрішньому просторі, недоступному безпосередньо для зовнішнього коду. При використанні інкапсуляції, властивості об'єкту можуть бути оголошені з різними рівнями доступу, такими як публічний, приватний або захищений. **Публічні** властивості доступні для зовнішнього коду і можуть бути змінені або отримані безпосередньо. **Приватні** властивості є недоступними безпосередньо ззовні об'єкта і можуть бути доступні лише через публічні методи (гетери та сетери), що дозволяє забезпечити контрольований доступ до даних. **Захищені** властивості можуть бути доступні лише для підкласів та самого об'єкта, що забезпечує більшу гнучкість при розширенні та зміні функціональності.

**Наслідування** є іншим важливим принципом ООП, що дозволяє створювати нові класи на основі вже існуючих. Цей механізм дозволяє підкласам успадковувати властивості та методи батьківського класу, що сприяє полегшенню розширення та модифікації функціональності. Підкласи можуть додавати нові властивості та методи, а також перевизначати або розширювати властивості та методи, успадковані від батьківського класу. Цей механізм сприяє створенню ієрархічних структур класів та забезпечує можливість перевикористовувати код.

**Поліморфізм**, третій принцип ООП, відображає здатність об'єктів виконувати різні дії в залежності від їх типу. Це означає, що об'єкти одного класу можуть мати різну реалізацію методів, але вони можуть бути викликані за допомогою одного загального інтерфейсу. Поліморфізм дозволяє використовувати об'єкти різних класів з однаковим інтерфейсом без необхідності знати конкретний тип об'єкта. Це забезпечує гнучкість та розширюваність коду, оскільки можна працювати з колекціями об'єктів різних типів, викликаючи спільні методи і отримуючи різні результати залежно від типу кожного об'єкта. Існує два типи поліморфізму: статичний поліморфізм і динамічний поліморфізм. **Статичний** поліморфізм (також відомий як перевантаження) дозволяє викликати різні реалізації методу залежно від переданих аргументів. Це досягається за допомогою перевантажених методів, де методи з однаковою назвою, але з різними параметрами, можуть бути викликані в залежності від типів переданих аргументів. **Динамічний** поліморфізм базується на успадкуванні та віртуальних методах. Віртуальні методи дозволяють підкласам перевизначати методи батьківського класу зі своєю власною реалізацією. Коли метод викликається для об'єкта, система викликає відповідну реалізацію методу залежно від типу об'єкта виклику. Це забезпечує можливість заміщення методів і створення поліморфних поведінок, де різні об'єкти можуть мати різну реалізацію методу з однаковою назвою.



**Перевантаження операторів** - це механізм в мові програмування C++, який дозволяє змінювати поведінку стандартних операторів для об'єктів класів або типів даних. Цей механізм дозволяє визначити спеціальні функції, які будуть викликатися при використанні операторів з відповідними типами об'єктів.

Давайте розглянемо приклади перевантаження цих операторів для класу "Counter", який представляє лічильник.

Постфіксний декремент (--):

```
class Counter {
private:
    int count;

public:
    Counter(int initialCount) : count(initialCount) { }

    // Перевантаження постфіксного декременту
    Counter operator--(int) {
        Counter temp(count); // Створення копії поточного об'єкта
        count--; // Зменшення значення лічильника
        return temp; // Повернення копії перед зменшенням
    }
};
```

У цьому прикладі, коли використовується постфіксний оператор --, викликається перевантажений оператор operator--(int). Він створює копію поточного об'єкта, зменшує значення лічильника і повертає копію перед зменшенням.

```
Counter c(5);
Counter result = c--; // Виклик перевантаженого постфіксного декременту
```

У цьому прикладі змінна c має початкове значення 5. При виконанні c--, значення лічильника зменшується на 1, але результатом є копія об'єкта c перед зменшенням.

Префіксний інкремент (++):

```
class Counter {
private:
    int count;

public:
    Counter(int initialCount) : count(initialCount) { }
```

```
// Перевантаження префіксного інкременту
Counter& operator++() {
    count++; // Збільшення значення лічильника
    return *this; // Повернення посилання на поточний об'єкт
}
};
```

У цьому прикладі, коли використовується префіксний оператор ++, викликається перевантажений оператор operator++(). Він збільшує значення лічильника і повертає посилання на поточний об'єкт.

```
Counter c(5);
Counter& result = ++c; // Виклик перевантаженого префіксного інкременту
```

У цьому прикладі змінна c має початкове значення 5. При виконанні ++c, значення лічильника збільшується на 1, і результатом є посилання на об'єкт c.

Перевантаження префіксного інкременту і постфіксного декременту дозволяє змінювати поведінку цих операторів для власних класів. Приклади, які я навів, демонструють лише основну ідею перевантаження, але ви можете розширити їх функціональність залежно від потреб вашої програми.

**Віртуальні функції** є ключовим поняттям в об'єктно-орієнтованому програмуванні (ООП) і використовуються для досягнення поліморфізму. Вони дозволяють об'єктам різних класів використовувати спеціальні реалізації функцій залежно від їх типу або посилання/вказівника, за якими вони викликаються.

У мові програмування C++, віртуальна функція визначається за допомогою ключового слова **virtual** в оголошенні базового класу. Похідні класи можуть перевизначати цю функцію за допомогою ключового слова **override**, що вказує на намір перевизначення.

Основні переваги віртуальних функцій включають:

1. Поліморфізм: Віртуальні функції дозволяють замінювати об'єкти посиланнями або вказівниками базового класу і викликати їх методи, що залежать від фактичного типу об'єкту. Це дозволяє розробляти загальні алгоритми, що працюють з різними типами об'єктів, і забезпечувати їх власну специфічну реалізацію в похідних класах.
2. Розширюваність: Віртуальні функції дозволяють додавати нові функціональні можливості до похідних класів, не змінюючи існуючий код базового класу. Це дозволяє легко розширювати функціонал програми без необхідності модифікувати багато коду.
3. Зворотна сумісність: Віртуальні функції дозволяють створювати ієрархії класів, де похідні класи можуть бути використані як заміна базових класів без порушення інтерфейсу. Це дозволяє зберігати зворотну сумісність коду, що залежить від базового класу, навіть якщо в майбутньому будуть введені нові похідні класи.

Отже, віртуальні функції важливі для реалізації поліморфізму і дозволяють створювати гнучкі та розширювані програми в рамках об'єктно-орієнтованого підходу. Вони є основою для використання поліморфних властивостей мови C++ Віртуальні функції грають важливу роль в об'єктно-орієнтованому програмуванні (ООП) і дозволяють досягти поліморфізму. Вони дозволяють об'єктам різних класів використовувати спеціалізовані реалізації функцій в залежності від їх типу або посилання/вказівника, за якими вони викликаються.

**Дружні функції** є концепцією в об'єктно-орієнтованому програмуванні, яка дозволяє функціям мати прямий доступ до приватних та захищених членів класу, до якого вони відносяться. Це дозволяє функції працювати з даними класу, що зазвичай були б недоступними. Опис дружніх функцій:

У класі можна оголосити функцію як "дружню", використовуючи ключове слово friend. Це означає, що ця функція має привілейований статус та може отримувати доступ до всіх приватних та захищених членів цього класу.

Одним з основних використань дружніх функцій є забезпечення зручного доступу до даних класу без необхідності робити ці дані публічними. Це дозволяє зберегти принцип інкапсуляції і забезпечити контроль над доступом до даних.

**Контейнери.** Як впливає з назви, контейнер - це об'єкт, який може містити в собі інші об'єкти. Існує кілька різних типів контейнерів.

Наприклад, клас Vector визначає динамічний масив, Deque створює двонаправлену чергу, а List являє собою зв'язний список. Ці контейнери називаються послідовними контейнерами (sequence containers), тому що в термінології STL послідовність - це лінійний список. STL також визначає асоціативні контейнери (associative containers), які забезпечують ефективний витяг значень на основі ключів.

Таким чином, асоціативні контейнери зберігають пари "ключ / значення". Прикладом може служити Map. Цей контейнер зберігає пари "ключ / значення", в яких кожен ключ є унікальним.

**Ітератори.** Ітератори - це об'єкти, які ведуть себе більш-менш подібно вказівникам.

Вони надають можливість виконувати циклічну обробку елементів контейнера - подібно до того, як використовується покажчик для організації циклу по масиву.

Існує п'ять типів ітераторів.

**Ітератори введення**(input\_iterator) підтримують операції рівності, розіменування й інкремента.

$==, !=, *i, ++i, i++, *i++$

Спеціальним випадком ітератора введення є istream\_iterator.

**Ітератори виведення**(output\_iterator) підтримують операції розіменування, припустимі тільки з лівої сторони присвоювання, і інкремента.

$++i, i++, *i=t, *i++=t$

Спеціальним випадком ітератора виводу є ostream\_iterator.

**3. Односпрямовані Ітератори**(forward\_iterator) підтримують всі операції ітераторів введення/виводу й, крім того, дозволяють без обмеження застосовувати присвоювання.

$==, !=, =, *i, ++i, i++, *i++$

**4. Двонаправлені Ітератори**(bidirectional\_iterator) мають всі властивості forward-ітераторів, а також мають додаткову операцію декремента ( $--i, i--, *i--$ ), що дозволяє їм проходити контейнер в обох напрямках.

**5. Ітератори довільного доступу**(random\_access\_iterator) мають всі властивості bidirectional-ітераторів, а також підтримують операції порівняння й адресної арифметики, тобто безпосередній доступ по індексі.

$i+=n, i+n, i-=n, i-n, i1-i2, i[n], i1<i2, i1<=i2, i1>i2, i1>=i2$

В STL також підтримуються зворотні Ітератори (reverse iterators). Зворотними ітераторами можуть бути або двонаправлені Ітератори, або Ітератори довільного доступу, але мають послідовність у зворотному напрямку.

У C#, ітератори є зручним засобом для ітерації (перебору) через колекції або колекційні об'єкти, такі як списки, масиви, словники тощо. Вони дозволяють пройти по елементах колекції без необхідності використання циклів або інших структур керування.

Ітератори в C# реалізуються за допомогою ключового слова 'yield', яке дозволяє методу повертати послідовність значень одне за одним, без необхідності повертати всю колекцію відразу.

Ітератори у С# дозволяють просто та ефективно працювати з колекціями, роблячи код більш зрозумілим та компактним.

Інтерфейс `IEnumerable` та ітератори в С# взаємопов'язані та часто використовуються разом для роботи з колекціями даних.

1. `IEnumerable`: Цей інтерфейс визначає метод `GetEnumerator()`, який повертає об'єкт, який реалізує інтерфейс `IEnumerator`. Інтерфейс `IEnumerable` не визначає метод для ітерації через колекцію, але вказує, що колекція може бути перебрана. Його часто використовують у контексті колекційних об'єктів.

```
public interface IEnumerable
{
    IEnumerator GetEnumerator();
}
```

2. Ітератори: Ітератори в С# використовують ключове слово `yield` для реалізації простого та зручного механізму ітерації через колекції. Ітератори повертають послідовність значень одне за одним при кожному виклику. Коли викликається метод, що містить `yield`, виконання методу призупиняється, а значення передаються назовні. Після цього метод продовжує виконання з місця припинення.

Комбінація `IEnumerable` та ітераторів робить код більш читабельним та ефективним. Ітератори дозволяють зручно ітерувати через колекції, а `IEnumerable` вказує, що ця колекція може бути перебрана за допомогою ітератора. Використання цих інтерфейсів дозволяє створювати код, який легко зрозуміти та підтримувати.

Інтерфейс `IEnumerator` в С# використовується для створення власних ітераторів, які дозволяють ітерувати (перебирати) елементи колекцій один за одним.

Ось опис інтерфейсу `IEnumerator`:

```
public interface IEnumerator
{
    object Current { get; } // Повертає поточний елемент у колекції
    bool MoveNext();        // Переміщує вказівник до наступного елементу у
колекції
    void Reset();           // Скидає вказівник до початкового стану колекції
}
```

## Програмна документація

Батьківський клас DigitalCounter визначає лічильник, який може бути використаний для вимірювання часу або будь-яких інших величин.

Клас містить методи для встановлення і отримання значень лічильника, перевірки меж, а також реалізацію операції () для встановлення значень полів даних, операцію ++ для збільшення значення лічильника на 1, - - для зменшення значення лічильника на 1, += для збільшення значення лічильника на задану величину, -= для зменшення значення лічильника на задану величину, операцію присвоєння об'єктів = та переміщення, потокові операції введення » та виведення « об'єктів.

Основні функції включають конструктори, деструктори, методи для отримання і встановлення меж, а також перевантаження операторів.

ArrayIterator.h:

Цей файл містить шаблон класу ArrayIterator, який служить для ітерації по вектору. Використовується для обходу векторів в класі Result.

```
#pragma once
#include <vector>

template <typename T>

class ArrayIterator
{
private:
    // An iterator that points to the current element of the vector
    typename std::vector<T>::iterator iter;

public:
    // Default constructor
    ArrayIterator() = default;
    // Constructor that initializes the iterator with the specified value
    ArrayIterator(typename std::vector<T>::iterator iter) : iter(iter) {}

    // Overloaded dereference operator, returns a reference to the current element
    T& operator*()
    {
        return *iter;
    }

    // Overloaded prefix increment operator, moves the iterator to the next element
    ArrayIterator<T>& operator++()
    {
        ++iter;
        return *this;
    }

    // Overloaded arrow operator, returns a pointer to the current element
    T* operator-->()
    {
        return &(*iter);
    }

    // Overloaded comparison operator for inequality, compares two iterators
    bool operator!=(const ArrayIterator<T>& other) const
```

```

    {
        return iter != other.iter;
    }
};

```

Stopwatch.h:

Унаслідуює всі методи та функціонал класу DigitalCounter та доповнює новими полями й методами для перевірки на запуск та встановлення й отримання імені:

```

class Stopwatch : public DigitalCounter
{
private:
    bool isRunning; // Flag indicating whether the stopwatch is running
    std::string name;

    bool isStarted() const; // Check if the stopwatch is running
    // Methods for setting and getting the participant's name
    void setName(const std::string& newName);
    std::string getName() const;

```

Result.h:

Методи: addRes() для додавання результатів забігу, getMinTime(), getMaxTime(), getAvgTime() для отримання статистики про результати, getBestOf3() для визначення трьох найкращих результатів і вектори для зберігання результатів:

```

#pragma once
#include "Stopwatch.h"
#include "ArrayIterator.h"
#include <algorithm>

class Result
{
private:
    // Vector to store race results
    std::vector<Stopwatch> results;
public:
    // Events with the result of the race
    void addRes(const Stopwatch& result);
    Stopwatch getMinTime() const;
    Stopwatch getMaxTime() const;
    double getAvgTime() const;

    // Vector to get all race results and three winners
    std::vector<Stopwatch> getResults() const;
    std::vector<Stopwatch> getBestOf3();
};

```

Ця програма моделює функціонал лічильника та секундоміра для вимірювання часу. Класи DigitalCounter та Stopwatch надають можливість встановлення меж та виконання операцій інкременту, декременту тощо. Клас Result використовується для обробки результатів забігів або подій, використовуючи об'єкти Stopwatch.

## Проектування інтерфейсу користувача

У завданні вказано, що мають бути перевантажені операції вставки та витягання для певних класів, а це можливо лише в тому випадку, якщо програму написано для консолі. А це один з вирішальних факторів при проектуванні користувацького інтерфейсу. Отже, необхідно передбачити зручний діалог при роботі у консолі з користувачем. Оскільки програма передбачає роботу з контейнерним класом, то необхідно передбачити зручне та зрозуміле введення інформації. На далі діалог з користувачем зручно організувати у вигляді меню. Пункти цього меню мають надавати певний функціонал користувачу і водночас забезпечити демонстрацію поставлених перед нами підзадач. Отже, доцільно визначити такі пункти меню:

```
Select the source of the input data or other events:

1) Manual input (must enter data from the console)
2) Data from file (data is read from the input file)
3) Demonstration all constructors
4) Demonstration of operators overloading
5) Close program

Input your operation:
```

Перших два пункти меню реалізовано так, щоб продемонструвати вибір вводу даних (при виборі «1» - дані потрібно буде ввести вручну з консолі, при виборі «2» - дані зчитуються з вхідного файлу) і появляться подальші можливі дії з контейнером, буде представлена демонстрація механізму пізнього зв'язування.

```
Input your operation: 2
Selected option 2: Data from file
The file was successfully read and closed :)

Select the source of the output data:

1) To the console
2) To a file
3) Close program

Input your operation:
```

Наступний третій пункт меню – це демонстрація всіх конструкторів батьківського класу а саме конструктора за замовчуванням, з параметрами, копіювання та переміщення.

Четвертий пункт дозволяє побачити реалізацію перевантажених операторів.



П'ятий пункт відповідає за вихід з програми.

Для всіх інших введених значень є перевірки, які будуть спрацювати і не давати доступитися до самої програми чи до її примусового припинення роботи.

Отже, ці пункти меню надають користувачеві необхідний функціонал для роботи зі всіма класами, і водночас задовольняють поставлені перед нами завдання про демонстрацію певних механізмів реалізації програмного коду.

### Графічний інтерфейс на C# з використанням Windows Forms:

Тепер розглянемо проектування другої версії програми з графічним інтерфейсом (ГІ). Програма з ГІ складається з базової форми і розміщених на ній елементів керування та трьох додаткових форм, призначених для додаткових параметрів. На основній формі розміщені такі елементи керування як 2 bindingNavigator, на яких розміщені 6 Label та 6 toolStripButton з піктограмами, також є 1 pictureBox для розуміння з чим пересікається програма.



Тут також є багато перевірок, які не дадуть нам можливість спочатку натискати такі кнопки як DeleteCompetitor, ViewList, WriteToFile, адже наша колекція ще пуста і ми не можемо робити ці дії, тому слід почати або з заповнення колекції вручну, натиснувши кнопку AddCompetitor або заповнення з вхідного файлу при натисканні кнопки ReadFromFile.

## Розроблення програми та її опис

Після планування структури програми, розроблення її екранних форм, елементів діалогу, графічних схем класів та відношень між ними здійсимо деталізацію структури складових частин програми, які загалом повинні виконувати розв'язування поставленої задачі та забезпечувати взаємодію з користувачем через спроектований інтерфейс. Розроблення складових частин вимагає знання можливостей мови програмування C++ і середовища програмування VisualStudio 2022. Оскільки описувати всі реалізовані у програмі методи – це занадто об'ємна робота для курсового проекту, то представимо реалізацію найбільш цікавих методів.

Розглянемо реалізацію полів конструкторів класів, основних методів, перевантажених операторів, дружніх функцій потокового вводу-виводу.

Почнемо з батьківського класу DigitalCounter.h:

Поля для збереження можливих значень лічильника:

```
private:
    // Possible counter values
    double value;
    double minimum;
    double maximum;
```

Всі конструктори та деструктор класу:

```
public:
    // The default constructor
    DigitalCounter();
    // Constructor specifying the minimum and maximum values
    DigitalCounter(double min, double max);
    // Copy constructor
    DigitalCounter(const DigitalCounter& other);
    // Move constructor
    DigitalCounter(DigitalCounter&& other);
    // Destructor
    virtual ~DigitalCounter();
```

Найважливіші методи класу для встановлення та отримання значень:

```
// Setting values
void setMinimum(double min);
void setMaximum(double max);
void setValue(double value);

// Getting the values
double getValue() const;
double getMinimum() const;
double getMaximum() const;
```

Перевантажені оператори, які в мене в подальшому використовуються:

```
// Operators overloading
DigitalCounter& operator= (const DigitalCounter& other);
DigitalCounter& operator= (DigitalCounter&& other);
DigitalCounter& operator++();
DigitalCounter& operator--();
DigitalCounter& operator+=(int increment);
```

Реалізація всього продемонстрованого в DigitalCounter.cpp:

Конструктори:

```
DigitalCounter::DigitalCounter() : minimum(10.0), maximum(15.0), value(minimum)
{
    checkBounds();
}
```

DigitalCounter(): Це конструктор за замовчуванням, який ініціалізує мінімальне значення на 10.0, максимальне значення на 15.0 та поточне значення на мінімальне значення. Після ініціалізації він викликає метод checkBounds(), щоб переконатися, що значення знаходиться в межах допустимих границь.

```
DigitalCounter::DigitalCounter(double min, double max) : minimum(min), maximum(max),
value(min)
{
    checkBounds();
}
```

DigitalCounter(double min, double max): Цей конструктор дозволяє користувачеві встановити специфічні значення для мінімального та максимального значень, а також ініціалізує поточне значення як мінімальне значення. Після ініціалізації він також викликає метод checkBounds() для перевірки меж допустимих значень.

```
DigitalCounter::DigitalCounter(const DigitalCounter& other) : minimum(other.minimum),
maximum(other.maximum), value(other.value)
{
    checkBounds();
}
```

DigitalCounter(const DigitalCounter& other): Це конструктор копіювання, який створює новий об'єкт, копіюючи значення мінімального, максимального та поточного значень з іншого об'єкта DigitalCounter. Після цього він також викликає метод checkBounds() для перевірки меж допустимих значень.

```
DigitalCounter::DigitalCounter(DigitalCounter&& other) : minimum(other.minimum),
maximum(other.maximum), value(other.value)
{
    other.minimum = 0;
    other.maximum = 0;
    other.value = 0;
    checkBounds();
}
```

DigitalCounter(DigitalCounter&& other): Це конструктор переміщення, який приймає інший об'єкт DigitalCounter через посилання на правий операнд та переміщує його дані до нового об'єкта. Після цього він обнуляє дані у переданому об'єкті. Після ініціалізації він викликає метод checkBounds() для перевірки меж допустимих значень.

Для реалізації конструкторів в мейні програми, за це відповідає метод `demonstrateConstructors()`, який викликається при запуску програми і після обрання події «Демонстрація всіх конструкторів» (клавіша «3»):

```
void demonstrateConstructors()
{
    std::cout << "\n *----- At this stage, you can see a demonstration of the
constructors -----*\n\n";

    DigitalCounter counter1;
    std::cout << " Default constructor: Minimum race time = " << counter1.getMinimum() <<
"sec, Maximum race time = " << counter1.getMaximum() << "sec\n";

    DigitalCounter counter2(15.5, 18.7);
    std::cout << " Constructor with parameters : Minimum race time = " <<
counter2.getMinimum() << "sec, Maximum race time = " << counter2.getMaximum() << "sec\n";

    DigitalCounter counter3(counter2);
    std::cout << " Copy constructor: Minimum race time = " << counter3.getMinimum() <<
"sec, Maximum race time = " << counter3.getMaximum() << "sec\n";

    DigitalCounter counter4(std::move(counter1));
    std::cout << " Move constructor: Minimum race time = " << counter4.getMinimum() <<
"sec, Maximum race time = " << counter4.getMaximum() << "sec\n\n";
}
```

Select the source of the input data or other events:

- 1) Manual input (must enter data from the console)
- 2) Data from file (data is read from the input file)
- 3) Demonstration all constructors
- 4) Demonstration of operators overloading
- 5) Close program

Input your operation: 3

Selected option 3: Demonstration all constructors

\*----- At this stage, you can see a demonstration of the constructors -----\*

Default constructor: Minimum race time = 10sec, Maximum race time = 15sec  
Constructor with parameters : Minimum race time = 15.5sec, Maximum race time = 18.7sec  
Copy constructor: Minimum race time = 15.5sec, Maximum race time = 18.7sec  
Move constructor: Minimum race time = 10sec, Maximum race time = 15sec

Основні методи:

```
void DigitalCounter::setMinimum(double min)
{
    minimum = min;
    checkBounds();
}
```

`setMinimum(double min)`: Цей метод приймає нове значення мінімуму і присвоює його полю `minimum`. Після цього він викликає метод `checkBounds()`, щоб перевірити, чи відповідає нове значення межах допустимих границь.

```
void DigitalCounter::setMaximum(double max)
{
    maximum = max;
    checkBounds();
}
```

setMaximum(double max): Цей метод приймає нове значення максимуму і присвоює його полю maximum. Після цього він також викликає метод checkBounds() для перевірки меж допустимих значень.

```
void DigitalCounter::setValue(double value)
{
    if (value <= maximum && value >= minimum)
    {
        this->value = value;
    }
    else if (value < minimum)
    {
        this->value = minimum;
    }
    else
    {
        this->value = maximum;
    }
    checkBounds();
}
```

setValue(double value): Цей метод встановлює нове поточне значення лічильника. Якщо вхідне значення знаходиться в межах мінімуму та максимуму, воно присвоюється полю value. Якщо вхідне значення менше мінімуму, value приймає значення мінімуму. Якщо вхідне значення більше максимуму, value приймає значення максимуму. Після цього викликається метод checkBounds() для перевірки меж допустимих значень.

```
double DigitalCounter::getValue() const
{
    return value;
}
```

```
double DigitalCounter::getMinimum() const
{
    return minimum;
}
```

```
double DigitalCounter::getMaximum() const
{
    return maximum;
}
```

Ці 3 методи повертають відповідні значення лічильника.

Всі ці методи є основою програми адже реалізуються в інших класах.

## Перевантажені оператори:

```
DigitalCounter& DigitalCounter::operator=(const DigitalCounter& other)
{
    if (this != &other)
    {
        minimum = other.minimum;
        maximum = other.maximum;
        value = other.value;
    }
    return *this;
}
```

operator= (const DigitalCounter& other): Цей оператор присвоєння копіює дані з одного об'єкта DigitalCounter в інший. Він перевіряє, чи не присвоюється об'єкт самому собі (порівняння this != &other), а потім копіює значення мінімуму, максимуму та поточного значення з іншого об'єкта до поточного. На завершення повертає посилання на поточний об'єкт.

```

DigitalCounter& DigitalCounter::operator=(DigitalCounter&& other)
{
    if (this != &other)
    {
        minimum = other.minimum;
        maximum = other.maximum;
        value = other.value;
        other.minimum = 0;
        other.maximum = 0;
        other.value = 0;
    }
    return *this;
}

```

operator=(DigitalCounter&& other): Цей оператор присвоєння виконує переміщення об'єкта DigitalCounter. Він копіює значення мінімуму, максимуму та поточного значення з іншого об'єкта до поточного, а потім обнуляє значення мінімуму, максимуму та поточного значення в іншому об'єкті.

```

DigitalCounter& DigitalCounter::operator++()
{
    ++value;
    return *this;
}

```

operator++(): Цей префіксний оператор інкременту збільшує поточне значення лічильника на одиницю.

```

DigitalCounter DigitalCounter::operator--(int)
{
    DigitalCounter temp(*this);
    temp.value = value - 1;
    return temp;
}

```

operator--(int): Цей постфіксний оператор декременту. Він створює копію поточного об'єкта, зменшує значення лічильника на одиницю, а потім повертає цю копію.

```

DigitalCounter& DigitalCounter::operator+=(int increment)
{
    value += increment;
    return *this;
}

```

operator+=(int increment): Цей оператор додає вказане значення до поточного значення лічильника. Після додавання він повертає посилання на поточний об'єкт.

Для демонстрації перевантажених операторів в мейні програми, за це відповідає метод demonstrateOperators(), який викликається при запуску програми і після обрання події «Демонстрація перевантажень операторів» (клавіша «4»):

```

void demonstrateOperators()
{
    std::cout << "\n *----- At this stage, you can see the change of race result for  
the disputed participant -----*\n\n";

    DigitalCounter counter(8.4, 12.6);
    std::cout << " One of the participants has the following results: minimum = " <<
counter.getMinimum() << "sec and maximum: " << counter.getMaximum() << "sec time of the  
sprinter's race!\n";
    std::cout << " The judges wanted to choose the winner because he has the best minimum  
race time = " << counter.getMinimum() << " seconds!\n";
}

```

```

std::cout << " But then it turned out that he started faster and 1 second was added to
him, now his race time = " << (++counter).getValue() << " seconds!\n";
std::cout << " He didn't like it and appealed, where he won himself 1 second, now it's
his time again = " << (counter--).getValue() << " seconds!\n";
std::cout << " But after the competition, organizers rewatched video and noticed delay,
so added 3 seconds to the sprinter!\n";
std::cout << " It's final result = " << (counter += 3).getValue() << " seconds, due to
which the sprinter took one of the last places!\n\n";
}

```

Select the source of the input data or other events:

- 1) Manual input (must enter data from the console)
- 2) Data from file (data is read from the input file)
- 3) Demonstration all constructors
- 4) Demonstration of operators overloading
- 5) Close program

Input your operation: 4

Selected option 4: Demonstration of operators overloading

\*----- At this stage, you can see the change of race result for the disputed participant -----\*

One of the participants has the following results: minimum = 8.4sec and maximum: 12.6sec time of the sprinter's race!  
The judges wanted to choose the winner because he has the best minimum race time = 8.4 seconds!  
But then it turned out that he started faster and 1 second was added to him, now his race time = 9.4 seconds!  
He didn't like it and appealed, where he won himself 1 second, now it's his time again = 8.4 seconds!  
But after the competition, organizers rewatched video and noticed delay, so added 3 seconds to the sprinter!  
It's final result = 11.4 seconds, due to which the sprinter took one of the last places!

Варто розглянути шаблонний клас – ітератор, який застосовується для роботи з контейнерним класом РЕЗУЛЬТАТИ:

```

#pragma once
#include <vector>

template <typename T>

class ArrayIterator
{
private:
    // An iterator that points to the current element of the vector
    typename std::vector<T>::iterator iter;

public:
    // Default constructor
    ArrayIterator() = default;
    // Constructor that initializes the iterator with the specified value
    ArrayIterator(typename std::vector<T>::iterator iter) : iter(iter) {}

    // Overloaded dereference operator, returns a reference to the current element
    T& operator*()
    {
        return *iter;
    }

    // Overloaded prefix increment operator, moves the iterator to the next element
    ArrayIterator<T>& operator++()
    {
        ++iter;
        return *this;
    }
}

```

```

}

// Overloaded arrow operator, returns a pointer to the current element
T* operator->()
{
    return &(*iter);
}

// Overloaded comparison operator for inequality, compares two iterators
bool operator!=(const ArrayIterator<T>& other) const
{
    return iter != other.iter;
}
};

```

Реалізація цього класу в самому контейнерному класі РЕЗУЛЬТАТИ:  
Result.cpp:

```

std::vector<Stopwatch> Result::getBestOf3()
{
    std::vector<Stopwatch> myTop;

    if (results.empty())
    {
        return myTop;
    }

    // Using a standard iterator for sorting
    std::sort(results.begin(), results.end(), [](const Stopwatch& a, const Stopwatch& b)
    {
        return a.getValue() < b.getValue();
    });

    // Save the third smallest result
    auto thirdMinIter = results.begin() + 2;

    ArrayIterator<Stopwatch> beginIter(results.begin());
    ArrayIterator<Stopwatch> endIter(results.end());

    // Add these values to a new vector
    for (ArrayIterator<Stopwatch> iter = beginIter; iter != endIter; ++iter)
    {
        if (iter->getValue() <= thirdMinIter->getValue())
        {
            myTop.push_back(*iter);
        }
    }

    return myTop;
}

```

Для реалізації поставленого перед нами завдання достатньо було організувати ітератор введення, і перевантажити для нього операції \*, ++, -, >, !=. Однак для даного завдання доцільніше було б організовувати прохід по масиву об'єктів у класі РЕЗУЛЬТАТИ, використовуючи вказівники на елементи масиву.



Реалізований механізм пізнього зв'язування у даному фрагменті:

```
DigitalCounter* res = nullptr;

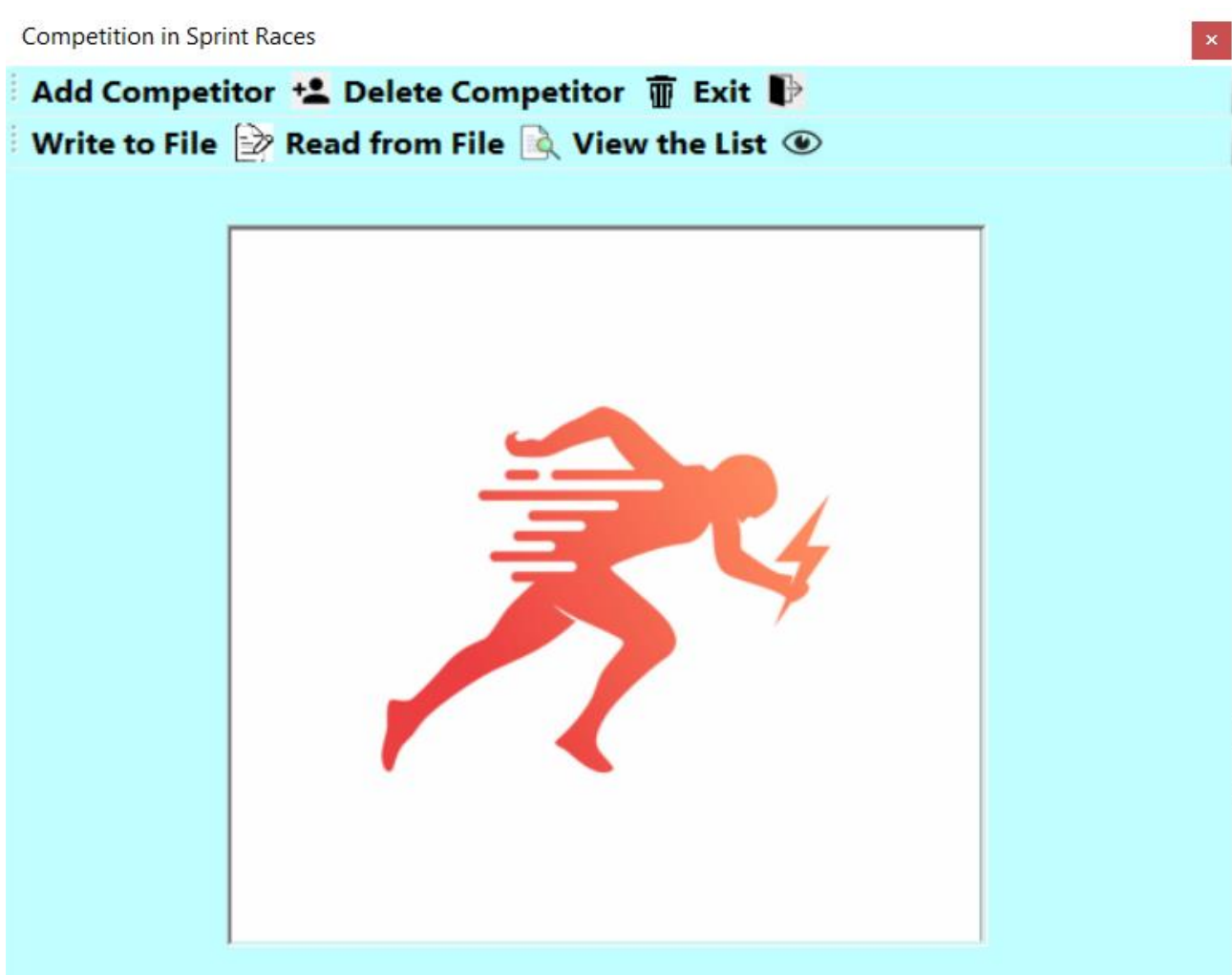
// Create, start, and stop the Stopwatch object
auto sv = Stopwatch(name, min, max);
sv.start();
sv.stop(max);

res = &sv;
if (dynamic_cast<Stopwatch*>(res) != nullptr)
{
    result.display(*dynamic_cast<Stopwatch*>(res));
}
```

Коли викликається метод `display()` за допомогою посилання на базовий клас `DigitalCounter`, компілятор не знає точного типу об'єкта, на який вказує вказівник `res`. Але, використовуючи `dynamic_cast<Stopwatch*>(res)`, він перевіряє, чи вказівник `res` вказує на об'єкт класу `Stopwatch`. І якщо це так, то викликає метод `display()` саме для об'єкта класу `Stopwatch`. Це дозволяє змінювати реалізацію методу `display()` у підкласі `Stopwatch`.

## Розроблення програми з графічним інтерфейсом з використанням Windows Forms

Розглянемо розробку програми на мові С# з використанням WindowsForms. На рисунку нижче зображено графічний інтерфейс програми спроектований засобами VisualStudio 2022.



Спочатку, розглянемо опис властивостей батьківського класу `DigitalCounter`, які містять методи `get` і `set` для встановлення їх значень:

```
public class DigitalCounter : IComparable<DigitalCounter>
{
    private double min;
    private double max;
    private double currentValue;

    public double MinValue
    {
        get { return min; }
        set
        {
            if (value >= max)
```

```

        {
            MessageBox.Show("Minimum value must be less than maximum value!", "Error
:(", MessageBoxButtons.OK, MessageBoxIcon.Error);
        }
        else
        {
            min = value;
        }
    }
}

public double MaxValue
{
    get { return max; }
    set
    {
        if (value <= min)
        {
            MessageBox.Show("Maximum value must be greater than min value!", "Error
:(", MessageBoxButtons.OK, MessageBoxIcon.Error);
        }
        else
        {
            max = value;
        }
    }
}

public double Value
{
    get { return currentValue; }
    set
    {
        if (value < min || value > max)
        {
            MessageBox.Show("Value is out of range!", "Error :(", MessageBoxButtons.OK,
MessageBoxIcon.Error);
        }
        else
        {
            currentValue = value;
        }
    }
}

```

Поля відповідно до їх назв відповідають за значення секундоміру (мінімальне, максимальне та поточне). У властивостях ми перевіряємо щоб кожне з цих значень не виходило за свої встановлені межі.

Також в цьому класі розглянемо всі конструктори, можливі перевантажені оператори, метод порівняння CompareTo() з інтерфейсу IComparable та віртуальний метод виводу інформації:

```

public DigitalCounter()
{
    min = 11.5;

```

```

        max = 15.5;
        currentValue = 60.0;
    }

    public DigitalCounter(double minValue, double maxValue)
    {
        min = minValue;
        max = maxValue;
        currentValue = minValue;
    }

    public DigitalCounter(DigitalCounter other)
    {
        min = other.min;
        max = other.max;
        currentValue = other.currentValue;
    }
    ~DigitalCounter() { }

    public static DigitalCounter operator+(DigitalCounter main, double sum)
    {
        main.currentValue += sum;
        if (main.currentValue > main.max)
        {
            main.currentValue = main.min + (main.currentValue - main.max - 1);
        }
        return main;
    }

    public static DigitalCounter operator-(DigitalCounter main, double difference)
    {
        main.currentValue -= difference;
        if (main.currentValue < main.min)
        {
            main.currentValue = main.max - (main.min - main.currentValue - 1);
        }
        return main;
    }

    public static DigitalCounter operator++(DigitalCounter main)
    {
        main.currentValue++;
        if (main.currentValue > main.max)
        {
            main.currentValue = main.min;
        }
        return main;
    }

    public static DigitalCounter operator--(DigitalCounter main)
    {
        main.currentValue--;
        if (main.currentValue < main.min)
        {
            main.currentValue = main.min;
        }
        return main;
    }

    public int CompareTo(DigitalCounter other)
    {
        return this.MinValue.CompareTo(other.MinValue);
    }

    public override string ToString()

```

```

    {
        return " ran the Fastest race in " + min + " seconds and the Longest in " + max + "
seconds";
    }
}

```

Клас Result містить в собі дві колекції класу Stopwatch, який є похідним від класу DigitalCounter. Також він містить всі методи, які сказані в завданні, клас наслідує інтерфейс IEnumerable для роботи з масивом об'єктів (колекція List).

```

using System.Collections;
using System.Collections.Generic;
using System.Linq;
using System.Windows.Forms;

public class Result : IEnumerable<Stopwatch>
{
    private List<Stopwatch> stopwatches = new List<Stopwatch>();

    public List<Stopwatch> Participants => stopwatches;

    public Result(List<Stopwatch> timer)
    {
        stopwatches = timer;
    }

    public void AddSprinter(Stopwatch participant)
    {
        stopwatches.Add(participant);
    }

    public void DisplayParticipants()
    {
        string message = "";
        foreach (var participant in Participants)
        {
            message += participant.ToString();
        }
    }

    public double MinTime()
    {
        if (stopwatches.Count == 0)
        {
            MessageBox.Show("The list of stopwatches is empty :(", "Error!",
MessageBoxButtons.OK, MessageBoxIcon.Error);
            return 0;
        }
        return stopwatches.Min(sw => sw.Value);
    }

    public double MaxTime()
    {
        if (stopwatches.Count == 0)
        {
            MessageBox.Show("The list of stopwatches is empty :(", "Error!",
MessageBoxButtons.OK, MessageBoxIcon.Error);
            return 0;
        }
        return stopwatches.Max(sw => sw.Value);
    }

    public double AllAvgTime()
    {

```

```

        if (stopwatches.Count == 0)
        {
            MessageBox.Show("The list of stopwatches is empty :(", "Error!",
                MessageBoxButtons.OK, MessageBoxIcon.Error);
        }

        double sum = 0;
        foreach (var participant in stopwatches)
        {
            sum += participant.MinValue + participant.MaxValue;
        }

        double allAvgTime = sum / (stopwatches.Count * 2);
        return allAvgTime;
    }

    public List<Stopwatch> Top3()
    {
        stopwatches.Sort();
        List<Stopwatch> top3Winners = stopwatches.Take(3).ToList();

        return top3Winners;
    }

    public void DisplayWinners()
    {
        List<Stopwatch> top3Winners = Top3();
        string message = "";
        foreach (var winner in top3Winners)
        {
            message += winner.ToString() + "\n";
        }
    }

    public IEnumerator<Stopwatch> GetEnumerator()
    {
        for (int i = 0; i < stopwatches.Count(); i++)
        {
            yield return stopwatches[i];
        }
    }

    IEnumerator IEnumerable.GetEnumerator()
    {
        return GetEnumerator();
    }
}

```

Реалізований механізм пізнього зв'язування у даному фрагменті:

```

DigitalCounter digitalCounter = new DigitalCounter(16.2, 18.8);
Stopwatch stopwatch = new Stopwatch("Tommy", 17.1, 20.4);

listBox1.Items.Add("Digital Counter: " + digitalCounter.ToString());
listBox1.Items.Add("Stopwatch: " + stopwatch.ToString());

```

#### Demonstration of Late Binding Mechanism:

Digital Counter: ran the Fastest race in 16,2 seconds and the Longest in 18,8 seconds  
 Stopwatch: Sprinter Tommy: ran the Fastest race in 17,1 seconds and the Longest in 20

Забезпечене читання і запис даних для формування масиву об'єктів:

```
private void toolStripButton6_Click(object sender, EventArgs e)
{
    try
    {
        if (File.Exists(inputFile))
        {
            string json = File.ReadAllText(inputFile);
            var stopwatches = JsonConvert.DeserializeObject<List<Stopwatch>>(json);
            list = new Result(stopwatches);
            MessageBox.Show("The file has been Read Successfully :)", "Success!",
            MessageBoxButtons.OK, MessageBoxIcon.Information);
        }
        else
        {
            MessageBox.Show("File not found :", "Error!", MessageBoxButtons.OK,
            MessageBoxIcon.Error);
        }
    }
    catch (Exception ex)
    {
        MessageBox.Show($"An error occurred while deserializing: {ex.Message}",
        "Error!", MessageBoxButtons.OK, MessageBoxIcon.Error);
    }
}

private void toolStripButton5_Click(object sender, EventArgs e)
{
    try
    {
        if (list.Participants.Any())
        {
            string json = JsonConvert.SerializeObject(list.Participants);
            File.WriteAllText(inputFile, json);
            MessageBox.Show("The data has been successfully added to the file :)",
            "Success!", MessageBoxButtons.OK, MessageBoxIcon.Information);
        }
        else
        {
            MessageBox.Show("There are no sprinters to save!", "Empty List!",
            MessageBoxButtons.OK, MessageBoxIcon.Information);
        }
    }
    catch (Exception ex)
    {
        MessageBox.Show($"An error occurred while serializing: {ex.Message}",
        "Error!", MessageBoxButtons.OK, MessageBoxIcon.Error);
    }
}
```

Збереження та зчитування списку учасників з файлу, використовуючи JSON для серіалізації та десеріалізації. Використання JSON є простим та зручним способом зберігання даних у форматі, який легко читається як людьми, так і програмами.

# Аналіз результатів комп'ютерної реалізації програми

## C++

Введення даних про учасників змагань:

```
D:\C#\CourseworkOOP_Part_C++\x64\Debug\CourseworkOOP_Part_C++.exe

*----- Results of the International Athletics Competitions -----*

Select the source of the input data or other events:

1) Manual input (must enter data from the console)
2) Data from file (data is read from the input file)
3) Demonstration all constructors
4) Demonstration of operators overloading
5) Close program (Enter)

Input your operation: 1
Selected option 1: Manual input

*----- At this stage, you need to define the stopwatch readings -----*
*----- Decide on the start of the countdown and the end of the stopwatch -----*

Enter participant's name: Alex
Enter the minimum time of the stopwatch: 9.6
Enter the maximum time of the stopwatch: 11.2

Participant's data successfully recorded :)

*----- Do you want to continue or change data? -----*
*----- If yes - enter 'y' or 'Y', if not - enter any key! -----*          y
Enter participant's name: Sara
Enter the minimum time of the stopwatch: 19.2
Enter the maximum time of the stopwatch: 22.6

Participant's data successfully recorded :)

*----- Do you want to continue or change data? -----*
*----- If yes - enter 'y' or 'Y', if not - enter any key! -----*          y
Enter participant's name: Kate
Enter the minimum time of the stopwatch: 13.6
Enter the maximum time of the stopwatch: 17.9

Participant's data successfully recorded :)

*----- Do you want to continue or change data? -----*
*----- If yes - enter 'y' or 'Y', if not - enter any key! -----*          y
Enter participant's name: Andrew
Enter the minimum time of the stopwatch: 18.7
Enter the maximum time of the stopwatch: 21.4

Participant's data successfully recorded :)

*----- Do you want to continue or change data? -----*
*----- If yes - enter 'y' or 'Y', if not - enter any key! -----*          h
Select the source of the output data:
```



Вивід інформації зі всіма підрахунками:

```
Participant's data successfully recorded :)

*----- Do you want to continue or change data? -----*
*----- If yes - enter 'y' or 'Y', if not - enter any key! -----*      h
Select the source of the output data:

1) To the console
2) To a file
3) Close program

Input your operation: 1
Selected option 1: Output to the console

*----- At this stage, you see the race results for one or more participants -----*

Participant's name: Alex
Minimum race time: 9.6 seconds
Maximum race time: 11.2 seconds

Participant's name: Sara
Minimum race time: 19.2 seconds
Maximum race time: 22.6 seconds

Participant's name: Kate
Minimum race time: 13.6 seconds
Maximum race time: 17.9 seconds

Participant's name: Andrew
Minimum race time: 18.7 seconds
Maximum race time: 21.4 seconds

Minimum race time: 9.6 seconds, runner: Alex
Maximum race time: 22.6 seconds, runner: Sara

Average race time for all participants: 18.275 seconds

The results of the 3 winners:
Runner: Alex has best a time of race: 9.6 seconds
Runner: Kate has best a time of race: 13.6 seconds
Runner: Andrew has best a time of race: 18.7 seconds
```

Зчитування з файлу:

```
D:\C#\CourseworkOOP_Part_C++\x64\Debug\CourseworkOOP_Part_C++.exe

*----- Results of the International Athletics Competitions -----*

Select the source of the input data or other events:

1) Manual input (must enter data from the console)
2) Data from file (data is read from the input file)
3) Demonstration all constructors
4) Demonstration of operators overloading
5) Close program (Enter)

Input your operation: 2
Selected option 2: Data from file
The file was successfully read and closed :)
```

Вивід зчитаної інформації в консоль:

```
Input your operation: 2
Selected option 2: Data from file
The file was successfully read and closed :)

Select the source of the output data:

1) To the console
2) To a file
3) Close program

Input your operation: 1
Selected option 1: Output to the console

*----- At this stage, you see the race results for one or more participants -----*

Participant's name: Sasha
Minimum race time: 10.1 seconds
Maximum race time: 13.2 seconds

Participant's name: Igor
Minimum race time: 8.2 seconds
Maximum race time: 9.9 seconds

Participant's name: Vasyl
Minimum race time: 15.5 seconds
Maximum race time: 17.6 seconds

Participant's name: Denys
Minimum race time: 8.7 seconds
Maximum race time: 10.8 seconds

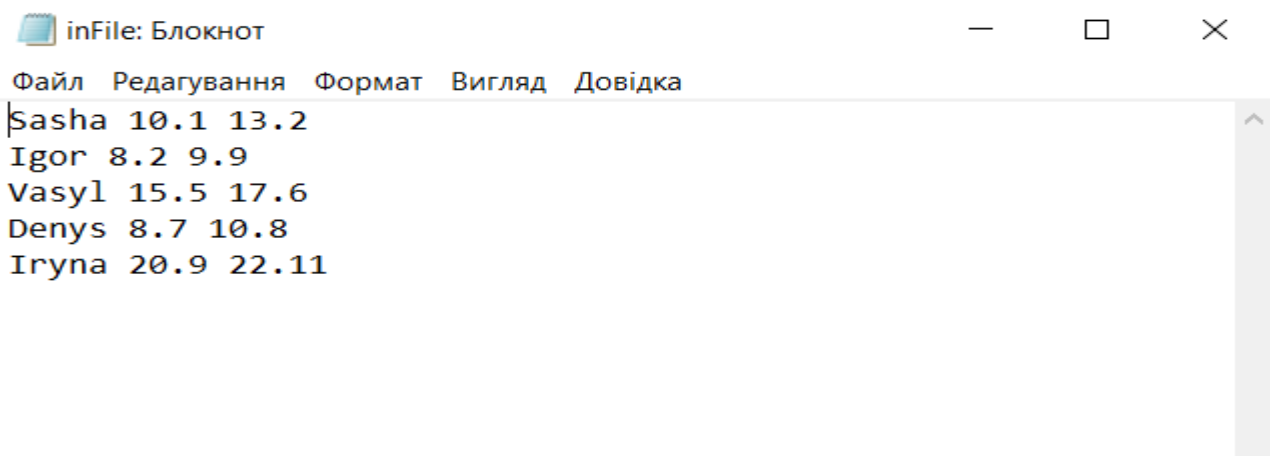
Participant's name: Iryna
Minimum race time: 20.9 seconds
Maximum race time: 22.11 seconds

Minimum race time: 8.2 seconds, runner: Igor
Maximum race time: 22.11 seconds, runner: Iryna

Average race time for all participants: 14.722 seconds

The results of the 3 winners:
Runner: Igor has best a time of race: 8.2 seconds
Runner: Denys has best a time of race: 8.7 seconds
Runner: Sasha has best a time of race: 10.1 seconds
```

Файл для зчитування даних:



Після вводу інформації обрано запис у файл:

```
Input your operation: 1
Selected option 1: Manual input

*----- At this stage, you need to define the stopwatch readings -----*
*----- Decide on the start of the countdown and the end of the stopwatch -----*

Enter participant's name: Michel
Enter the minimum time of the stopwatch: 12.4
Enter the maximum time of the stopwatch: 15.7

Participant's data successfully recorded :)

*----- Do you want to continue or change data? -----*
*----- If yes - enter 'y' or 'Y', if not - enter any key! -----*      y
Enter participant's name: Steven
Enter the minimum time of the stopwatch: 20.5
Enter the maximum time of the stopwatch: 24.8

Participant's data successfully recorded :)

*----- Do you want to continue or change data? -----*
*----- If yes - enter 'y' or 'Y', if not - enter any key! -----*      y
Enter participant's name: Iryna
Enter the minimum time of the stopwatch: 14.7
Enter the maximum time of the stopwatch: 18.2

Participant's data successfully recorded :)

*----- Do you want to continue or change data? -----*
*----- If yes - enter 'y' or 'Y', if not - enter any key! -----*      y
Enter participant's name: Alexa
Enter the minimum time of the stopwatch: 16.7
Enter the maximum time of the stopwatch: 20.1


Participant's data successfully recorded :)

*----- Do you want to continue or change data? -----*
*----- If yes - enter 'y' or 'Y', if not - enter any key! -----*      h
Select the source of the output data:

1) To the console
2) To a file
3) Close program

Input your operation: 2
Selected option 2: Output to file The file was successfully completed and closed :)
```

Вихідний файл для збереження результатів:

 outFile: Блокнот

Файл Редагування Формат Вигляд Довідка

```
Participant's name: Michel
Minimum race time: 12.4 seconds
Maximum race time: 15.7 seconds
Participant's name: Steven
Minimum race time: 20.5 seconds
Maximum race time: 24.8 seconds
Participant's name: Iryna
Minimum race time: 14.7 seconds
Maximum race time: 18.2 seconds
Participant's name: Alexa
Minimum race time: 16.7 seconds
Maximum race time: 20.1 seconds

Minimum race time: 12.4 seconds, runner: Michel
Maximum race time: 24.8 seconds, runner: Steven

Average race time for all participants: 19.7 seconds

The results of the 3 winners:
Runner: Michel has a best time of race: 12.4 seconds
Runner: Iryna has a best time of race: 14.7 seconds
Runner: Alexa has a best time of race: 16.7 seconds
```

Вивід інформації, як працюють всі мої конструктори:

```
Input your operation: 3
Selected option 3: Demonstration all constructors

*----- At this stage, you can see a demonstration of the constructors -----*

Default constructor: Minimum race time = 10sec, Maximum race time = 15sec
Constructor with parameters : Minimum race time = 15.5sec, Maximum race time = 18.7sec
Copy constructor: Minimum race time = 15.5sec, Maximum race time = 18.7sec
Move constructor: Minimum race time = 10sec, Maximum race time = 15sec
```

Демонструю роботу перевантажених операторів, у вигляді спірної ситуації:

```
Input your operation: 4
Selected option 4: Demonstration of operators overloading

*----- At this stage, you can see the change of race result for the disputed participant -----*

One of the participants has the following results: minimum = 8.4sec and maximum: 12.6sec time of the sprinter's race!
The judges wanted to choose the winner because he has the best minimum race time = 8.4 seconds!
But then it turned out that he started faster and 1 second was added to him, now his race time = 9.4 seconds!
He didn't like it and appealed, where he won himself 1 second, now it's his time again = 8.4 seconds!
But after the competition, organizers rewatched video and noticed delay, so added 3 seconds to the sprinter!
It's final result = 11.4 seconds, due to which the sprinter took one of the last places!
```

Завершую програму:

```
Input your operation: 5
Selected option 5: Close program

Thank you for your visit. Goodbye :)

D:\C#\CourseworkOOP_Part_C++\x64\Debug\CourseworkOOP_Part_C++.exe (process 10140) exited with code 1.
To automatically close the console when debugging stops, enable Tools->Options->Debugging->Automatically close the console when debugging stops.
Press any key to close this window . . .
```

## C#



Обираю кнопку «Добавити учасника», та заповнюю список спринтерів:

The screenshot shows a dialog box titled "Add Competition Participants". The dialog has a blue background. It contains three input fields with labels: "Enter the Competitor Name:" with the text "Alex", "Enter the Minimum Race Time:" with the text "9.3", and "Enter the Maximum Race Time:" with the text "12.8". Below these fields is an "Error Message:" label followed by an empty text box. At the bottom of the dialog, there are two buttons: "Continue Entering" and "Get Over".

Після заповнення учасників, повертаюся в головне меню та вибираю кнопку «Переглянути список» (зверху також представлений механізм пізнього зв'язування), за допомогою кнопки знаходжу трьох переможців змагань.

Viewing the List of Competition Sprinters

**Demonstration of Late Binding Mechanism:**  
Digital Counter: ran the Fastest race in 16,2 seconds and the Longest in 18,8 seconds  
Stopwatch: Sprinter Tommy: ran the Fastest race in 17,1 seconds and the Longest in 20

**List of Sprinters:**  
Sprinter Alex: ran the Fastest race in 9,3 seconds and the Longest in 12,8 seconds  
Sprinter Egor: ran the Fastest race in 11,5 seconds and the Longest in 14,8 seconds  
Sprinter Olena: ran the Fastest race in 18 seconds and the Longest in 20,1 seconds  
Sprinter Alexa: ran the Fastest race in 10,2 seconds and the Longest in 13,6 seconds

**Sprinters who won Prizes:**  
Sprinter Alex: ran the Fastest race in 9,3 seconds and the Longest in 12,8 seconds  
Sprinter Alexa: ran the Fastest race in 10,2 seconds and the Longest in 13,6 seconds  
Sprinter Egor: ran the Fastest race in 11,5 seconds and the Longest in 14,8 seconds  
Average Race Time of All Sprinters = 13,7875 seconds

**Find 3 Winners** **Get Over**

Після перегляду списку всіх учасників, вертаюся знову в меню, де обираю кнопку «Видалити учасника», вибравши учасника (за іменем), якого буде видалено зі списку:

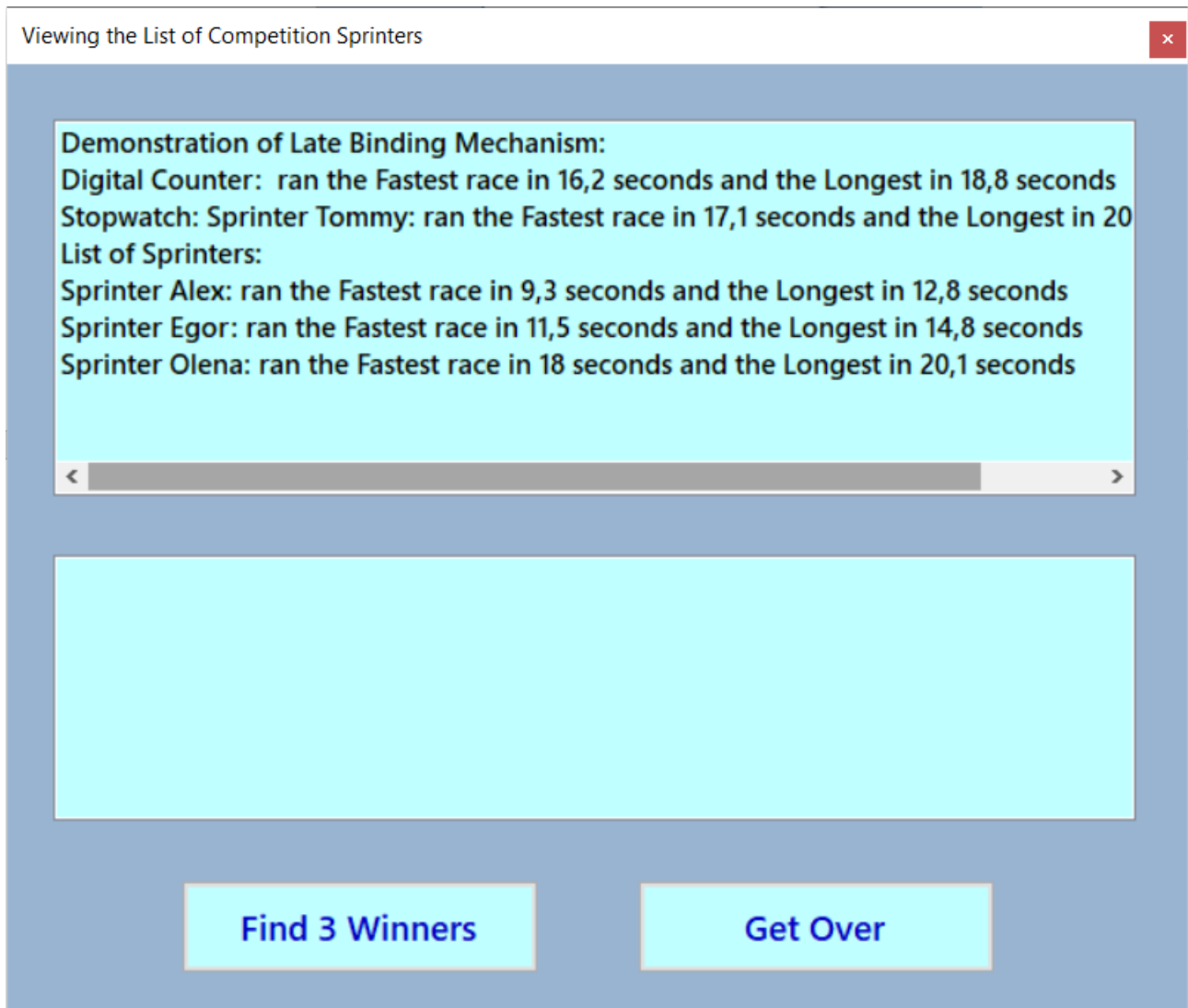
Delete Competition Participants

**Choose a Name:**

Alex  
Alexa  
Egor  
Olena

**Delete** **Exit**

Після видалення можна знову повернутися в перегляд списку та переконатися, що учасника з іменем «Alexa» - успішно видалено:



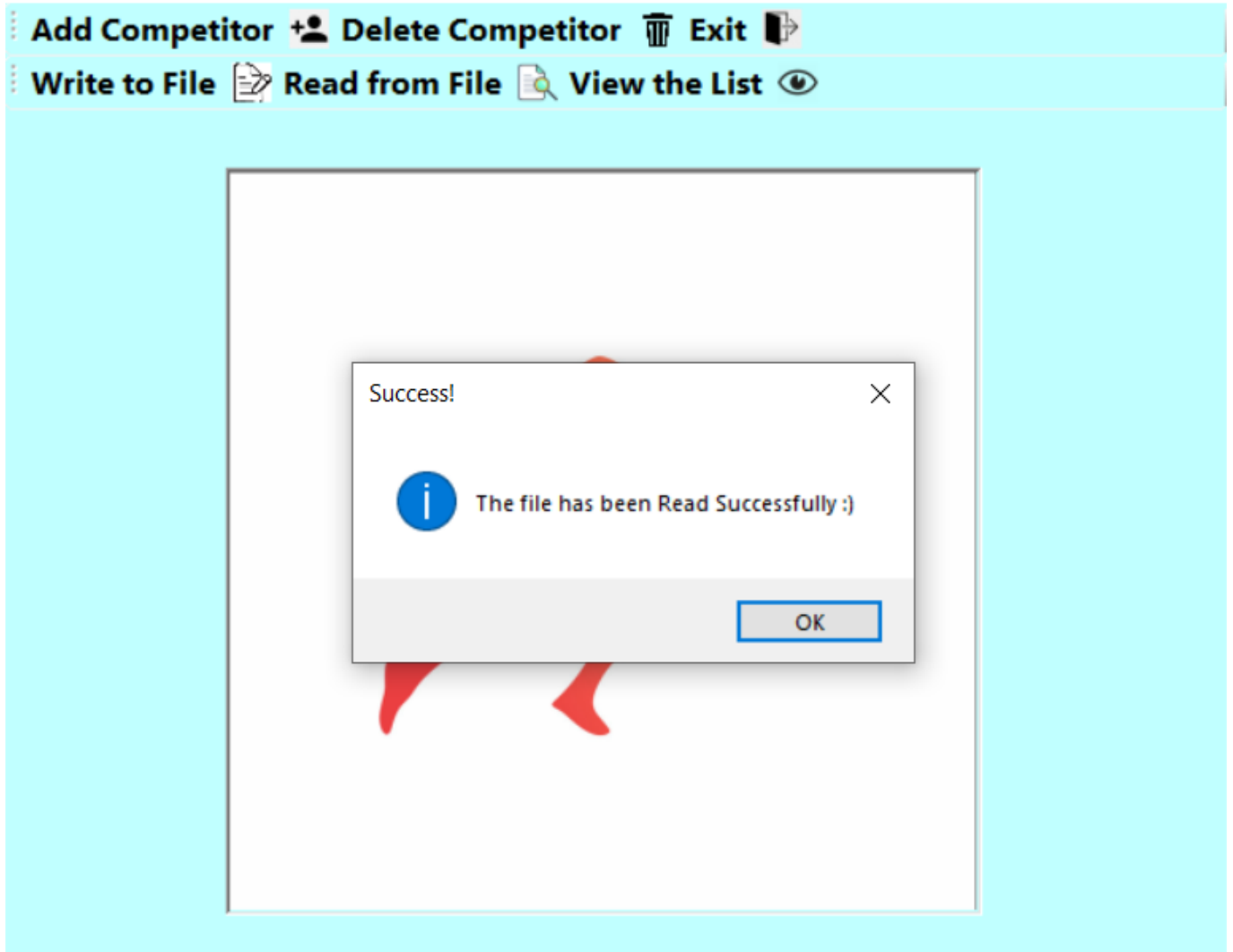
Щоб сереалізувати дані у файл, потрібно натиснути кнопку «Зберегти у файл» в головному меню, після чого дані, які записано в нашу колекцію, зберуться у вихідному файлі в JSON форматі, ось як це виглядає:

myFile.txt:

```
*myFile: Блокнот
Файл Редагування Формат Вигляд Довідка
[{"Running":false,"Name":"Vlad","MaxValue":14.5,"MinValue":11.2,"Value":11.2}
,{"Running":false,"Name":"Alexa","MaxValue":17.8,"MinValue":14.6,"Value":14.6}
,{"Running":false,"Name":"Andrew","MaxValue":11.2,"MinValue":9.9,"Value":9.9}
,{"Running":false,"Name":"Sara","MaxValue":20.2,"MinValue":18.7,"Value":18.7}
,{"Running":false,"Name":"Sasha","MaxValue":13.8,"MinValue":10.2,"Value":10.2}]
```

Для того, щоб десереалізувати дані з файлу, потрібно обрати кнопку «Зчитати з файлу» в головному меню, в подальшому з'явиться можливість працювати з колекцією.

Competition in Sprint Races





## Висновки

- Розроблений програмний продукт «Result» (результати секундоміру) використовує об'єктно-орієнтоване програмування для створення користувацьких типів даних (класів) і забезпечення їх взаємодії. Цей продукт дозволяє вносити дані у цифровий лічильник, здійснювати певні значення часу за різними критеріями та переглядати інформацію про заміри.
- Класи `DigitalCounter` і `Stopwatch` складають основу цього програмного продукту, де `Stopwatch` є похідним від `DigitalCounter`. Клас `Result` містить масив об'єктів класу `Stopwatch` та реалізує ітератор для роботи з цим масивом.
- Об'єктно-орієнтоване програмування дозволяє приховати деталі реалізації, використовувати повторний код та інтерпретувати виклики методів на етапі виконання. Використання принципів ООП, таких як інкапсуляція, поліморфізм та успадкування, дозволяє створювати більш масштабні проекти з кращим захистом від помилок.
- У подальшому розширення функціональності програми може включати в себе додавання нових можливостей, які полегшать ведення звітності та аналіз змагань. Такий продукт може слугувати як основа для розробки більш складних та функціональних програмних продуктів у майбутньому.

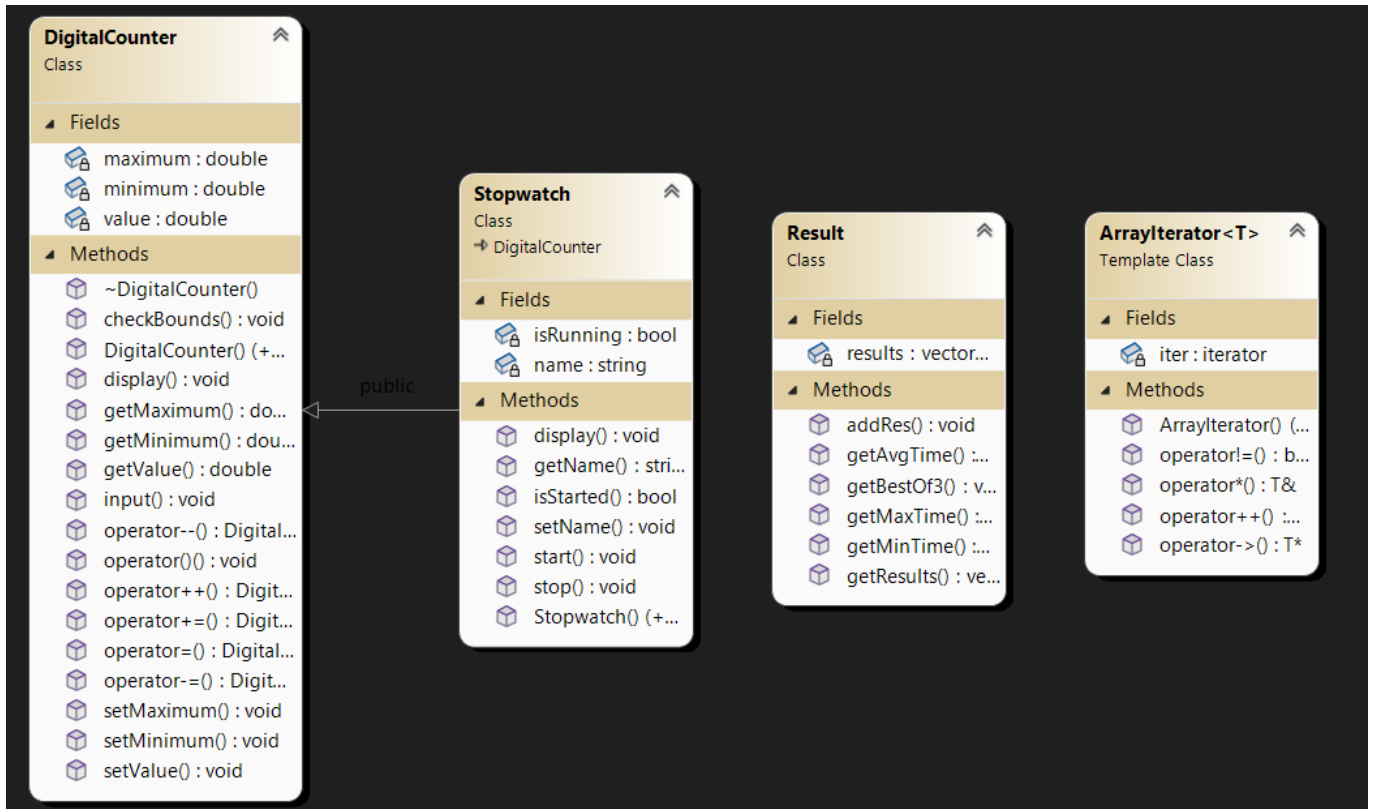
## Список використаних джерел

- 1) <https://www.bestprog.net/uk/2022/09/07/c-overloading-of-operators-general-information-overloading-of-unary-operators-ua/> - Перевантаження операторів.
- 2) [https://elearning.sumdu.edu.ua/free\\_content/lectured:035d976fe705881ad71ba486aa8d45ac0b3ce889/20200921190923//1799464/index.html](https://elearning.sumdu.edu.ua/free_content/lectured:035d976fe705881ad71ba486aa8d45ac0b3ce889/20200921190923//1799464/index.html) - ООП C++
- 3) <https://acode.com.ua/urok-215-potoky-vvodu-i-vyvodu/> - Поточкові операції.
- 4) <https://learn.microsoft.com/uk-ua/cpp/standard-library/cpp-standard-library-reference?view=msvc-140> – STL.
- 5) <https://www.guru99.com/uk/c-sharp-serialization.html> - Серіалізація та десеріалізація в C#
- 6) <https://www.youtube.com/@SimpleCodeIT> – Ютуб канал про програмування.

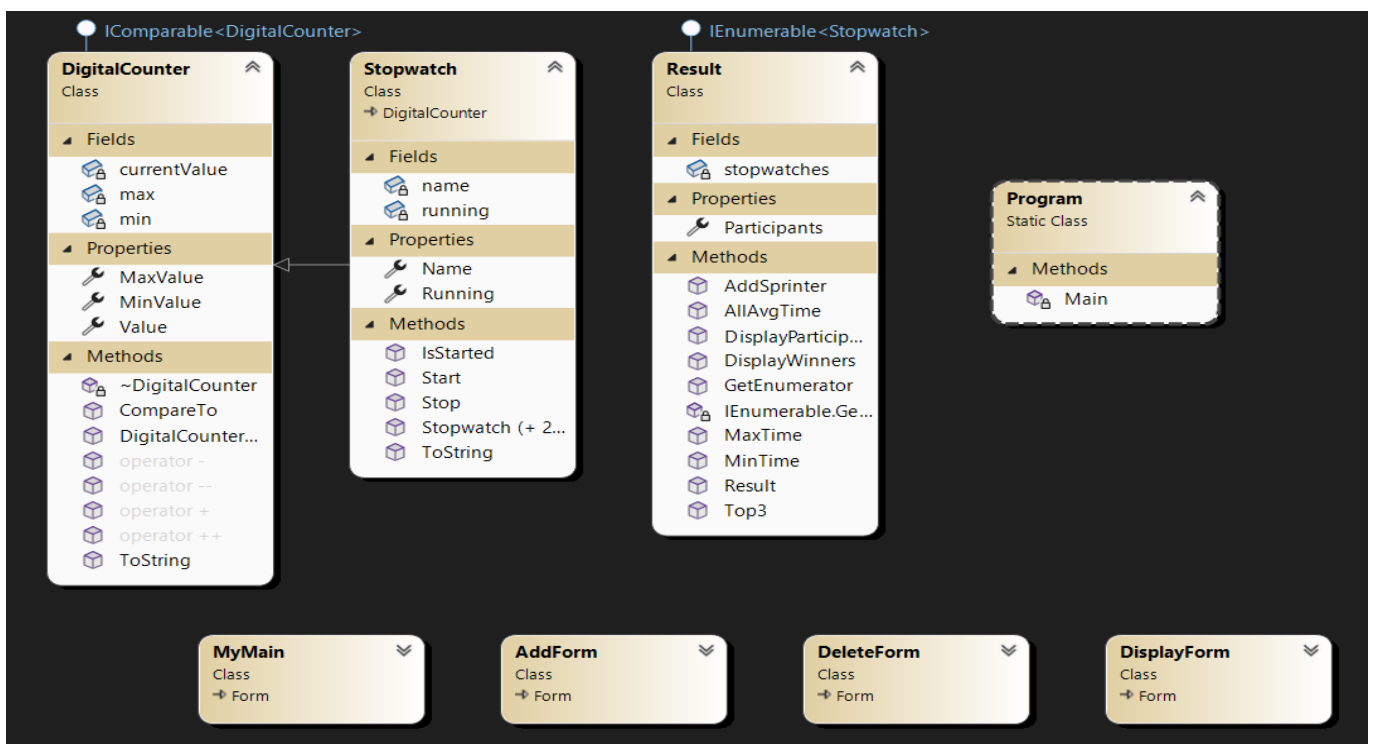
## Додатки

### Додаток А. Діаграма класів

C++



C#



## Додаток В. Код програмного застосування

### C++

DigitalCounter.h:

```
#pragma once

#include <iostream>

class DigitalCounter
{
private:
    // Possible counter values
    double value;
    double minimum;
    double maximum;

public:
    // The default constructor
    DigitalCounter();
    // Constructor specifying the minimum and maximum values
    DigitalCounter(double min, double max);
    // Copy constructor
    DigitalCounter(const DigitalCounter& other);
    // Move constructor
    DigitalCounter(DigitalCounter&& other);
    // Destructor
    virtual ~DigitalCounter();

    // Setting values
    void setMinimum(double min);
    void setMaximum(double max);
    void setValue(double value);

    // Check for a range
    void checkBounds();

    // Getting the values
    double getValue() const;
    double getMinimum() const;
    double getMaximum() const;

    // Operator () to set data field values
    void operator()(double newValue, double newMinimum, double newMaximum)
    {
        value = newValue;
        minimum = newMinimum;
        maximum = newMaximum;
        checkBounds();
    }

    // Operators overloading
    DigitalCounter& operator= (const DigitalCounter& other);
    DigitalCounter& operator= (DigitalCounter&& other);
    DigitalCounter& operator++();
    DigitalCounter& operator++(int);
    DigitalCounter& operator--();
    DigitalCounter& operator--(int);
    DigitalCounter& operator+=(int increment);
    DigitalCounter& operator-=(int decrement);
```

```

// Virtual method for outputting and inputting the counter value
virtual void display(std::ostream& os) const;
virtual void input(std::istream& is);

// Friendly functions for outputting and inputting the counter value to the stream
friend std::ostream& operator << (std::ostream& os, const DigitalCounter& counter);
friend std::istream& operator >> (std::istream& is, DigitalCounter& counter);
};

```

DigitalCounter.cpp:

```

#include "DigitalCounter.h"

DigitalCounter::DigitalCounter() : minimum(10.0), maximum(15.0), value(minimum)
{
    checkBounds();
}

DigitalCounter::DigitalCounter(double min, double max) : minimum(min), maximum(max),
value(min)
{
    checkBounds();
}

DigitalCounter::DigitalCounter(const DigitalCounter& other) : minimum(other.minimum),
maximum(other.maximum), value(other.value)
{
    checkBounds();
}

DigitalCounter::DigitalCounter(DigitalCounter&& other) : minimum(other.minimum),
maximum(other.maximum), value(other.value)
{
    other.minimum = 0;
    other.maximum = 0;
    other.value = 0;
    checkBounds();
}

DigitalCounter::~DigitalCounter() {}

void DigitalCounter::setMinimum(double min)
{
    minimum = min;
    checkBounds();
}

void DigitalCounter::setMaximum(double max)
{
    maximum = max;
    checkBounds();
}

void DigitalCounter::setValue(double value)
{
    if (value <= maximum && value >= minimum)
    {
        this->value = value;
    }
    else if (value < minimum)
    {
        this->value = minimum;
    }
}

```

```

    }
    else
    {
        this->value = maximum;
    }
    checkBounds();
}

double DigitalCounter::getValue() const
{
    return value;
}

double DigitalCounter::getMinimum() const
{
    return minimum;
}

double DigitalCounter::getMaximum() const
{
    return maximum;
}

void DigitalCounter::checkBounds()
{
    if (value < minimum)
    {
        value = minimum;
    }
    if (value > maximum)
    {
        value = maximum;
    }
}

DigitalCounter& DigitalCounter::operator=(const DigitalCounter& other)
{
    if (this != &other)
    {
        minimum = other.minimum;
        maximum = other.maximum;
        value = other.value;
    }
    return *this;
}

DigitalCounter& DigitalCounter::operator=(DigitalCounter&& other)
{
    if (this != &other)
    {
        minimum = other.minimum;
        maximum = other.maximum;
        value = other.value;
        other.minimum = 0;
        other.maximum = 0;
        other.value = 0;
    }
    return *this;
}

DigitalCounter& DigitalCounter::operator++()
{
    ++value;
    return *this;
}

```

```

DigitalCounter DigitalCounter::operator++(int)
{
    DigitalCounter temp(*this);
    temp.value = value + 1;
    return temp;
}

DigitalCounter& DigitalCounter::operator--()
{
    --value;
    return *this;
}

DigitalCounter DigitalCounter::operator--(int)
{
    DigitalCounter temp(*this);
    temp.value = value - 1;
    return temp;
}

DigitalCounter& DigitalCounter::operator+=(int increment)
{
    value += increment;
    return *this;
}

DigitalCounter& DigitalCounter::operator-=(int decrement)
{
    value -= decrement;
    return *this;
}

void DigitalCounter::display(std::ostream& os) const
{
    os << " Current value: " << getValue() << std::endl;
}

void DigitalCounter::input(std::istream& is)
{
    double newValue;
    is >> newValue;
    if (newValue < minimum)
    {
        newValue = minimum;
    }
    else if (newValue > maximum)
    {
        newValue = maximum;
    }
    value = newValue;
}

std::ostream& operator<<(std::ostream& os, const DigitalCounter& counter)
{
    counter.display(os);
    return os;
}

std::istream& operator>>(std::istream& is, DigitalCounter& counter)
{
    counter.input(is);
    return is;
}

```

## Stopwatch.h:

```
#pragma once
#include "DigitalCounter.h"

class Stopwatch : public DigitalCounter
{
private:
    bool isRunning; // Flag indicating whether the stopwatch is running
    std::string name;

public:
    // Default constructor
    Stopwatch();
    // Constructor with parameters
    Stopwatch(std::string name, double min, double max);

    void start(); // Start the stopwatch
    void stop(double newValue); // Stop the stopwatch
    bool isStarted() const; // Check if the stopwatch is running
    void display(std::ostream& os) const override; // Display the stopwatch value

    // Methods for setting and getting the participant's name
    void setName(const std::string& newName);
    std::string getName() const;
};
```

## Stopwatch.cpp:

```
#include "Stopwatch.h"

Stopwatch::Stopwatch() : DigitalCounter(), isRunning(false) {}

Stopwatch::Stopwatch(std::string name, double min, double max) : DigitalCounter(min, max),
isRunning(false), name(name) {}

void Stopwatch::start()
{
    isRunning = true;
}

void Stopwatch::stop(double newValue)
{
    if (isRunning)
    {
        isRunning = false;
        setValue(newValue);
    }
}

bool Stopwatch::isStarted() const
{
    return isRunning;
}

void Stopwatch::display(std::ostream& os) const
{
    if (isRunning)
    {
        os << "Running";
    }
}
```



```

    }
    else
    {
        DigitalCounter::display(os);
    }
}

void Stopwatch::setName(const std::string& newName)
{
    name = newName;
}

std::string Stopwatch::getName() const
{
    return name;
}

```

## Result.h:

```

#pragma once
#include "Stopwatch.h"
#include "ArrayIterator.h"
#include <algorithm>

class Result
{
private:
    // Vector to store race results
    std::vector<Stopwatch> results;
public:
    // Events with the result of the race
    void addRes(const Stopwatch& result);
    Stopwatch getMinTime() const;
    Stopwatch getMaxTime() const;
    double getAvgTime() const;

    // Vector to get all race results and three winners
    std::vector<Stopwatch> getResults() const;
    std::vector<Stopwatch> getBestOf3();
};

```

## Result.cpp:

```

#include "Result.h"

void Result::addRes(const Stopwatch& result)
{
    results.push_back(result);
}

Stopwatch Result::getMinTime() const
{
    if (results.empty())
    {
        throw std::runtime_error("There are no results, you need to fill in the race data!");
    }

    // Finding the element with the smallest value in the vector
    auto minRes = std::min_element(cbegin(results), cend(results),
        [](const Stopwatch& a, const Stopwatch& b)
        {

```

```

        return a.getValue() < b.getValue();
    });

    return (*minRes);
}

Stopwatch Result::getMaxTime() const
{
    if (results.empty())
    {
        throw std::runtime_error("There are no results, you need to fill in the race
data!");
    }

    // Finding the element with the largest value in the vector
    auto maxRes = std::max_element(cbegin(results), cend(results),
        [](const Stopwatch& a, const Stopwatch& b)
        {
            return a.getValue() < b.getValue();
        });

    return (*maxRes);
}

double Result::getAvgTime() const
{
    if (results.empty())
    {
        throw std::runtime_error("There are no results, you need to fill in the race
data!");
    }

    double sum = 0;
    // Summation of run time values in a vector
    for (const auto& stopwatch : results)
    {
        sum += stopwatch.getValue();
    }

    return sum / results.size();
}

std::vector<Stopwatch> Result::getResults() const
{
    return results;
}

std::vector<Stopwatch> Result::getBestOf3()
{
    std::vector<Stopwatch> myTop;

    if (results.empty())
    {
        return myTop;
    }

    // Using a standard iterator for sorting
    std::sort(results.begin(), results.end(), [](const Stopwatch& a, const Stopwatch& b)
    {
        return a.getValue() < b.getValue();
    });

    // Save the third smallest result
    auto thirdMinIter = results.begin() + 2;

```

```

    ArrayIterator<Stopwatch> beginIter(results.begin());
    ArrayIterator<Stopwatch> endIter(results.end());

    // Add these values to a new vector
    for (ArrayIterator<Stopwatch> iter = beginIter; iter != endIter; ++iter)
    {
        if (iter->getValue() <= thirdMinIter->getValue())
        {
            myTop.push_back(*iter);
        }
    }

    return myTop;
}

```

ArrayIterator.h:

```

#pragma once
#include <vector>

template <typename T>

class ArrayIterator
{
private:
    // An iterator that points to the current element of the vector
    typename std::vector<T>::iterator iter;

public:
    // Default constructor
    ArrayIterator() = default;
    // Constructor that initializes the iterator with the specified value
    ArrayIterator(typename std::vector<T>::iterator iter) : iter(iter) {}

    // Overloaded dereference operator, returns a reference to the current element
    T& operator*()
    {
        return *iter;
    }

    // Overloaded prefix increment operator, moves the iterator to the next element
    ArrayIterator<T>& operator++()
    {
        ++iter;
        return *this;
    }

    // Overloaded arrow operator, returns a pointer to the current element
    T* operator->()
    {
        return &(*iter);
    }

    // Overloaded comparison operator for inequality, compares two iterators
    bool operator!=(const ArrayIterator<T>& other) const
    {
        return iter != other.iter;
    }
};

```

## Main.cpp:

```
#include "DigitalCounter.h"
#include "Stopwatch.h"
#include "Result.h"
#include <sstream>
#include <fstream>
#include <stdexcept>

Result result{};
Stopwatch stopwatch{};

DigitalCounter* res = nullptr;

void readFromConsole()
{
    std::cout << "\n\n *----- At this stage, you need to define the
stopwatch readings -----*\n";

    std::cout << " *----- Decide on the start of the countdown and the end of the
stopwatch -----*\n\n";

    while (true)
    {
        double min = 0, max = 0;
        std::string input;
        std::string name;

        bool validName = false;
        bool validTime = false;

        while (!validName)
        {
            std::cout << " Enter participant's name: ";
            std::getline(std::cin, name);

            // Check for the correctness of entering the name
            bool letters = false;
```

```

    for (char c : name)
    {
        if (!std::isalpha(c))
        {
            letters = true;
            break;
        }
    }

    if (name.empty() || name.size() < 3 || name.find(' ') != std::string::npos ||
letters)
    {
        std::cout << " Invalid input. Name must contain at least 3 letters, have no
spaces, and consist only of letters!\n";
    }
    else
    {
        validName = true;
    }
}

while (!validTime)
{
    std::cout << " Enter the minimum time of the stopwatch: ";
    std::getline(std::cin, input);

    // Checking the correctness of entering the minimum time
    bool validInput = !input.empty() && std::all_of(input.begin(), input.end(),
[](char c)
    {
        return std::isdigit(c) || c == '.';
    });

    if (!validInput)
    {
        std::cout << " Invalid input. Enter only real numbers!\n";
    }
}

```

```

else
{
    min = std::stod(input);

    std::cout << " Enter the maximum time of the stopwatch: ";
    std::getline(std::cin, input);

    // Checking the correctness of entering the maximum time
    validInput = !input.empty() && std::all_of(input.begin(), input.end(),
[](char c)
    {
        return std::isdigit(c) || c == '.';
    });

    if (!validInput)
    {
        std::cout << " Invalid input. Enter only real numbers!\n";
    }
    else
    {
        max = std::stod(input);

        if (min >= max)
        {
            std::cout << " Invalid input. Start time should be less than end
time!\n";
        }
        else
        {
            validTime = true;
        }
    }
}

// Create, start, and stop the Stopwatch object
auto sv = Stopwatch(name, min, max);

```

```

        sv.start();
        sv.stop(max);

        // Assign the address of the Stopwatch object to the res pointer and add the result
to Result
        res = &sv;
        if (dynamic_cast<Stopwatch*>(res) != nullptr)
        {
            result.addRes(*dynamic_cast<Stopwatch*>(res));
        }

        std::cout << "\n Participant's data successfully recorded :)\n\n";

        std::cout << " *----- Do you want to continue or change data? -----
-----*\n";
        std::cout << " *----- If yes - enter 'y' or 'Y', if not - enter any key! --
-----*\t";

        std::getline(std::cin, input);
        if (input != "y" && input != "Y")
        {
            break;
        }
    }
}

void readFromFile(const std::string& filename)
{
    std::ifstream in(filename);
    if (!in.is_open())
    {
        throw std::runtime_error(" Error opening file: " + filename);
    }
    else
    {
        std::string name;
        double min, max;

```

```

while (in >> name >> min >> max)
{
    auto sv = Stopwatch(name, min, max);
    sv.start();
    sv.stop(max);

    res = &sv;
    if (dynamic_cast<Stopwatch*>(res) != nullptr)
    {
        result.addRes(*dynamic_cast<Stopwatch*>(res));
    }
}

std::cout << " The file was successfully read and closed :)\n\n";
in.close();
}

}

void printResults()
{
    try
    {
        std::cout << "\n\n *----- At this stage, you see the race results for one
or more participants -----*\n\n";

        auto allResults = result.getResults();
        Stopwatch minTime = result.getMinTime();
        Stopwatch maxTime = result.getMaxTime();

        for (const auto& stopwatch : allResults)
        {
            std::cout << " Participant's name: " << stopwatch.getName() << std::endl;
            std::cout << " Minimum race time: " << stopwatch.getMinimum() << " seconds\n";
            std::cout << " Maximum race time: " << stopwatch.getMaximum() << "
seconds\n\n";
        }
    }
}

```



```

        std::cout << "\n Minimum race time: " << minTime.getMinimum() << " seconds" << ",
runner: " << minTime.getName() << std::endl;

        std::cout << " Maximum race time: " << maxTime.getMaximum() << " seconds" << ",
runner: " << maxTime.getName() << std::endl;

        if (allResults.size() >= 2)
        {
            std::cout << "\n Average race time for all participants: " <<
result.getAvgTime() << " seconds\n";
        }
        else
        {
            std::cout << "\n In order to calculate the average time of the participants'
race, there must be two or more of them!\n\n";
        }

        if (allResults.size() >= 4)
        {
            std::cout << "\n The results of the 3 winners: \n";
            for (auto time : result.getBestOf3())
            {
                std::cout << " Runner: " << time.getName() << " has best a time of race: "
<< time.getMinimum() << " seconds\n";
            }
            std::cout << "\n\n";
        }
        else
        {
            std::cout << " To determine the three winners of the competition, there must be
four or more applicants!\n\n";
        }
    }
    catch (const std::exception& e)
    {
        std::cout << " Error: " << e.what() << std::endl;
    }
}

```

```

void writeToFile(const std::string& filename)

```

```

{
    std::ofstream out(filename);
    if (!out.is_open())
    {
        throw std::runtime_error(" ,Error opening file: " + filename);
    }
    else
    {
        auto allResults = result.getResults();
        Stopwatch minTime = result.getMinTime();
        Stopwatch maxTime = result.getMaxTime();

        for (const auto& stopwatch : allResults)
        {
            out << " Participant's name: " << stopwatch.getName() << std::endl;
            out << " Minimum race time: " << stopwatch.getMinimum() << " seconds\n";
            out << " Maximum race time: " << stopwatch.getMaximum() << " seconds\n";
        }

        out << "\n Minimum race time: " << minTime.getMinimum() << " seconds" << ", runner: "
        << minTime.getName() << std::endl;

        out << " Maximum race time: " << maxTime.getMaximum() << " seconds" << ", runner: "
        << maxTime.getName() << std::endl;

        if (allResults.size() >= 2)
        {
            out << "\n Average race time for all participants: " << result.getAvgTime() <<
            " seconds\n";
        }
        else
        {
            out << "\n In order to calculate the average time of the participants' race,
there must be two or more of them!\n\n";
        }

        if (allResults.size() >= 4)
        {
            out << "\n The results of the 3 winners: \n";

```

```

        for (auto time : result.getBestOf3())
        {
            out << " Runner: " << time.getName() << " has a best time of race: " <<
time.getMinimum() << " seconds\n";
        }
        out << "\n\n";
    }
    else
    {
        out << " To determine the three winners of the competition, there must be four
or more applicants!\n\n";
    }
    std::cout << " The file was successfully completed and closed :)\n\n";
    out.close();
}
}

void demonstrateConstructors()
{
    std::cout << "\n *----- At this stage, you can see a demonstration of the
constructors -----*\n\n";

    DigitalCounter counter1;

    std::cout << " Default constructor: Minimum race time = " << counter1.getMinimum() <<
"sec, Maximum race time = " << counter1.getMaximum() << "sec\n";

    DigitalCounter counter2(15.5, 18.7);

    std::cout << " Constructor with parameters : Minimum race time = " <<
counter2.getMinimum() << "sec, Maximum race time = " << counter2.getMaximum() << "sec\n";

    DigitalCounter counter3(counter2);

    std::cout << " Copy constructor: Minimum race time = " << counter3.getMinimum() <<
"sec, Maximum race time = " << counter3.getMaximum() << "sec\n";

    DigitalCounter counter4(std::move(counter1));

    std::cout << " Move constructor: Minimum race time = " << counter4.getMinimum() <<
"sec, Maximum race time = " << counter4.getMaximum() << "sec\n\n";
}

```

```

void demonstrateOperators()
{
    std::cout << "\n *----- At this stage, you can see the change of race result for
the disputed participant -----*\n\n";

    DigitalCounter counter(8.4, 12.6);

    std::cout << " One of the participants has the following results: minimum = " <<
counter.getMinimum() << "sec and maximum: " << counter.getMaximum() << "sec time of the
sprinter's race!\n";

    std::cout << " The judges wanted to choose the winner because he has the best minimum
race time = " << counter.getMinimum() << " seconds!\n";

    std::cout << " But then it turned out that he started faster and 1 second was added to
him, now his race time = " << (++counter).getValue() << " seconds!\n";

    std::cout << " He didn't like it and appealed, where he won himself 1 second, now it's
his time again = " << (counter--).getValue() << " seconds!\n";

    std::cout << " But after the competition, organizers rewatched video and noticed delay,
so added 3 seconds to the sprinter!\n";

    std::cout << " It's final result = " << (counter += 2).getValue() << " seconds, due to
which the sprinter took one of the last places!\n\n";
}

int main()
{
    std::cout << "\n *----- Results of the International Athletics
Competitions -----*\n\n";

    bool invalidChoice = false;

    while (!invalidChoice)
    {
        try
        {
            std::cout << "\n Select the source of the input data or other events:\n\n";
            std::cout << " 1) Manual input (must enter data from the console)\n";
            std::cout << " 2) Data from file (data is read from the input file)\n";
            std::cout << " 3) Demonstration all constructors\n";
            std::cout << " 4) Demonstration of operators overloading\n";
            std::cout << " 5) Close program (Enter)\n\n";
            std::cout << " Input your operation: ";

```

```

std::string input;
std::getline(std::cin, input);

// Check input for a number
bool isDigit = true;
for (char c : input)
{
    if (!std::isdigit(c))
    {
        isDigit = false;
        break;
    }
}

if (!isDigit)
{
    std::cout << " Invalid input. Please enter a number!\n\n";
    continue;
}

int choice1 = std::stoi(input);

switch (choice1)
{
case 1:
{
    std::cout << " Selected option 1: Manual input\n";
    readFromConsole();
    break;
}
case 2:
{
    std::cout << " Selected option 2: Data from file\n";
    readFromFile("inFile.txt");
    break;
}
}

```

```

case 3:
{
    std::cout << " Selected option 3: Demonstration all constructors\n";
    demonstrateConstructors();
    continue;
}
case 4:
{
    std::cout << " Selected option 4: Demonstration of operators
overloading\n";
    demonstrateOperators();
    continue;
}
case 5:
{
    std::cout << " Selected option 5: Close program\n";
    std::cout << "\n Thank you for your visit. Goodbye :)\n\n";
    return 1;
}
default:
    std::cout << " Invalid choice. Please enter a valid option!\n\n";
    continue;
}

std::cout << " Select the source of the output data:\n\n";
std::cout << " 1) To the console\n";
std::cout << " 2) To a file\n";
std::cout << " 3) Close program\n\n";
std::cout << " Input your operation: ";

std::getline(std::cin, input);

for (char c : input)
{
    if (!std::isdigit(c))
    {
        isDigit = false;
    }
}

```

```

        break;
    }
}

if (!isDigit)
{
    std::cout << " Invalid input. Please enter a number!\n\n";
    continue;
}

int choice2 = std::stoi(input);

switch (choice2)
{
case 1:
{
    std::cout << " Selected option 1: Output to the console\n";
    printResults();
    break;
}
case 2:
{
    std::cout << " Selected option 2: Output to file";
    writeToFile("outFile.txt");
    break;
}
case 3:
{
    std::cout << " Selected option 3: Close program\n";
    std::cout << "\n Thank you for your visit. Goodbye :)\n\n";
    return 1;
}
default:
    std::cout << " Invalid choice. Please enter a valid option!\n\n";
    continue;
}

```

```

    }
    catch (const std::exception& e)
    {
        std::cout << " Error: " << e.what() << std::endl;
        return 1;
    }
}

return 0;
}

```

## C#

DigitalCounter.cs:

```

using System;
using System.Windows.Forms;

public class DigitalCounter : IComparable<DigitalCounter>
{
    private double min;
    private double max;
    private double currentValue;

    public double MaxValue
    {
        get { return max; }
        set
        {
            if (value <= min)
            {
                MessageBox.Show("Maximum value must be greater than min value!", "Error
:(", MessageBoxButtons.OK, MessageBoxIcon.Error);
            }
            else
            {
                max = value;
            }
        }
    }

    public double MinValue
    {
        get { return min; }
        set { min = value; }
    }

    public double Value
    {
        get { return currentValue; }
        set { currentValue = value; }
    }
}

```



```

public DigitalCounter()
{
    min = 11.5;
    max = 15.5;
    currentValue = 60.0;
}

public DigitalCounter(double minValue, double maxValue)
{
    min = minValue;
    max = maxValue;
    currentValue = minValue;
}

public DigitalCounter(DigitalCounter other)
{
    min = other.min;
    max = other.max;
    currentValue = other.currentValue;
}
~DigitalCounter() { }

public static DigitalCounter operator +(DigitalCounter main, double sum)
{
    main.currentValue += sum;
    if (main.currentValue > main.max)
    {
        main.currentValue = main.min + (main.currentValue - main.max - 1);
    }
    return main;
}

public static DigitalCounter operator -(DigitalCounter main, double difference)
{
    main.currentValue -= difference;
    if (main.currentValue < main.min)
    {
        main.currentValue = main.max - (main.min - main.currentValue - 1);
    }
    return main;
}

public static DigitalCounter operator ++(DigitalCounter main)
{
    main.currentValue++;
    if (main.currentValue > main.max)
    {
        main.currentValue = main.min;
    }
    return main;
}

public static DigitalCounter operator --(DigitalCounter main)
{
    main.currentValue--;
    if (main.currentValue < main.min)
    {
        main.currentValue = main.min;
    }
    return main;
}

public int CompareTo(DigitalCounter other)
{

```

```

        return this.MinValue.CompareTo(other.MinValue);
    }

    public override string ToString()
    {
        return " ran the Fastest race in " + min + " seconds and the Longest in " + max + "
seconds";
    }
}

```

Stopwatch.cs:

```

using System.Windows.Forms;
public class Stopwatch : DigitalCounter
{
    private bool running;
    private string name;

    public bool Running
    {
        get { return running; }
        private set { running = value; }
    }

    public string Name
    {
        get { return name; }
        set
        {
            if (string.IsNullOrEmpty(value))
            {
                MessageBox.Show("Name cannot be null or whitespace :(", "Error",
MessageBoxButtons.OK, MessageBoxIcon.Error);
            }
            else
            {
                name = value;
            }
        }
    }

    public Stopwatch() : base(0, 0)
    {
        running = false;
        name = "Sasha";
    }

    public Stopwatch(string name, double minValue, double maxValue) : base(minValue,
maxValue)
    {
        running = false;
        this.name = name;
    }

    public Stopwatch(Stopwatch other) : base(other)
    {
        running = other.running;
        name = other.name;
    }

    public void Start()
    {

```

```

        running = true;
    }

    public void Stop()
    {
        running = false;
    }

    public bool IsStarted()
    {
        return running;
    }

    public override string ToString()
    {
        return "Sprinter " + name + ":" + base.ToString();
    }
}

```

Result.cs:

```

using Newtonsoft.Json;
using System.Collections;
using System.Collections.Generic;
using System.Linq;
using System.Windows.Forms;

[JsonObject(MemberSerialization.OptIn)]
public class Result : IEnumerable<Stopwatch>
{
    [JsonProperty]
    private List<Stopwatch> stopwatches = new List<Stopwatch>();

    [JsonProperty]
    public List<Stopwatch> Participants => stopwatches;

    public Result()
    {
        stopwatches = new List<Stopwatch>();
    }

    public Result(List<Stopwatch> timer)
    {
        stopwatches = timer;
    }

    public void AddSprinter(Stopwatch participant)
    {
        stopwatches.Add(participant);
    }

    public void DisplayParticipants()
    {
        string message = "";
        foreach (var participant in stopwatches)
        {
            message += participant.ToString();
        }
    }

    public double MinTime()
    {
        if (stopwatches.Count == 0)

```

```

        {
            MessageBox.Show("The list of stopwatches is empty :", "Error!",
MessageBoxButtons.OK, MessageBoxIcon.Error);
            return 0;
        }
        return stopwatches.Min(sw => sw.MinValue);
    }

    public double MaxTime()
    {
        if (stopwatches.Count == 0)
        {
            MessageBox.Show("The list of stopwatches is empty :", "Error!",
MessageBoxButtons.OK, MessageBoxIcon.Error);
            return 0;
        }
        return stopwatches.Max(sw => sw.MaxValue);
    }

    public double AllAvgTime()
    {
        if (stopwatches.Count == 0)
        {
            MessageBox.Show("The list of stopwatches is empty :", "Error!",
MessageBoxButtons.OK, MessageBoxIcon.Error);
        }

        double sum = 0;
        foreach (var participant in stopwatches)
        {
            sum += participant.MinValue + participant.MaxValue;
        }

        double allAvgTime = sum / (stopwatches.Count * 2);
        return allAvgTime;
    }

    public List<Stopwatch> Top3()
    {
        stopwatches.Sort();
        List<Stopwatch> top3Winners = stopwatches.Take(3).ToList();

        return top3Winners;
    }

    public void DisplayWinners()
    {
        List<Stopwatch> top3Winners = Top3();
        string message = "";
        foreach (var winner in top3Winners)
        {
            message += winner.ToString() + "\n";
        }
    }

    public IEnumerator<Stopwatch> GetEnumerator()
    {
        for (int i = 0; i < stopwatches.Count(); i++)
        {
            yield return stopwatches[i];
        }
    }

    IEnumerator IEnumerable.GetEnumerator()
    {

```

```

        return GetEnumerator();
    }
}

```

AddForm.cs:

```

using System;
using System.Globalization;
using System.Text.RegularExpressions;
using System.Windows.Forms;

namespace Coursework00P_Part_C_
{
    public partial class AddForm : Form
    {
        Result list;
        public AddForm(Result list)
        {
            InitializeComponent();
            this.list = list;
        }

        private void btnExit1_Click(object sender, EventArgs e)
        {
            this.Close();
        }

        private void btnContinue_Click(object sender, EventArgs e)
        {
            string name = tbxName.Text;
            double minTime;
            double maxTime;

            if (string.IsNullOrEmpty(name))
            {
                txbError1.Text = "All fields must be filled in!";
                return;
            }
            else if (name.Length < 3)
            {
                txbError1.Text = "Name must contain at least 3 Letters!";
                return;
            }
            else if (!Regex.IsMatch(name, @"^[a-zA-Z\s]+$"))
            {
                txbError1.Text = "Text fields must contain Only Letters!";
                return;
            }
            else if (!Regex.IsMatch(name, @"^[a-zA-Z\s]+$"))
            {
                txbError1.Text = "Text fields must contain Only Letters!";
                return;
            }
            else if (!double.TryParse(tbxMinTime.Text, NumberStyles.AllowDecimalPoint,
CultureInfo.InvariantCulture, out minTime))
            {
                txbError1.Text = "Invalid value for Minimum time!";
                return;
            }

            else if (!double.TryParse(tbxMaxTime.Text, NumberStyles.AllowDecimalPoint,
CultureInfo.InvariantCulture, out maxTime))
            {

```

```

        txbError1.Text = "Invalid value for Maximum time!";
        return;
    }
    else if (minTime >= maxTime)
    {
        txbError1.Text = "Min time must be less than Max time!";
        return;
    }
    else
    {
        tbxName.Clear();
        tbxMinTime.Clear();
        tbxMaxTime.Clear();
        txbError1.Clear();

        Stopwatch participant = new Stopwatch(name, minTime, maxTime);
        list.AddSprinter(participant);
    }
    MessageBox.Show("The Sprinters has been Successfully Added to the List :)",
"Success!", MessageBoxButtons.OK, MessageBoxIcon.Information);
}
}
}

```

DeleteForm.cs:

```

using System;
using System.Data;
using System.Linq;
using System.Windows.Forms;

namespace CourseworkOOP_Part_C_
{
    public partial class DeleteForm : Form
    {
        Result list;
        public DeleteForm(Result list)
        {
            InitializeComponent();
            this.list = list;

            var names = list.Participants.Select(x => x.Name).Distinct();
            foreach (var country in names)
            {
                cmbxNames.Items.Add(country);
            }
        }

        private void btnExit2_Click(object sender, EventArgs e)
        {
            this.Close();
        }

        private void btnDelete_Click(object sender, EventArgs e)
        {
            if (cmbxNames.SelectedItem == null)
            {
                MessageBox.Show("Please select a Spinter to Delete!", "Error!",
                MessageBoxButtons.OK, MessageBoxIcon.Error);
                return;
            }
        }
    }
}

```

```

        var toDelete = list.Participants.Where(p => p.Name ==
(string)cmbxNames.SelectedItem).ToList();
        foreach (var participant in toDelete)
        {
            list.Participants.Remove(participant);
        }
        cmbxNames.Items.Remove(cmbxNames.SelectedItem);

        MessageBox.Show("Selected Sprinters have been Successfully Deleted!",
"Success!", MessageBoxButtons.OK, MessageBoxIcon.Information);
    }
}

```

DisplayForm.cs:

```

using System;
using System.Windows.Forms;

namespace Coursework00P_Part_C_
{
    public partial class DisplayForm : Form
    {
        Result list;
        public DisplayForm(Result list)
        {
            InitializeComponent();
            this.list = list;

            listBox1.Items.Add("\n\nDemonstration of Late Binding Mechanism:");

            DigitalCounter digitalCounter = new DigitalCounter(16.2, 18.8);
            Stopwatch stopwatch = new Stopwatch("Tommy", 17.1, 20.4);

            listBox1.Items.Add("Digital Counter: " + digitalCounter.ToString());
            listBox1.Items.Add("Stopwatch: " + stopwatch.ToString());

            listBox1.Items.Add("\nList of Sprinters: ");

            if (list != null)
            {
                foreach (var participant in list.Participants)
                {
                    listBox1.Items.Add(participant.ToString());
                }
                list.DisplayParticipants();
            }
            else
            {
                MessageBox.Show("The list object is not initialized :(", "Error",
MessageButtons.OK, MessageBoxIcon.Error);
            }
        }

        private void btnExit2_Click(object sender, EventArgs e)
        {
            this.Close();
        }

        private void btnShow_Click(object sender, EventArgs e)
        {
            listBox2.Items.Clear();
            listBox2.Items.Add("Sprinters who won Prizes: ");
        }
    }
}

```

```

        double allAvgTime = list.AllAvgTime();
        foreach (var winners in list.Top3())
        {
            listBox2.Items.Add(winners.ToString());
        }
        list.DisplayWinners();
        listBox2.Items.Add("\n\nAverage Race Time of All Sprinters = " + allAvgTime + "
seconds");

        listBox2.Items.Add("\n\nMinimum race time = " + list.MinTime() + " seconds");
        listBox2.Items.Add("\n\nMaximum race time = " + list.MaxTime() + " seconds");
    }
}

```

MyMain.cs:

```

using System;
using System.Collections.Generic;
using System.IO;
using System.Linq;
using System.Windows.Forms;
using Newtonsoft.Json;

namespace CourseworkOOP_Part_C_
{
    public partial class MyMain : Form
    {
        Result list;

        private const string inputFile = "myFile.txt";
        public MyMain()
        {
            InitializeComponent();
            list = new Result(new List<Stopwatch>());
        }

        private void toolStripButton1_Click(object sender, EventArgs e)
        {
            AddForm addForm = new AddForm(list);
            addForm.ShowDialog();
        }

        private void toolStripButton3_Click(object sender, EventArgs e)
        {
            DeleteForm deleteForm = new DeleteForm(list);

            if (list.Participants.Count == 0)
            {
                MessageBox.Show("The List of Sprinters is Empty! Unable to Delete :(", "Empty
List!", MessageBoxButtons.OK, MessageBoxIcon.Information);
                return;
            }
            else
            {
                deleteForm.ShowDialog();
            }
        }

        private void toolStripButton2_Click(object sender, EventArgs e)
        {
            DisplayForm displayForm = new DisplayForm(list);

            if (list.Participants.Count == 0)
            {
                MessageBox.Show("The List of Sprinters is Empty! Unable to View :(", "Empty
List!", MessageBoxButtons.OK, MessageBoxIcon.Information);
            }
        }
    }
}

```



```

        return;
    }
    else
    {
        displayForm.ShowDialog();
    }
}

private void Exit_Click(object sender, EventArgs e)
{
    MessageBox.Show("Thanks for visiting, see you again :)", "Goodbye!",
    MessageBoxButtons.OK, MessageBoxIcon.Information);
    Application.Exit();
}

private void toolStripButton6_Click(object sender, EventArgs e)
{
    try
    {
        if (File.Exists(inputFile))
        {
            string json = File.ReadAllText(inputFile);
            var stopwatches = JsonConvert.DeserializeObject<List<Stopwatch>>(json);
            list = new Result(stopwatches);
            MessageBox.Show("The file has been Read Successfully :)", "Success!",
            MessageBoxButtons.OK, MessageBoxIcon.Information);
        }
        else
        {
            MessageBox.Show("File not found :( ", "Error!", MessageBoxButtons.OK,
            MessageBoxIcon.Error);
        }
    }
    catch (Exception ex)
    {
        MessageBox.Show($"An error occurred while deserializing: {ex.Message}",
        "Error!", MessageBoxButtons.OK, MessageBoxIcon.Error);
    }
}

private void toolStripButton5_Click(object sender, EventArgs e)
{
    try
    {
        if (list.Participants.Any())
        {
            string json = JsonConvert.SerializeObject(list.Participants);
            File.WriteAllText(inputFile, json);
            MessageBox.Show("The data has been successfully added to the file :)",
            "Success!", MessageBoxButtons.OK, MessageBoxIcon.Information);
        }
        else
        {
            MessageBox.Show("There are no sprinters to save!", "Empty List!",
            MessageBoxButtons.OK, MessageBoxIcon.Information);
        }
    }
    catch (Exception ex)
    {
        MessageBox.Show($"An error occurred while serializing: {ex.Message}", "Error!",
        MessageBoxButtons.OK, MessageBoxIcon.Error);
    }
}
}
}

```