КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ ТОРГОВЕЛЬНО-ЕКОНОМІЧНИЙ УНІВЕРСИТЕТ

Кафедра інженерії програмного забезпечення та кібербезпеки

Захищено на кафедрі інженерії програмного забезпечення та кібербезпеки
«14»_травня_____2021р.
з оцінкою_94 бали _____

Підпис членів комісії:
_____
_____
_____

КУРСОВА РОБОТА
з дисципліни
«ОБ'ЄКТНО-ОРІЄНТОВАНЕ ПРОГРАМУВАННЯ»
НА ТЕМУ:

«Розробка програмного додатку для налаштування струнного інструменту мовою програмування C#»

Виконала: студентка факультету інформаційних технологій
2 курсу 12 групи
Войткевич Аліса Андріївна

Науковий керівник:
асистент кафедри інженерії програмного забезпечення та кібербезпеки
Гнатченко Дмитро Дмитрович

Київ 2021

Київський національний торговельно-економічний університет

Кафедра інженерії програмного забезпечення та кібербезпеки

КАРТКА

завдання та контролю за ходом виконання курсової роботи

1. Прізвище та ініціали студента Войткевич А. А.
2. Факультет Факультет інформаційних технологій(ФІТ)
3. Курс, група 2 курс 12 група
4. Форма навчання денна
5. Номер залікової книжки №34/19-272
6. Номер та назва теми курсової роботи Розробка програмного додатку для налаштування струнного інструменту мовою програмування C#
7. Дата погодження теми з науковим керівником 21.02.2021
8. Дата затвердження плану роботи 21.02.2021
9. Дата подання виконаної роботи на кафедрі 07.05.2021
10. Дата захисту роботи 14.05.2021
11. Тему, план і строки прийняв до виконання _____

(підпис студента)

12. Науковий керівник Гнатченко

(прізвище та підпис)

“ 14 ” травня 2021р.

ЗМІСТ

ВСТУП

Анотація: Загальний обсяг роботи склав 31 сторінки, де знаходиться 15 рисунків, 1 таблиця, 2 блок-схеми, 8 додатків. Розглянуто такі основні поняття як: об’єктно–орієнтоване програмування, основні концепції об’єктно–орієнтованого програмування, мови програмування, мова програмування C#, Visual Studio, Windows Presentation Foundation. Сутність роботи полягає у розробці програмного додатку для налаштування струнного інструменту.

Об'єктно-орієнтоване програмування (ООП) є невід'ємною частиною розробки програмного забезпечення. Мови програмування, що використовують основні ідеї та принципи концепції ООП, можна використовувати для розробки програм для будь-якої платформи, у тому числі додатків для персональних комп’ютерів і мобільних пристроїв.

Об'єктно-орієнтовані мови програмування поставляються з багатими бібліотеками об'єктів, а код, розроблений в ході реалізації проекту, також може бути повторно використаний в майбутньому при створенні інших програм. Так, об'єктно-орієнтовані програми можна писати на таких популярних нині мовах, як Python, JavaScript, C++, PHP і інші. Нині C# вважається переважною мовою для розробки під Windows.

Актуальність теми: Інформатизація суспільства сьогодення впливає на усі сфери суспільного життя, у тому числі й музики. Програми для роботи зі звуком та налаштування музичних інструментів користуються попитом у цій сфері. На даний час існує багато способів налаштування інструменту, наприклад, за допомогою камертона, переносного, мобільного чи вбудованого приладу. Так як тюнер - невід'ємний аксесуар будь-якого музиканта, оскільки саме він відповідає за правильність звучання, можливість налаштування інструменту в будь-який час лише завдяки мікрофону і відповідному додатку на персональному комп’ютері дуже корисна. Швидкий доступ і зручність використання приваблюють користувача.

Завдання курсової полягає у визначенні основних засад об’єктно-орієнтованого програмування, середовища програмування та відповідного інструментарію, дослідженні наявних програмних забезпечень на ринку, описі і розробці програмного додатку для налаштування струнного інструменту засобами Visual C#, створенні зручного інтерфейсу для користувача.

Метою даної курсової роботи є створення зручного, точного, швидкого програмного додатку для налаштування струнного інструменту мовою програмування C#, як для початківців, так і для професіоналів.

Об’єктом дослідження є управління програмним додатком для налаштування струнного інструменту.

Предметом дослідження є програмний додаток для налаштування струнного інструменту мовою програмування C#.

Під час виконання проекту буде створено програмний додаток для налаштування струнного інструменту мовою програмування C#, інструменти програмування містять інтегроване середовище розробки Microsoft Visual Studio, компілятор Microsoft Visual C#.

РОЗДІЛ 1
ТЕОРЕТИЧНІ ПОЛОЖЕННЯ ОБ’ЄКТНО-ОРІЄНТОВАНОГО ПРОГРАМУВАННЯ

1.1. Об’єктно-орієнтоване програмування як одна з домінуючих моделей програмування

Об’єктно-орієнтованим можна назвати набір принципів програмування, що передбачає опис предметної області, як комплекс об’єктів, що взаємодіють. В даному випадку, об’єкт – це комбінація функцій, змінних і даних, що має внутрішні властивості і виконує набір пов’язаних дій (алгоритмів). Функціонування об’єкту визначається реалізацією цих дій.

Об'єктно-орієнтоване програмування, або ООП, є підходом до розв'язання задач, де всі обчислення здійснюються з використанням об'єктів. Об'єкт - це компонент програми, який вміє виконувати певні дії і знає як взаємодіяти з іншими елементами програми. [1]

Об'єктно-орієнтоване програмування — модель комп'ютерного програмування, де увага сконцентрована навколо даних або об’єктів, а не навколо функцій і логіки, необхідних для маніпулювання ними. Така тактика програмування доцільна для великих, складних програм, що активно оновлюються та підтримуються.

Переваги ООП включають в себе:

1) легкість читання – чіткі форми конструкції, формату операторів, оголошення даних, введення коментарів, застосування ключових слів, необмеженість довжини ідентифікаторів;
2) швидкість написання – можливість паралельного розвитку класів;
3) простоту реалізації великого функціоналу – весь код розділений на ряд елементів;
4) багаторазове використання коду – за допомогою успадкування клас і підклас можуть бути похідними, а дані та функції можуть бути використані;

5) безпечний розвиток – приховані дані, які не можуть бути оцінені жодною зовнішньою функцією;
6) масштабованість та ефективність – програми не є одноразовими, але застарілий код потрібно оновлювати.

Об'єктно-орієнтоване програмування – це зразок програмування, що базується на уявленні програми у вигляді комплексу об'єктів, кожен з яких є екземпляром певного класу, а класи утворюють ієрархію спадкоємства.

Об'єкт — це базова одиниця ООП, що представляє собою реальну сутність. Об’єкт має ідентичність, стан, властивості, поведінку і містить відомості і код для управління даними. Об’єкти можуть взаємодіяти без точних даних чи коду один одного, знаючи тип отриманого повідомлення та тип відповіді, що повертається об’єктами. Структура і поведінка схожих об'єктів визначає загальний для них клас. Перш ніж створювати об’єкт, потрібно мати клас. Терміни «екземпляр класу» і «об'єкт» взаємозамінні[2].

Клас – це схема або план об’єкта, яка містить змінні для збереження даних і функцій з метою виконання операцій над ними. Клас не займає простору пам’яті і є лише абстракцією істотних властивостей об’єкту. Це тип даних, що визначається користувачем і складається з даних і функцій користувачів, які можуть використовуватися при умові створення екземпляра класу. З точки зору програмування клас можна розглядати як набір даних (полів, атрибутів, членів класу) і функцій для роботи з ними (методів) [3].

Після створення об’єкту, його подальші дії керуються методами. Метод в об’єктно-орієнтованому програмуванні – це процедура, пов’язана з класом. Метод визначає поведінку об’єктів, створених з класу. Тобто це дія, котру об’єкт здатний виконати.

Функція – це комбінація інструкцій, що об’єднані для досягнення певного результату. Зазвичай функція потребує деяких вхідних даних (аргументів), щоб повернути певний результат. На відміну від метода, функція

незалежна і не пов’язана з класом. Функцію можна використовувати в будь-якому місці коду, без наявності об’єкта для використання.

1.2. Основні концепції об’єктно-орієнтованого програмування

Об’єктно-орієнтоване програмування базується на таких фундаментальних принципах:

1) Інкапсуляція
2) Абстракція
3) Успадкування
4) Поліформізм

Інкапсуляція – це об'єднання даних разом з методами, що працюють з цими даними, в єдине ціле. Багато мов програмування часто використовують інкапсуляцію у вигляді класів. Інкапсуляція також може посилатися на механізм обмеження прямого доступу до деяких компонентів об'єкта, такий, що користувачі не можуть отримати доступ до значень стану для всіх змінних конкретного об'єкта. Інкапсуляція може бути використана для приховування як членів даних, так і функцій даних або методів, пов'язаних з екземпляром класу або об'єкта.[4]

Абстракція даних відноситься до приховування будь-яких недоречних даних про об'єкт або екземпляр класу, фонових деталей чи реалізації і до надання лише важливої інформації про дані зовнішньому світу, щоб зменшити складність і допомогти користувачам ефективніше взаємодіяти з програмою. Термін абстракція, на відміну від інкапсуляції, може бути використаний для опису процесу приховування деякої інформації, що міститься в об'єкті або класі, але він також може посилатися на сам об'єкт. Абстракція - це будь-яка іменна сутність, що містить вибірку даних та поведінку, характерну для конкретного використання об’єкту, що виник.

Успадкування представляє собою механізм, що дозволяє певному класу отримувати властивості іншого. Тобто успадкування дозволяє створювати

новий клас, де повторно використовуються члени даних та методи існуючого базового класу. Похідний клас – це новий клас, а базовий клас – це існуючий клас. Успадкування дозволяє користувачу повторно використовувати код, коли це можливо, і зменшити його надмірність. Похідний клас може мати додаткові функції і при цьому повністю задовольняти базовий клас.

Поліморфізм – це здатність повідомлення відображатися більш, ніж в одній формі. Поліморфізм дає можливість об’єктам змінювати форму в залежності від контексту під час виконання програми. Це означає, що будь-який дочірній об'єкт класу може приймати будь-яку форму класу в батьківській ієрархії і, звичайно ж, сам по собі. Насправді це означає, що дочірній об'єкт класу може бути призначений будь-якому посиланню класу в його батьківській ієрархії і, звичайно ж, самому собі.

1.3. Об’єктно-орієнтоване програмування як новий підхід до створення програм

Більшість сучасних мов створені для полегшення об’єктно-орієнтованого програмування. Але техніками ООП можна користуватися і не в об’єктно-орієнтованій мові, а застосування об’єктно-орієнтованої мови не гарантує об’єктно-орієнтованість коду. Насправді, безліч комп’ютерних програм і велика частина контенту в Інтернеті побудована на високорівневих об’єктно-орієнтованих мовах програмування. Основні сфери застосування об’єктно-орієнтованих мов охоплюють системи реального часу, моделювання, дизайн інтерфейсу користувача, бази даних, системи підтримки рішень, автоматизацію управління та багато іншого.

Найпопулярнішими об’єктно-орієнтованими мовами програмування є Java, Ruby, Delphi, Python, C++, PHP, C#. Наприклад, Java є однією з найпопулярніших мов програмування, що використовується для розробки мобільних пристроїв, розробки програмного забезпечення та багато іншого. Більшість світових розробників обирають Python за його простий синтаксис та

здатність виконувати комплексні дії. Він використовується в штучному інтелекті, вивченні даних та машинному вивченні. Ця мова добре підходить для створення об’єктів. Ruby - чиста мова ООП, яка працює на об'єктах. Всі ці об'єкти є значеннями в Ruby. Можна використовувати існуючі коди для створення того, що ви хочете кодувати, замість того, щоб створювати абсолютно нову програму на Ruby з нуля.

Висновки до Розділу 1

Отже, підсумовуючи, можна сказати, що об’єктно-орієнтоване програмування – це спосіб програмування, що напрямлений на реалізацію дійсних об’єктів, таких як успадкування, поліморфізм, інкапсуляція чи абстракція. А основною метою ООП є сполучення даних і функцій, які з ними взаємодіють, щоб доступ до цих даних мала тільки певна функція.

Об’єктно-орієнтоване програмування стало невід’ємною частиною розробки програмних забезпечень. Такі переваги об’єктно-орієнтованого програмування, як збереження коду, можливість повторного використання, безпека і зручність, простота усунення несправностей, дають більшу продуктивність і якісний результат.

РОЗДІЛ 2
АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ВИБІР ІНСТРУМЕНТІВ РОЗРОБКИ

2.1. Аналіз проблеми і методи її вирішення

Будь-який інструмент вимагає точного налаштування, для того, щоб він міг звучати в відповідності зі своїм призначенням. Деякі види вимагають регулювання дуже часто (скрипка), інші - дуже рідко (рояль, барабани) або отримують особливості звучання в момент виготовлення (дерев'яні духові). Гітара займає проміжне положення: як щипковий струнний інструмент вона вимагає налаштування перед кожним сеансом гри.

Для того, щоб спростити процес налаштування і скоротити час, необхідний для цього, були розроблені різні пристосування. Хоча вмілий музикант повинен вміти налаштовувати гітару на слух, це не завжди зручно, а головне - не надто швидко.

Майже кожна людина має вільний доступ до комп’ютера, де містяться забезпечення на будь-який випадок у житті. Тому камертони та тюнери втрачають свою актуальність. На зміну їм приходять програмні додатки, як зручний, а іноді надійний спосіб налаштування інструменту.

Перевагами користування програмою перед тюнером-кліпсою або модулем є:

- безкоштовність, тоді як інші типи тюнерів мають ціну;
- зручність, так як пристрій завжди під рукою.

Серед наявних програмних забезпечень на ринку найпопулярнішими є:

1) Pitchlab, що має дуже широкий діапазон частот і це робить його чудовим додатком для налаштування гітари. Може аналізувати висоту, гармонію і навіть форми сигналу вашого звуку;
2) gStrings - може керувати чутливістю мікрофона, генерувати тони та поставляється з налаштуваннями для різних темпераментів;

3) Roadie Tuner – можливість запрограмувати власні налаштування для всіх струнних інструментів і додаткова опція у вигляді цікавих публікацій в блозі;
4) Martin Tuner - постачається з навчальними посібниками для тренування слуху. Він включає в себе підручник з перетягування струн гітари та загалом гарний додаток з безліччю додаткових функцій.

Проаналізувавши дані програмні додатки, було вирішено створити простий, швидкий і зручний програмний додаток для налаштування струнного інструменту як для аматорів, так і для професіоналів.

Принцип роботи програми полягає в наступному: комп'ютер поглинає живий звук чи музику за допомогою мікрофону, підключеного до звукової плати. Сучасні звукові плати можуть записувати цифрові сигнали. Цифровий сигнал - це набір дискретних звукових значень, які рівномірно розташовані в звуковому діапазоні. Цифровий сигнал не надає ніякої інформації про частоти, які присутні у звуці. Щоб визначити їх, дані повинні бути проаналізовані. Програма порівнює звук з частотами у пам'яті. І зіставляє відповідну частоту з нотою. На екрані з'являється нота і найближча до неї частота.

2.2. Вибір середовища та інструментів розробки

Процес створення програмного забезпечення важко уявити без середовища розробки. IDE (Integrated Development Environment) – інтегроване, єдине середовище розробки, що використовується розробниками для створення різноманітного програмного забезпечення. IDE представляє собою комплекс з декількох інструментів, а саме: текстового редактору, компілятору або інтерпретатору, автоматизації збірки та налагоджувачу. Крім того, IDE може містити інструменти для інтеграції з системами керування джерелами та іншими корисними утилітами. Є IDE, які призначені для роботи лише з однією мовою програмування, але більшість сучасних IDE дозволяють працювати одночасно з кількома.[5]

IDE дозволяє об’єднувати різноманітні аспекти написання комп’ютерної програми, такі як:

- редагування вихідного коду – спрощення процесу за допомогою автозаповнення та виділення синтаксису;
- вибір синтаксису – надання візуальних підказок, виділення різноманітних елементів, ключових слів;
- автозаповнення – передбачення подальшого вводу середовищем розробки;
- налагодження – надання середовищем розробки інструментів, що досліджують різноманітні змінні і перевіряють код;
- збірка файлів – компіляція вихідного коду файлу в клас, що виконується.

Враховуючи вище сказані аспекти, можна досягти мети IDE – підвищити продуктивність програміста.

При виборі середовища розробки потрібно ґрунтуватися на таких факторах:

- простота використання;
- мова розробки;
- на яких платформах працює.

Найпоширенішими IDE є Eclipse, Kite, NetBeans, VS Code, Code::Blocks, SlickEdit, Microsoft Visual Studio.

Eclipse – це платформа з відкритим вихідним кодом, що дозволяє користувачу створювати додатки для розробки. Підтримує такі мови програмування, як Python, C / C ++, Ruby, PERL, Java. Платформа, що підтримується – Windows. В основному використовується для розробки інтегрованих середовищ розробки, клієнтських додатків та інших інструментів. Надає готові шаблони коду, автоматично перевіряє синтаксис і

підтримує віддалене керування. Займає багато пам’яті, потребує перезавантаження після встановлення будь-якого додатка.

Kite – плагін для IDE, допоміжний помічник програмування, який працює на основі штучного інтелекту. Підтримує більше 16 мов. Надає підпис функції під час введення тексту, підказки при наведенні миші, забезпечує підтримку електронною поштою. Але потрібно надсилати код на свою домашню базу.

SlickEdit – мультиплатформний редактор коду. Підтримує більше 60 мов програмування. Платформи, що підтримуються: Windows, Linux, Mac, and Raspberry Pi. Пропонує підтримку контролю версій за допомогою стелажу, забезпечує інтеграцію JUnit, аналіз символів, розширення та відступ синтаксису. Працює повільно.

Visual Studio - це інтегроване середовище розробки, що дозволяє створювати .Net-додатки, веб-додатки та багато іншого. Підтримує 36 різних мов, доступний як для Windows, так і для MacOS.

Rider – кросплатформне IDE, призначене для роботи з .NET Framework та .NET Core. Допомагає працювати з SQL та базами даних, забезпечує 2200+ перевірок коду в реальному часі та сотні контекстних дій та рефакторингу. Rider має вбудовану підтримку TypeScript, JavaScript, HTML, CSS та Sass. Підтримує платформи Windows, Linux, Mac. Запускається повільно, ресурсомісткий .

Visual Studio Code – текстовий редактор, що забезпечує автозаповнення функціями IntelliSense. Підтримує C #, TypeScript, JavaScript та Node.js. і такі платформи, як MacOS, Windows, Linux. Легко розширюється, налаштовується, пропонує рефакторинг та налагодження коду. Потребує більш просунутих функцій налагодження, опцій для користувача.

Проаналізувавши наявні IDE та врахувавши переваги і недоліки, середовищем розробки було вибрано Visual Studio.

Visual Studio – інтегроване в Microsoft середовище розробки, яке можна використовувати для розробки консолей, графічних інтерфейсів користувача

(GUI), Windows Forms, веб-служб та веб-додатків. Visual Studio використовується для написання власного коду та керованого коду, що підтримується Microsoft Windows, Windows Mobile, Windows CE, .NET Framework, .NET Compact Framework та Microsoft Silverlight. Редактор коду Visual Studio .NET підтримує IntelliSense та рефакторинг коду, в той час як інтегрований налагоджувач Visual Studio .NET підтримує як налагодження вихідного, так і машинного рівня. Visual Studio .NET включає інші вбудовані інструменти, такі як конструктор форм, що корисно при створенні графічних програм; веб-дизайнер, який створює динамічні веб-сторінки; дизайнер класів, який використовується для створення власних бібліотек, і конструктор схем для підтримки баз даних.[6]

Вбудовані мови включають в себе C #, C ++, JavaScript, HTML, VB, .NET та F#. За допомогою плагінів можлива підтримка інших мов, зокрема M, Python та Ruby.

C # - одна з найстаріших, найнадійніших і найпоширеніших мов комп'ютерного програмування. Компанія Microsoft розробила простий, потужний, об’єктно-орієнтований та статично-типізований інструмент розробки програмного забезпечення.

C# використовується при створенні комп’ютерних, мобільних, хмарних та веб-додатків. На C# можна розробити програми математичного моделювання, комп’ютерні ігри, платіжні шлюзи та багато іншого. Відносно новими є реалізації наукових програм на C#. Ця мова програмування підійде для реалізації проектів різного рівня складності.

Мова базується на C++ та Java, але володіє додатковими розширеннями, бібліотеками та концепціями для реалізації різних програм та компонентно-орієнтованих концепцій програмування.

C# - це фактично гібрид різних мов. При цьому C# синтаксично не менш (якщо не більш) чистий, ніж Java, і такий же простий, як Visual Basic, але володіє практично тією ж потужністю і гнучкістю, що і C++ [7].

Подібно до Java, C# реалізував такі ідеї:

1) віртуальна машина – платформа .Net виконує програму подібно до віртуальної машини Java;
2) байт-код – програмний код компілюється в проміжну мову MSIL(Microsoft Intermediate Language), і тільки потім перетворюється в машинну мову, залежно від платформи на якій запускається програма;
3) керований код – оскільки програми, написані на C#, запускаються виключно у віртуальному середовищі CLR(Common Language Runtime), то можна контролювати їх виконання і зупиняти їх у будь-який момент, а також контролювати використання пам'яті, при необхідності - виділяти або видаляти частини пам'яті, які використовує програма[8].

Особливостями C# є:

1) Абсолютна підтримка класів та об’єктно-орієнтованого програмування, а також наслідування інтерфейсів і реалізацій, віртуальних функцій та перезавантаження операторів;
2) Повний, чіткий набір основних типів;
3) Вбудована підтримка автоматичної генерації XML-документації;
4) Автоматичне звільнення динамічно розподіленої пам'яті;
5) Можливість відмітки класів і методів атрибутами, що визначає користувач;
6) Повний доступ до бібліотеки базових класів .NET, а також додатковий доступ до Windows API;
7) Прямий доступ до пам’яті;
8) Підтримка властивостей і подій в стилі VB;
9) Можливість використання C# для написання динамічних web-сторінок ASP.NET.[7]

Таблиця 1. **Переваги C# перед іншими мовами програмування (C, C++, Java)**

Переваги перед C та C++	- Складено на проміжній мові (CIL) незалежно від мови, на якій він був розроблений, або цільової архітектури і операційної системи - Покажчики більше не потрібні - Можливості віддзеркалення - Не потрібно турбуватися про заголовок файлів ".h" - Визначення класів і функцій може бути виконане у будь-якому порядку - Декларація функцій і класів не потрібна - Класи можуть бути визначені в класах - Немає глобальних функцій або змінних, все належить класу
Переваги перед C++ та Java	- Формалізована концепція методів get-set, тому код стає більш розбірливим - Чистіше управління подіями (з використанням делегатів)
Переваги перед Java	- Набагато ефективніший швидший - Має більш примітивні типи значень - Індексатори дозволяють отримати доступ до об'єктів, як до масивів - Умовна компіляція - Перевантаження оператора - Використання покажчиків (обмежене)

Джерело: побудовано на основі джерела [9]

Висновки до Розділу 2

Проаналізувавши предметну область, наявні середовища розробки та інструменти, можна зробити висновок, що для створення програми вдало підходить середовище програмування IDE Microsoft Visual Studio. Такі задачі, як ефективність, результативність та продуктивність будуть досягнуті функціями MS Visual Studio під час розробки програмного забезпечення. На платформі Microsoft .NET, яка розвивається і користується попитом, набирає популярність мова програмування C#. Зручний і послідовний синтаксис, простота і універсальність стали основним мотивом для вибору мови програмування C#.

У процесі курсової роботи буде розроблено швидкий, зручний, точний програмний додаток для налаштування струнного інструменту для всіх музикантів: і аматорів, і професіоналів.

РОЗДІЛ 3
РОБРОБКА ПРОГРАМНОГО ДОДАТКУ ДЛЯ НАЛАШТУВАННЯ СТРУННОГО ІНСТРУМЕНТУ «TUNER»

3.1. Налаштування середовища розробки

Для роботи необхідно встановити на комп’ютер Microsoft Visual Studio. Виконати інсталяцію MS Visual Studio можна з офіційного сайту Microsoft, розділ «Developer&IT».

Під час інсталяції:

1. Оберіть «Інструменти» > «Параметри імпорту та експорту» на панелі меню, щоб відкрити «Майстер налаштування імпорту та експорту».
2. У майстрі налаштування імпорту та експорту оберіть «Скинути всі налаштування», а потім натисніть «Далі».

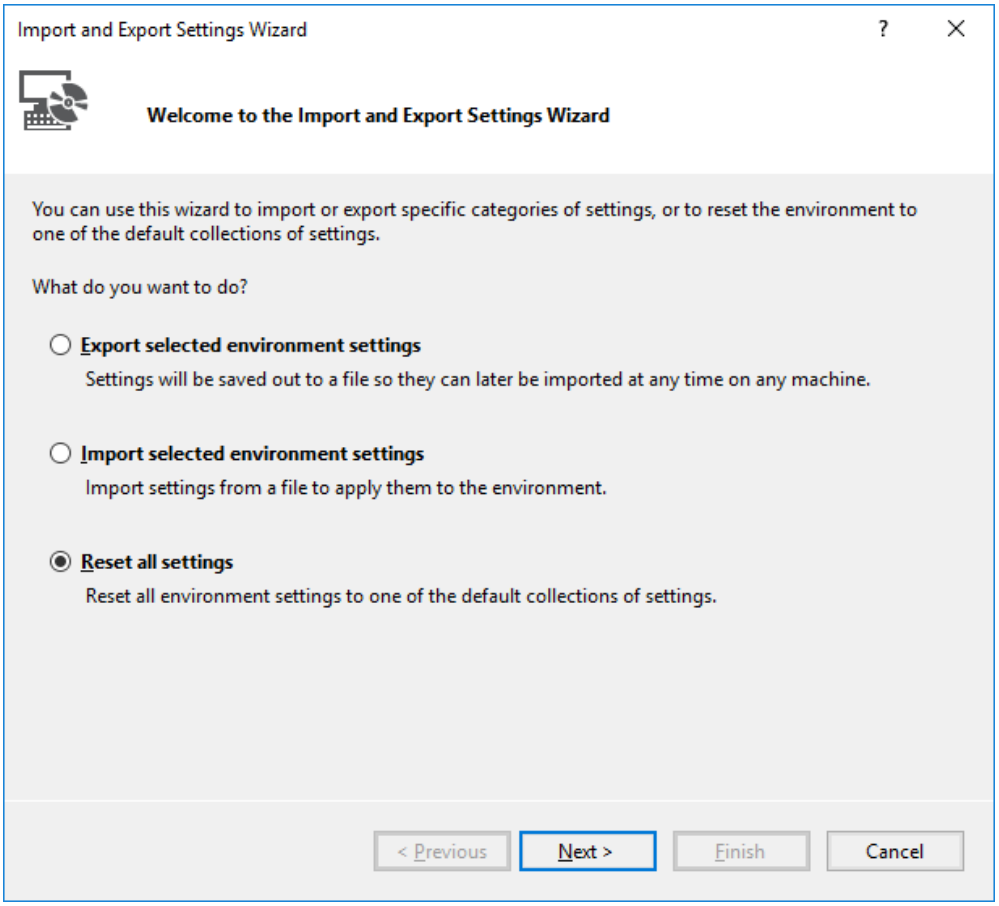

Рисунок 3.1 **Import and Export Settings Wizard**

Джерело: побудовано автором (знімок з екрану)

3. На сторінці «Зберегти поточні налаштування» оберіть «Так» чи «Ні», а потім «Далі».
4. На сторінці «Вибір колекції налаштувань» за замовчуванням оберіть колекцію, а потім натисніть «Готово».

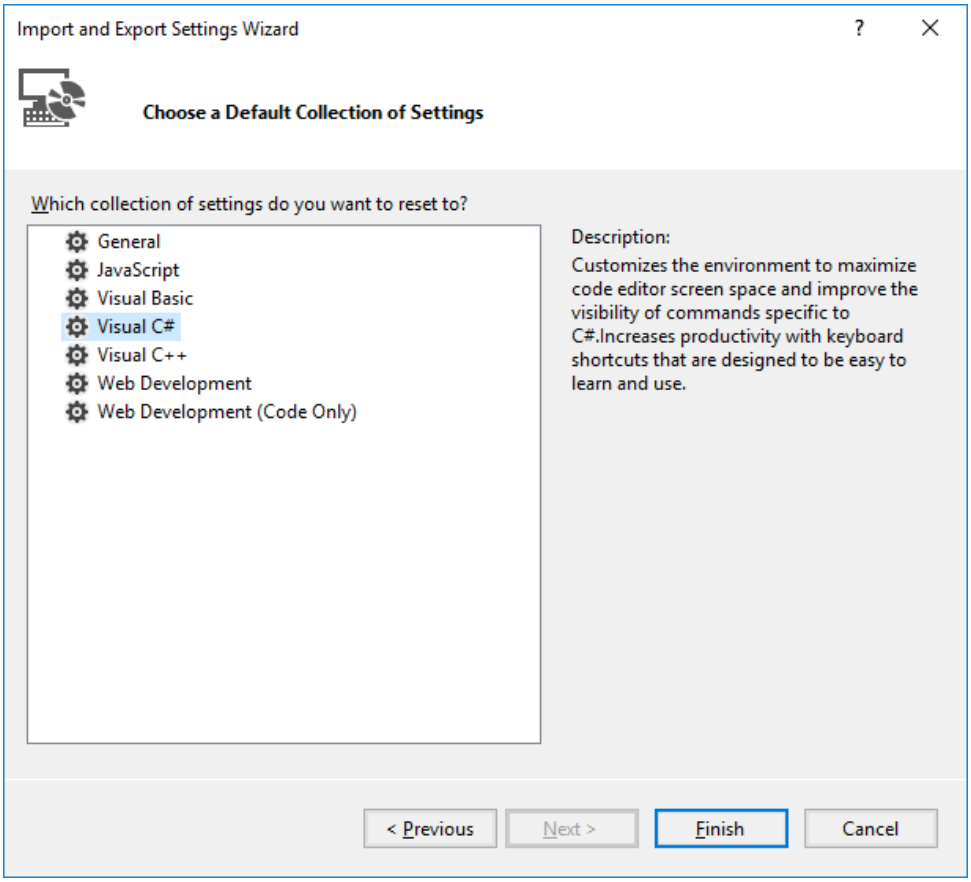

Рисунок 3.2 **Import and Export Settings Wizard**

Джерело: побудовано автором (знімок з екрану)

5. На сторінці “Скидання завершено” оберіть “Закрити”.

Якщо в меню середовища розробки вибрати пункт File > New > Project, на екрані з'явиться діалогова панель New Project.

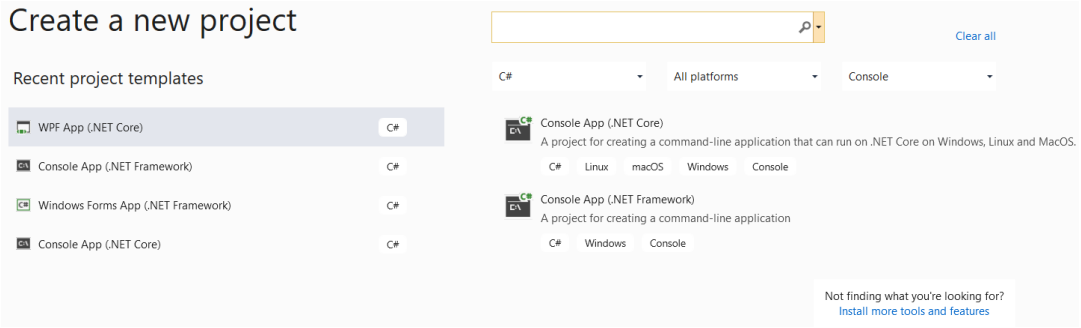

Рисунок 3.3 **Create a new project**

Джерело: побудовано автором (знімок з екрану)

При створенні нового проекту в поле Location необхідно вказати ім'я каталогу, в якому слід зберегти його файли. При цьому в даному каталозі автоматично буде створено інший каталог, ім'я якого збігається з ім'ям проекту.

В панель інструментів або меню Microsoft Visual Studio входять такі елементи, як Файл (File), Редагування (Edit), Проект (Project) та інші.

Рисунок 3.4 **Панель інструментів**

Джерело: побудовано автором (знімок з екрану)

3.2. Опис програми

Короткий принцип роботи тюнера: ми протягом реального часу слухаємо потік з мікрофона, який ми вказали за допомогою спеціальної кнопки Select Micro. Потім ми перетворимо масив бітів, який представляє звук, в дробове значення, яке є частотою звуку, за допомогою Pitcher-а, а потім вже по частоті звуку ми знаходимо найближче потрапляння до частоти звуку нот.

Тюнер заснований на бібліотеці NAudio для C #. Це бібліотека для роботи зі звуком на платформі .NET, що являється «обгорткою» для системних API, що спрощують розробку. Обгортки взаємодії для API-інтерфейсів NAudio Windows WaveIn захоплюють звук.

Дані надходять з джерела IWaveProvider. NAudio пропонує клас утиліти для перетворення звуку з короткого в плаваючий, який називається Wave16ToFloatProvider.

В класі Pitch встановлено формат запису. У даному проекті записуємо моно (тобто один канал), 16-бітний, 44,1 кГц аудіо – налаштування за замовчуванням для більшості мікрофонів.

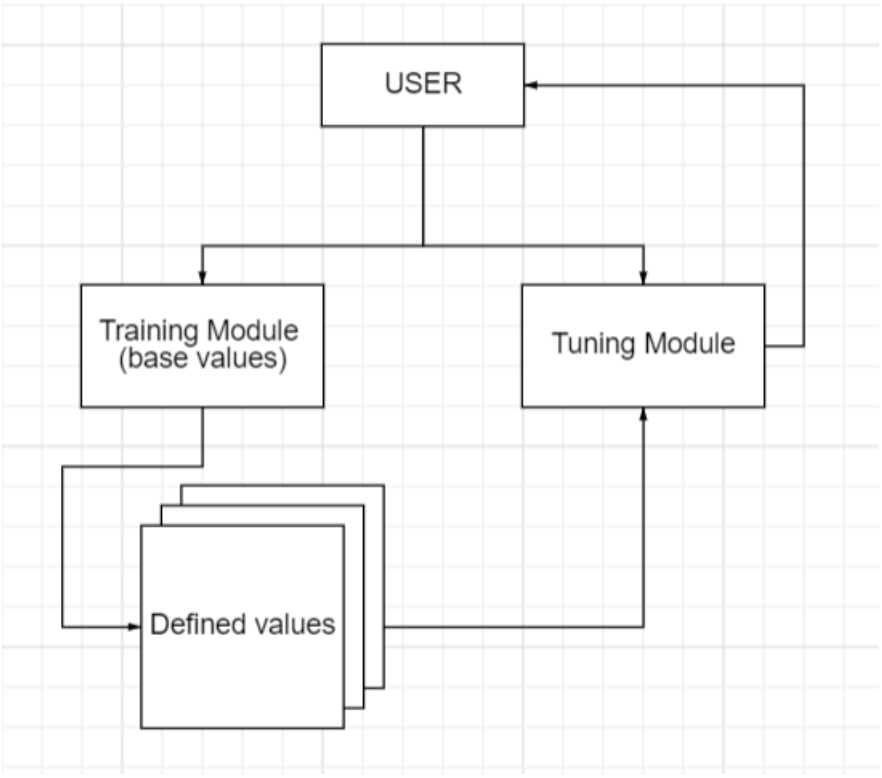

Рисунок 3.5 **Загальна архітектура системи**

Джерело: побудовано автором в системі Programforyou (знімок з екрану)

3.3. Опис основних класів

В об’єктно-орієнтованому програмуванні клас є ключовим елементом, у межах якого реалізується розробка програм. У даному програмному додатку реалізовані класи Pitch, Sound, Correlator, Note, а також спеціальний клас enum, що представляє групу констант, - FrequencyAccuracy.

Pitch

За допомогою методу Get ми отримуємо значення частоти звуку з буфера (масиву з байтами).

```
public float Get(byte[] buffer)
{
    if (waveBuffer == null || waveBuffer.MaxSize < buffer.Length)
    {
        waveBuffer = new WaveBuffer(buffer.Length);
    }

    int bytesRead = source.Read(waveBuffer, 0, buffer.Length);

    if (bytesRead > 0)
    {
        bytesRead = buffer.Length;
    }

    int frames = bytesRead / sizeof(float);

    return pitchDetector.DetectPitch(waveBuffer.FloatBuffer, frames);
}
```

Рисунок 3.6 **Метод Get**

Джерело: побудовано автором (знімок з екрану)

Correlator

Цей клас потрібен для накладання буфера з числовими значеннями звуку на кількість фреймів. А повертає він частоту звуку (цільного, складеного з масиву значень).

```
public float DetectPitch(float[] buffer, int frames)
{
    Tuple<float, int> corrTuple = GetPitchCorr(buffer, frames);
    int maxLag = corrTuple.Item2;
    float maxCorr = corrTuple.Item1;
    SetPrevBuffer(buffer, frames);
    float noiseThreshold = frames / PER_SECOND;
    if (maxCorr < noiseThreshold || maxLag == 0)
        return 0.0f;
    return this.sampleRate / maxLag;
}
```

Рисунок 3.7 **DetectPitch**

Джерело: побудовано автором (знімок з екрану)

Sound

Цей же клас реалізує логіку співвідношення частоти і гітарних нот. Спочатку ми ініціалізуємо список з нотами та їх значеннями:

```
private static Dictionary<string, float> noteBaseFreqs = new Dictionary<string, float>()
    {
        { "C", 16.35f },
        { "C#", 17.32f },
        { "D", 18.35f },
        { "Eb", 19.45f },
        { "E", 20.60f },
        { "F", 21.83f },
        { "F#", 23.12f },
        { "G", 24.50f },
        { "G#", 25.96f },
        { "A", 27.50f },
        { "Bb", 29.14f },
        { "B", 30.87f },
    };
```

Рисунок 3.8 **Ініціалізація значень нот**

Джерело: побудовано автором (знімок з екрану)

Потім ми створюємо подію, яка прослуховує дані з нашого мікрофона і починає весь процес перетворення звуку в масив байтів.

```
public void Detect(int deviceIndex)
{

    WaveInEvent waveIn = InitWaveIn(deviceIndex);
    bufferedWaveProvider = new BufferedWaveProvider(waveIn.WaveFormat);
    waveIn.StartRecording();

    IWaveProvider stream = new Wave16ToFloatProvider(bufferedWaveProvider);
    Pitch pitch = new Pitch(stream);

    ThreadStart newThread = new ThreadStart(delegate {
        RecordAudio(waveIn, pitch, stream);
    });

    Thread myThread = new Thread(newThread) { IsBackground = true };
    myThread.Start();
}
```

Рисунок 3.9 **Подія WaveInEvent**

Джерело: побудовано автором (знімок з екрану)

Після того, як ми отримали частоту звуку - визначаємо її строкове написання:

```
public static Note GetNote(float freq)
{
    float baseFreq;
    foreach (var note in noteBaseFreqs)
    {

        baseFreq = note.Value;
        for (int i = 1; i <= MAX_NOTE_NUMBER; i++)
        {
            double pow = Math.Pow(2, i - 1);
            if (pow < 2)
                pow = 2;
            float nextFreq = Sound.GetNextFrequency(new Note(note.Key + $"{i - 1}", baseFreq, baseFreq));
            double freqDiff = Math.Abs(baseFreq - freq);
            double diff = nextFreq - baseFreq;
            if (((freqDiff <= diff/2) && (nextFreq > freq) && freq < baseFreq + diff / 2)
                || (freq == baseFreq))
            {
                if(i == 1)
                    return new Note(note.Key, freq, baseFreq);
                return new Note(note.Key + $"{i - 1}", freq, baseFreq);
            }
            baseFreq *= 2;
        }
    }
    return null;
}
```

Рисунок 3.10 **GetNote**

Джерело: побудовано автором (знімок з екрану)

Note

Цей клас відображає абстракцію зіграної ноти. Ось які властивості він має:

Value - строковий запис ноти.

Frequency – частота ноти.

BaseFrequency – частота ідеально зіграної ноти.

Accuracy – точність попадання в ідеально зіграну ноту (далеко - far, близько – near, точно - exact).

Точність потрапляння обчислюється наступним чином:

```
private void CalculateAccuracy()
{
    float nextFreq = Sound.GetNextFrequency(this);
    double freqDiff = Math.Abs(BaseFrequency - Frequency);
    double diff = nextFreq - BaseFrequency;
    if (freqDiff > diff / 4)
        Accuracy = FrequencyAccuracy.Far;
    else if (freqDiff <= diff / 4 && freqDiff > diff / 10)
        Accuracy = FrequencyAccuracy.Near;
    else
        Accuracy = FrequencyAccuracy.Exact;
}
```

Рисунок 3.11 **CalculateAccuracy**

Джерело: побудовано автором (знімок з екрану)

Спочатку ми знаходимо наступну по частоті ноту, потім знаходимо різницю між ними і ділимо на 2 (це буде максимально далеке значення даної ноти), а потім ми просто обчислюємо в якій частині знаходиться нота: в половині, в чверті або настільки близько, що результат, поділений на 10 буде більше, ніж поточна похибка в частоті.

Алгоритм обчислення наступної ноти представлений в класі Sound:

```
public static float GetNextFrequency(Note _note)
{
    bool isReturnNext = false;
    if(_note.Value.Remove(_note.Value.Length - 1, 1) == "B")
            return (float)(noteBaseFreqs["C"] * Math.Pow(2, _note.SPN + 1));
    foreach (var note in noteBaseFreqs)
    {
        if (isReturnNext)
            return (float)(note.Value * Math.Pow(2, _note.SPN));
        if (note.Key == _note.Value.Remove(_note.Value.Length - 1, 1))
            isReturnNext = true;
    }
    return 0;
}
```

Рисунок 3.12 **GetNextFrequency**

Джерело: побудовано автором (знімок з екрану)

Тут обчислення відбуваються за допомогою SPN (Scientific Pitch Notation). Наукова нотація звуку – це метод визначення музичного тону шляхом об’єднання назви музичної ноти та числа, що ідентифікує висоту октави. Тобто це цифра поруч з нотою, наприклад, C4, G#3. Якщо нота є останньою серед показників SPN (тобто нота B), це означає, що наступною за рахунком буде нота C.

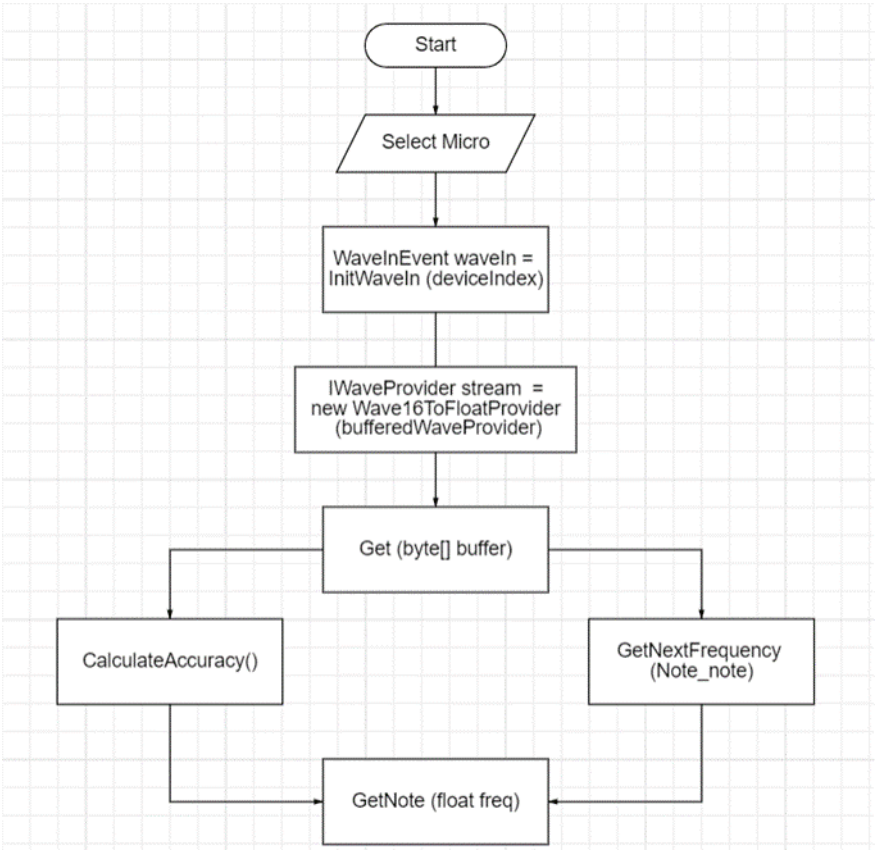

Рисунок 3.13 **Блок-схема алгоритму роботи тюнера**

Джерело: побудовано автором в системі Programforyou (знімок з екрану)

3.4. Керівництво користувача

Програма «Tuner» володіє доступним, інтуїтивно зрозумілим інтерфейсом, що дає можливість користуватися додатком навіть недосвіченим користувачам. Платформа розробки Windows Presentation Foundation підтримує широкий набір компонентів для розробки додатків, включаючи

модель додатку, ресурси, елементи управління, графіку, макет, документи і безпеку[10].

Користувач запускає тюнер і обирає вихідний пристрій (мікрофон), натиснувши кнопку «Select Micro».

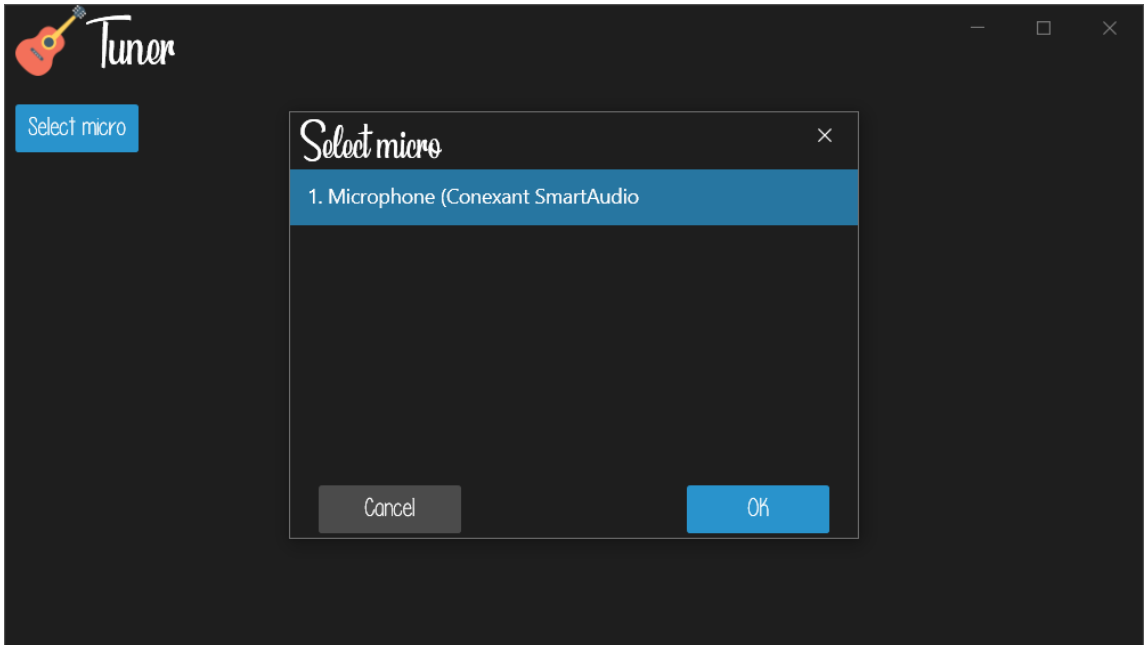

Рисунок 3.14 **Інтерфейс програмного додатку (MicroWindow)**

Джерело: побудовано автором (знімок з екрану)

Починайте грати з першої відкритої струни. Звіряйте точність налаштування: тюнер буде показувати ноту, на якій звучить кожна струна. Ваша мета - зробити так, щоб кожна струна видавала еталонний звук відповідно до програмного. Якщо на екрані з’явився напис «Higher» жовтого кольору, або «Lower» червоного, то продовжуйте налаштовувати інструмент далі: підтягніть або ослабте струну. Якщо ви бачите «Done» зеленого кольору, то ваш інструмент налаштовано. Намагайтеся грати ближче до мікрофона на вашому пристрої, так, щоб тюнер добре «чув» звук вашої гітари.

Рисунок 3.15 **Інтерфейс програмного додатку (MainWindow)**

Джерело: побудовано автором (знімок з екрану)

Висновки до Розділу 3

Програмний додаток для налаштування струнного інструменту “Tuner” розроблено у середовищі Microsoft Visual Studio мовою програмування C#. Інтерфейс користувача розроблено у Windows Presentations Foudation.

Основні класи, що реалізовані у програмному додатку: Pitch, Sound, Correlator, Note, FrequencyAccuracy.

Алгоритм роботи тюнера полягає в прослуховуванні потоку з мікрофону, який вказано за допомогою кнопки «Select Micro». Перетворенні масиву бітів, який представляє звук, в дробове значення, яке є частотою звуку за допомогою Pitcher-а. Знаходженні по частоті звуку найближчого потрапляння до частоти звуку нот.

ВИСНОВКИ

У результаті виконання роботи було визначено основні засади об’єктно-орієнтованого програмування, обрано середовище програмування та відповідний інструментарій, досліджено ринок на наявність програмних забезпечень для налаштування струнного інструменту, описано і розроблено програмний додаток для налаштування струнного інструменту засобами Visual C#, а також створено зручний інтерфейс для користувача.

Отже, об’єктно-орієнтоване програмування – це модель програмування, яка дозволяє працювати з об’єктами. А об’єкти, в свою чергу, мають поля для зберігання даних, і методи, що можуть виконувати визначені задачі. Клас – це шаблон об’єкту. Класи використовуються для створення та управління новими об'єктами та підтримки успадкування - ключового компонента об'єктно-орієнтованого програмування та механізму повторного використання коду. Додаткові переваги ООП включають в себе повторне використання коду, масштабованість та ефективність. Об’єктно-орієнтоване програмування базується на наступних принципах:

- Інкапсуляція – реалізація і стан об’єкту знаходяться всередині визначеного класу чи границі.
- Абстракція – приховування деталей реалізації від користувача, надана лише функціональність.
- Успадкування – виведення класу з іншого класу для ієрархії, що мають комплекс атрибутів і методів.
- Поліморфізм – об’єкти можуть приймати декілька форм в залежності від контексту.

Середовищем розробки було обрано Microsoft Visual Studio. Це швидке і просте забезпечення для різних платформ (Windows, Mac OS, Linux), що дозволяє розробляти веб-додатки, додатки для хмарних сервісів, мобільних та десктопних платформ. Visual Studio підтримує розробку мовою

програмування C#. Це одна з найпопулярніших мов програмування, якій легко навчитися та просто використовувати. Надає чітку структуру програмам і дозволяє повторно використовувати код.

Проаналізувавши предметну область і наявні пропозиції на ринку, було вирішено розробити програмний додаток для налаштування струнного інструменту мовою C#. Метою даної курсової роботи визначено створення зручного, точного, швидкого програмного додатку, як для початківців, так і для професіоналів.

Врахувавши мету, програмний продукт «Tuner» надав можливість точно та швидко налаштувати струнний інструмент і став чудовим аналогом камертону чи переносного, мобільного приладу. Він гарантує абсолютну чистоту звучання в порівнянні з регулюванням на слух, і підходить тим, хто не впевнений у якості свого слуху на 100%.

Для реалізації інтерфейсу користувача використовується Windows Presentation Foundation. WPF використовує мову XAML. Інтерфейс користувача є простим та доступним навіть для людини, яка ніколи не користувалася тюнером і не знає теорії музики.

«Tuner» включає в себе прийом та аналіз звукового сигналу, визначення частоти і відповідне зіставлення з нотою найближчої частоти.

Отже, мету і завдання курсової роботи було виконано.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Object-Oriented Programming vs. Procedural Programming [Електронний ресурс]: Study.com. -URL: https://study.com/academy/lesson/object-oriented-programming-vs-procedural-programming.html (дата звернення: 19.04.2021)
2. Объектно-ориентированное программирование / Г.И. Радченко, Е.А. Захаров. – Челябинск: Издательский центр ЮУрГУ, 2013. – 167 с.
3. Об'єктно-орієнтоване програмування [Електронний ресурс]: Освітній сайт КНУБА. Кафедра інформаційних технологій. Об'єктно-орієнтоване програмування. Лекція-1. -URL: org2.knuba.edu.ua/pluginfile.php/56765/mod_resource/content/0/Лекція-1.pdf
4. Encapsulation [Електронний ресурс]: Sumo Logic. DevOps and Security Glossary Terms. Encapsulation. -URL: https://www.sumologic.com/glossary/encapsulation (дата звернення: 14.04.2021)
5. 10 лучших IDE [Електронний ресурс]: Time Web. Комьюнити. 10 лучших IDE. -11.12.2019. -URL: https://timeweb. com/ru/community/articles/5-luchshih-ide-1 (дата звернення: 14.04.2021)
6. What is Visual Studio .NET? [Електронний ресурс]: Techopedia. Visual Studio .NET. -URL: https://www.techopedia.com/definition/15740/visual-studio-net (дата звернення: 14.04.2021)
7. История создания языка программирования C#/ Барабанов Н.В. [Електронний ресурс]: Web.Informatics.ru. История создания языка программирования C#. -URL: https://web.informatics.ru/works/17-18/web_online/barabanov_n_v/language_c.html (дата звернення: 14.04.2021)
8. Введение в C# [Електронний ресурс]: Program.top. Руководство. Введение в C#. -URL: https://programm.top/c-sharp/tutorial/introduction/ (дата звернення: 14.04.2021)
9. C# and it's advantages over other programming languages/ Jijith Jayakumar. -19.09.2016. [Електронний ресурс]: Wordpress.com. C# and it's advantages over other programming languages. -URL: https://jvjijith.wordpress. com/2016/09/19/c-and-its-advantages-over-other-programming-languages/ (дата звернення: 14.04.2021)
10. Початок роботи з WPF [Електронний ресурс]: Офіційний сайт Microsoft.com. Документація. -URL: https://docs.microsoft. com/ru-ru/visualstudio/designers/getting-started-with-wpf?view=vs-2019 (дата звернення: 14.04.2021)

ДОДАТКИ

Лістинг програми

Додаток А

Клас Correlator

```
class Correlator
    {
#region Constants

        private const int MAX_FREQ = 335;
        private const int MIN_FREQ = 75;
        private const float PER_SECOND = 1000f;
        private const double CORRELATION_ACCURACY = 0.9;

        #endregion

        #region Properties

        private float[] prevBuffer { get; set; }
        private int maxFreq { get; set; }
        private int minFreq { get; set; }
        private float sampleRate { get; set; }

        #endregion

        #region Ctor


        public Correlator(int sampleRate)
        {
            this.sampleRate = (float)sampleRate;

            maxFreq = sampleRate / MIN_FREQ;
            minFreq = sampleRate / MAX_FREQ;
        }

        #endregion

        #region Methods

        public float DetectPitch(float[] buffer, int frames)
        {
            Tuple<float, int> corrTuple = GetPitchCorr(buffer, frames);
            int maxLag = corrTuple.Item2;
            float maxCorr = corrTuple.Item1;
            SetPrevBuffer(buffer, frames);
            float noiseThreshold = frames / PER_SECOND;
            if (maxCorr < noiseThreshold || maxLag == 0)
                return 0.0f;
            return this.sampleRate / maxLag;
        }

        private void SetPrevBuffer(float[] buffer, int frames)
        {
            for (int n = 0; n < frames; n++)
                prevBuffer[n] = buffer[n];

        }
```

```
        private Tuple<float, int> GetPitchCorr(float[] buffer, int frames)
        {
            if (prevBuffer == null)
                prevBuffer = new float[frames];
            float secCor = 0;
            int secLag = 0;
            float maxCorr = 0;
            int maxLag = 0;
            for (int lag = maxFreq; lag >= minFreq; lag--)
            {
                float corr = 0;
                for (int i = 0; i < frames; i++)
                {
                    int oldIndex = i - lag;
                    float sample = ((oldIndex < 0) ? prevBuffer[frames + oldIndex] :
buffer[oldIndex]);
                    corr += (sample * buffer[i]);
                }
                if (corr > maxCorr)
                {
                    maxCorr = corr;
                    maxLag = lag;
                }
                if (corr >= CORRELATION_ACCURACY * maxCorr)
                {
                    secCor = corr;
                    secLag = lag;
                }
            }
            return new Tuple<float, int>(maxCorr, maxLag);
        }

        #endregion
}
```

Додаток Б

Клас FrequencyAccuracy

```
public enum FrequencyAccuracy
    {
        Far, Near, Exact
    }
```

Клас Note

```
public class Note
    {
        public Note(string value, float frequency, float baseFreq)
        {
            this.Value = value;
            this.Frequency = frequency;
            this.BaseFrequency = baseFreq;
            CalculateAccuracy();
        }

        public string Value { get; set; }
        public int SPN {
            get
            {
                if(int.TryParse(Value.LastOrDefault().ToString(), out int a))
                    return int.Parse(Value.LastOrDefault().ToString());
                return 0;
```

```
            }
        }
        public float Frequency { get; set; }
        public float BaseFrequency { get; set; }
        public FrequencyAccuracy Accuracy { get; private set; } =
FrequencyAccuracy.Exact;

        private void CalculateAccuracy()
        {
            float nextFreq = Sound.GetNextFrequency(this);
            double freqDiff = Math.Abs(BaseFrequency - Frequency);
            double diff = nextFreq - BaseFrequency;
            if (freqDiff > diff / 4)
                Accuracy = FrequencyAccuracy.Far;
            else if (freqDiff <= diff / 4 && freqDiff > diff / 10)
                Accuracy = FrequencyAccuracy.Near;
            else
                Accuracy = FrequencyAccuracy.Exact;
        }
    }
```

Додаток В

Клас Pitch

```
public class Pitch
    {

        IWaveProvider source;
        WaveBuffer waveBuffer;
        Correlator pitchDetector;

        public Pitch(IWaveProvider source)
        {
            if (source.WaveFormat.SampleRate != Sound.SAMPLE_RATE)
            {
                throw new ArgumentException("Source must be at 44.1kHz");
            }

            if (source.WaveFormat.Encoding != WaveFormatEncoding.IeeeFloat)
            {
                throw new ArgumentException("Source must be IEEE floating point audio
data");
            }

            if (source.WaveFormat.Channels != 1)
            {
                throw new ArgumentException("Source must be a mono input source");
            }

            this.source = source;
            this.pitchDetector = new Correlator(source.WaveFormat.SampleRate);
            this.waveBuffer = new WaveBuffer(Sound.BUFFER_SIZE);
        }

        public float Get(byte[] buffer)
        {
            if (waveBuffer == null || waveBuffer.MaxSize < buffer.Length)
            {
                waveBuffer = new WaveBuffer(buffer.Length);
            }

            int bytesRead = source.Read(waveBuffer, 0, buffer.Length);
```

```
            if (bytesRead > 0)
            {
                bytesRead = buffer.Length;
            }

            int frames = bytesRead / sizeof(float);

            return pitchDetector.DetectPitch(waveBuffer.FloatBuffer, frames);
        }
    }
```

Додаток Г

Клас Sound

```
public class Sound
    {
        #region Constants

        private const int MAX_NOTE_NUMBER = 9;

        private const float PLUS_FREQ_DIFF = 0.5f;
        private const float MINUS_FREQ_DIFF = 0.485f;

        public const int SAMPLE_RATE = 44100;
        public const int BUFFER_SIZE = 8192;

        #endregion

        private BufferedWaveProvider bufferedWaveProvider = null;



        private static Dictionary<string, float> noteBaseFreqs = new Dictionary<string,
float>()
            {
                { "C", 16.35f },
                { "C#", 17.32f },
                { "D", 18.35f },
                { "Eb", 19.45f },
                { "E", 20.60f },
                { "F", 21.83f },
                { "F#", 23.12f },
                { "G", 24.50f },
                { "G#", 25.96f },
                { "A", 27.50f },
                { "Bb", 29.14f },
                { "B", 30.87f },
            };

        public static string[] GetDevices()
        {
            string[] devices = new string[WaveIn.DeviceCount];
            for (int i = 0; i < devices.Length; i++)
            {
                devices[i] = $"{i+1}. {WaveInEvent.GetCapabilities(i).ProductName}";
            }
            return devices;
        }


        public void Detect(int deviceIndex)
        {
```

```
    WaveInEvent waveIn = InitWaveIn(deviceIndex);
    bufferedWaveProvider = new BufferedWaveProvider(waveIn.WaveFormat);
    waveIn.StartRecording();

    IWaveProvider stream = new Wave16ToFloatProvider(bufferedWaveProvider);
    Pitch pitch = new Pitch(stream);

    ThreadStart newThread = new ThreadStart(delegate {
        RecordAudio(waveIn, pitch, stream);
    });

    Thread myThread = new Thread(newThread) { IsBackground = true };
    myThread.Start();
}

private WaveInEvent InitWaveIn(int deviceIndex)
{
    WaveInEvent waveIn = new WaveInEvent
    {
        DeviceNumber = deviceIndex,
        WaveFormat = new WaveFormat(SAMPLE_RATE, 1)
    };
    waveIn.DataAvailable += WaveInDataAvailable;
    return waveIn;
}

private void RecordAudio(WaveInEvent waveIn, Pitch pitch, IWaveProvider stream)
{
    byte[] buffer = new byte[BUFFER_SIZE];
    int bytesRead;
    do
    {
        bytesRead = stream.Read(buffer, 0, buffer.Length);
        float freq = pitch.Get(buffer);
        if (freq != 0)
            App.MainVM.Note =  GetNote(freq);
    } while (bytesRead != 0);
    StopRecording(waveIn);
}


private void StopRecording(WaveInEvent waveIn)
{
    waveIn.StopRecording();
    waveIn.Dispose();
}



/// <summary>
/// Event for writing data into buffer.
/// </summary>
void WaveInDataAvailable(object sender, WaveInEventArgs e)
{
    if (bufferedWaveProvider != null)
    {
        bufferedWaveProvider.AddSamples(e.Buffer, 0, e.BytesRecorded);
        bufferedWaveProvider.DiscardOnBufferOverflow = true;
    }
}


public static Note GetNote(float freq)
{
```

```
            float baseFreq;
            foreach (var note in noteBaseFreqs)
            {

                baseFreq = note.Value;
                for (int i = 1; i <= MAX_NOTE_NUMBER; i++)
                {
                    double pow = Math.Pow(2, i - 1);
                    if (pow < 2)
                        pow = 2;
                    float nextFreq = Sound.GetNextFrequency(new Note(note.Key + $"{i -
1}", baseFreq, baseFreq));
                    double freqDiff = Math.Abs(baseFreq - freq);
                    double diff = nextFreq - baseFreq;
                    if (((freqDiff <= diff/2) && (nextFreq > freq) && freq < baseFreq +
diff / 2)
                        || (freq == baseFreq))
                    {
                        if(i == 1)
                            return new Note(note.Key, freq, baseFreq);
                        return new Note(note.Key + $"{i - 1}", freq, baseFreq);
                    }
                    baseFreq *= 2;
                }
            }
            return null;
        }

        public static float GetNextFrequency(Note _note)
        {
            bool isReturnNext = false;
            if(_note.Value.Remove(_note.Value.Length - 1, 1) == "B")
                    return (float)(noteBaseFreqs["C"] * Math.Pow(2, _note.SPN + 1));
            foreach (var note in noteBaseFreqs)
            {
                if (isReturnNext)
                    return (float)(note.Value * Math.Pow(2, _note.SPN));
                if (note.Key == _note.Value.Remove(_note.Value.Length - 1, 1))
                    isReturnNext = true;
            }
            return 0;
        }
    }
```

Додаток Г

BaseViewModel

```
public class BaseViewModel : INotifyPropertyChanged
    {
        #region PropertyChanged
        /// <summary>
        /// Event for updating value.
        /// </summary>
        public event PropertyChangedEventHandler PropertyChanged;
        /// <summary>
        /// Method to update bound value.
        /// </summary>
        /// <param name="prop">The name of property that has changed.</param>
        public virtual void OnPropertyChanged([CallerMemberName]string prop = "")
        {
            PropertyChanged?.Invoke(this, new PropertyChangedEventArgs(prop));
        }
        #endregion
```

```
    }
```

Додаток Д

MainVM

```
public class MainVM : BaseViewModel
    {
        public MainVM()
        {
            Sound sound = new Sound();
            sound.Detect(App.MicroWindowVM.DeviceIndex);
        }

        #region Properties

        #region Note

        private Note note = new Note("N/A", 0, 0);
        public Note Note
        {
            get => note;
            set
            {
                note = value;
                if(note != null)
                     IsHigher = note.BaseFrequency - note.Frequency > 0;
                OnPropertyChanged("Note");
            }
        }

        #endregion


        #region IsHigher

        private bool isHigher;
        public bool IsHigher
        {
            get => isHigher;
            set
            {
                isHigher = value;
                OnPropertyChanged("IsHigher");
            }
        }

        #endregion

        #endregion

        #region Commands


        #region SelectMicroCommand
        private RelayCommand selectMicroCommand;
        public RelayCommand SelectMicroCommand
        {
            get => selectMicroCommand ?? (selectMicroCommand = new RelayCommand(obj =>
            {
                MicroWindow mw = new MicroWindow();
                mw.ShowDialog();
            }));
        }
```

```
        #endregion
        #endregion
    }
```

Додаток Е

MicroWindowVM

```
public class MicroWindowVM: BaseViewModel
    {



        #region Properties


        #region DeviceIndex
        private int deviceIndex = 0;
        public int DeviceIndex
        {
            get => deviceIndex;
            set
            {
                deviceIndex = value;
                OnPropertyChanged("DeviceIndex");
            }
        }
        #endregion

        public List<string> MicroNames { get; set; } = new List<string>() { "Ritmix RDM-
126", "Defender Pulse 420" };

        #endregion


        #region Constructor
        public MicroWindowVM()
        {
            MicroNames = Sound.GetDevices().ToList();
        }
        #endregion

        #region Commands

        #region CloseCommand
        public Action CloseAction { get; set; }
        private RelayCommand closeCommand;
        public RelayCommand CloseCommand
        {
            get => closeCommand ?? (closeCommand = new RelayCommand(obj =>
            {
                DeviceIndex = 0;
                CloseAction();
            }));

        }
        #endregion

        #region SetDeviceCommand
        private RelayCommand setDeviceCommand;
        public RelayCommand SetDeviceCommand
        {
            get => setDeviceCommand ?? (setDeviceCommand = new RelayCommand(obj =>
            {
```

```
                CloseAction();
            }));

        }
        #endregion

        #endregion

    }
```

Додаток Є

RelayCommand

```
public class RelayCommand: ICommand
    {
        private Action<object> execute;
        private Func<object, bool> canExecute;

        public event EventHandler CanExecuteChanged
        {
            add { CommandManager.RequerySuggested += value; }
            remove { CommandManager.RequerySuggested -= value; }
        }

        public RelayCommand(Action<object> execute, Func<object, bool> canExecute = null)
        {
            this.execute = execute;
            this.canExecute = canExecute;
        }

        public bool CanExecute(object parameter)
        {
            return this.canExecute == null || this.canExecute(parameter);
        }

        public void Execute(object parameter)
        {
            this.execute(parameter);
        }
    }
```

Київський національний торговельно-економічний університет

Рецензія на курсову роботу (проект) і результат захисту

Студентки __ Войткевич Аліси Андріївни
(прізвище, ім'я та по батькові)

___2___ курсу __12____ групи _____ФІТ_____факультету

Курсова робота (проект) з _____«Об'єктно-орієнтованого програмування»_____
(назва навчальної дисципліни)

Тема Розробка програмного додатку для налаштування струнного інструменту мовою програмування C#

Реєстраційний № ____, дата одержання 11.05.2021р.

Науковий керівник _____ асист. каф. Гнатченко Д. Д._____
(вчене звання, прізвище, ініціали)

Зміст рецензії

У курсовій роботі студентки Войткевич А. А. на тему «Розробка програмного додатку для налаштування струнного інструменту мовою програмування C#» зазначено актуальність дослідження, мету, об'єкт, предмет та задачі дослідження.

У теоретичній частині курсової роботи студентка розглянула базові концепції об'єктно–орієнтованого програмування, середовища та мови програмування. Автором детально описано та охарактеризовано принципи ООП. Також було розглянуто питання актуальності даного додатку, проаналізовано ринок на вже існуючі аналогічні додатки. Для реалізації інтерфейсу користувача, автор розробив подання для відповідних форм програмного модулю інтерфейсу користувача. У третьому розділі подано детальний опис розробки програмного додатку за допомогою мови C# у середовищі Visual Studio. В цілому курсова робота студентки Войткевич А. А. на тему «Розробка програмного додатку для налаштування струнного інструменту мовою програмування C#» оформлена у відповідності до Вимог, може бути допущена до захисту та заслуговує на позитивну оцінку.

Допущено до захисту “11”____05_____ 2021 р.

Захист планується о _10.00_____ “__14___”_____05___ 2021 р.

(час)

кафедра інженерії програмного забезпечення та кібербезпеки

(місце роботи комісії)

_____

(підпис наукового керівника)

Курсова робота захищена “_14_”___05_____ 2021 р.
з оцінкою __«відмінно» 94 бали_____

(за шкалою КНТЕУ, національною шкалою та шкалою ЄКТС)

Комісія:

1. _____ ___Бебешко Б. Т._____
(підпис) (прізвище, ініціали)

2. _____ ___Жирова Т.О._____
(підпис) (прізвище, ініціали)

3. _____ _Хорольська К.В._____
(підпис) (прізвище, ініціали)