Полтавський університет економіки і торгівлі
Навчально-науковий інститут денної освіти
Форма навчання денна
Кафедра комп’ютерних наук та інформаційних технологій

Допускається до захисту
Завідувач кафедри
_____Олена ОЛЬХОВСЬКА
(підпис)

«___»_____202_ р.

# КВАЛІФІКАЦІЙНА РОБОТА
**на тему**
# «РОЗРОБКА НАВЧАЛЬНОГО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ З ТЕМИ «STREAM API В JAVA» ДИСЦИПЛІНИ «СУЧАСНІ ПАРАДИГМИ ПРОГРАМУВАННЯ»

**зі спеціальності 122 «Комп’ютерні науки»**
**освітня програма «Комп’ютерні науки»**
**ступеня бакалавр**

**Виконавець роботи** Мандрика Дмитро Романович
_____«__»_____202_р. (підпис)
**Науковий керівник** к.ф.-м.н.,доц., Олексійчук Юрій Федорович
_____«__»_____202_р. (підпис)
**Рецензент**

**ПОЛТАВА 2026**

## РЕФЕРАТ

JAVA, STREAM API, STREAMLEARN, ФУНКЦІОНАЛЬНЕ ПРОГРАМУВАННЯ, НАВЧАЛЬНИЙ ЗАСТОСУНОК, JAVAFX, ПОТОКОВА ОБРОБКА ДАНИХ.

Кваліфікаційна робота присвячена розробці навчального програмного застосунку **StreamLearn** для вивчення технології Stream API мови програмування Java.

**Об'єктом дослідження** є процес навчання використанню технології Stream API у мові програмування Java.

Предметом дослідження є методи та програмні засоби навчання потоковій обробці даних із використанням Stream API.

**Метою роботи** є розробка навчального програмного застосунку, який забезпечує вивчення теоретичних основ Stream API, демонстрацію прикладів використання потоків даних, перевірку знань користувачів та виконання практичних вправ.

У процесі виконання роботи використано методи системного аналізу, об'єктно-орієнтованого проєктування, алгоритмізації, програмування мовою Java та тестування програмного забезпечення. Для реалізації програмного продукту використано мову програмування Java, технологію JavaFX для створення графічного інтерфейсу користувача та систему керування залежностями Maven.

У результаті виконання роботи проведено аналіз сучасного навчального програмного забезпечення та можливостей технології Stream API, спроєктовано архітектуру програмного продукту, розроблено модулі теоретичних матеріалів, демонстраційних прикладів, тестування знань, практичних вправ і оцінювання результатів навчання. Реалізовано навчальний застосунок StreamLearn, який

забезпечує вивчення потокової обробки даних у середовищі Java та підтримує контроль рівня засвоєння навчального матеріалу.

Проведене функціональне тестування підтвердило коректність роботи всіх основних модулів системи та відповідність реалізованого програмного забезпечення поставленим вимогам.

Практичне значення роботи полягає у можливості використання розробленого програмного продукту в навчальному процесі закладів вищої освіти, а також для самостійного вивчення сучасних засобів функціонального програмування мовою Java.

**ЗМІСТ**

## ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, СКОРОЧЕНЬ І ТЕРМІНІВ

| **Умовні позначення, символи, скорочення, терміни** | **Пояснення умовних позначень, скорочень, символів** |
|---|---|
| API | Application Programming Interface (інтерфейс програмування застосунків) |
| Stream API | Технологія обробки потоків даних у Java |
| JVM | Java Virtual Machine (віртуальна машина Java) |
| JDK | Java Development Kit (комплект засобів розробки Java) |
| JRE | Java Runtime Environment (середовище виконання Java) |
| JavaFX | Платформа для створення графічних інтерфейсів користувача |
| GUI | Graphical User Interface (графічний інтерфейс користувача) |
| Maven | Система автоматизації складання та керування залежностями проєкту |
| IDE | Integrated Development Environment (інтегроване середовище розробки) |

| | |
|---|---|
| OOP | Object-Oriented Programming (об'єктноорієнтоване програмування) |
| CRUD | Create, Read, Update, Delete (основні операції роботи з даними) |
| List | Колекція впорядкованих елементів у Java |
| Filter | Проміжна операція Stream API для відбору елементів |
| Map | Проміжна операція Stream API для перетворення елементів |
| Reduce | Термінальна операція Stream API для агрегації результатів |
| Collect | Термінальна операція Stream API для збору результатів обробки |
| Lambda-вираз | Анонімна функція, що використовується у функціональному програмуванні Java |
| Stream | Потік даних для послідовної або паралельної обробки елементів |
| StreamLearn | Розроблений у роботі навчальний програмний застосунок |

# ВСТУП

Сучасний етап розвитку інформаційних технологій характеризується активним впровадженням цифрових інструментів у сферу освіти. Особливої актуальності набуває використання навчального програмного забезпечення, яке забезпечує інтерактивну взаємодію користувача з навчальним матеріалом, підвищує ефективність засвоєння знань та сприяє формуванню практичних навичок програмування. У підготовці фахівців з комп'ютерних наук важливе місце займає вивчення сучасних парадигм програмування, серед яких функціональний підхід відіграє одну з ключових ролей.

У мові програмування Java, починаючи з версії 8, функціональні можливості реалізовані за допомогою таких механізмів, як лямбда-вирази, функціональні інтерфейси та Stream API. Зокрема, Stream API надає зручні засоби для декларативної обробки колекцій даних, дозволяючи зосередитися на логіці перетворень, а не на деталях реалізації алгоритмів. Вивчення цієї технології є важливим елементом формування сучасного програмного мислення, однак традиційні підходи до навчання часто не забезпечують достатнього рівня практичного засвоєння матеріалу.

Актуальність теми полягає у необхідності створення ефективного навчального програмного забезпечення, яке поєднує теоретичні відомості з практичними прикладами використання Stream API та забезпечує інтерактивну підтримку навчального процесу. Розробка такого програмного продукту дозволяє покращити розуміння принципів функціонального програмування, підвищити рівень залученості студентів та сформувати навички застосування сучасних інструментів обробки даних.

Метою кваліфікаційної роботи є розробка навчального програмного забезпечення з теми «Stream API в Java», яке забезпечує

демонстрацію основних можливостей потокової обробки даних, а також аналіз ефективності використання функціонального підходу у порівнянні з імперативним.

Для досягнення поставленої мети необхідно вирішити такі завдання:

- проаналізувати сучасні підходи до створення навчального програмного забезпечення;
- дослідити теоретичні основи функціонального програмування та Stream API у Java;
- обґрунтувати вибір технологій для реалізації програмного продукту;
- спроєктувати архітектуру навчального застосунку;
- реалізувати програмний продукт із використанням Stream API;
- провести тестування та оцінку ефективності розробленого рішення;
- виконати порівняльний аналіз функціонального та імперативного підходів до обробки даних.

Об’єктом дослідження є процес навчання програмуванню з використанням сучасних парадигм.

Предметом дослідження є методи та програмні засоби реалізації потокової обробки даних у мові Java із застосуванням Stream API у вигляді навчального програмного забезпечення.

Методи дослідження включають аналіз наукової та технічної літератури, порівняльний аналіз підходів до обробки даних, методи об’єктно-орієнтованого та функціонального програмування, а також експериментальну перевірку ефективності розробленого програмного продукту.

Наукова новизна одержаних результатів полягає у розробці навчального програмного застосунку, який поєднує демонстраційні, аналітичні та інтерактивні можливості для вивчення Stream API, а

також у проведенні порівняльного аналізу ефективності функціонального та імперативного підходів до обробки даних.

Практичне значення роботи полягає у можливості використання розробленого програмного забезпечення у навчальному процесі під час викладання дисципліни «Сучасні парадигми програмування», а також для самостійного вивчення студентами принципів роботи Stream API.

# 1 ПОСТАНОВКА ЗАДАЧІ

Сучасний розвиток програмної інженерії характеризується активним використанням функціональних підходів до обробки даних. Одним із важливих інструментів реалізації функціонального програмування в мові

Java є технологія Stream API, яка була впроваджена починаючи з версії Java 8. Використання потоків даних дозволяє значно спростити реалізацію алгоритмів обробки колекцій, підвищити читабельність програмного коду та забезпечити ефективне використання обчислювальних ресурсів.

Незважаючи на широке застосування Stream API у сучасній розробці програмного забезпечення, її вивчення викликає певні труднощі у студентів. Це пов’язано зі зміною підходу до програмування від імперативного до декларативного стилю, необхідністю розуміння лямбдавиразів, функціональних інтерфейсів та принципів побудови конвеєрів обробки даних. Традиційні методи навчання, що базуються переважно на теоретичному викладі матеріалу, не завжди забезпечують достатній рівень засвоєння практичних навичок використання Stream API.

Для підвищення ефективності навчального процесу доцільним є створення спеціалізованого програмного продукту, який поєднуватиме теоретичні відомості, інтерактивні приклади, практичні вправи та засоби контролю знань. Такий підхід дозволить забезпечити активну взаємодію користувача з навчальним матеріалом, підвищити рівень зацікавленості та сприяти формуванню практичних навичок роботи з потоками даних у Java.  Метою роботи є розробка навчального програмного продукту для вивчення технології Stream API в мові програмування Java, який забезпечує подання теоретичного матеріалу,

виконання практичних завдань, проходження тестування та оцінювання рівня знань користувача.

Для досягнення поставленої мети необхідно вирішити такі завдання: Проаналізувати сучасні підходи до створення навчального програмного забезпечення;

Дослідити теоретичні основи функціонального програмування та технології

Stream API;

Визначити вимоги до навчального програмного продукту;

Спроєктувати структуру та архітектуру програмної системи;

Реалізувати модулі подання теоретичного матеріалу;

Реалізувати систему тестування знань користувача;

Розробити модуль практичних вправ із перевіркою правильності відповідей;

Реалізувати систему оцінювання результатів навчання;

Провести тестування розробленого програмного продукту та оцінити його працездатність.

**Функціональні вимоги до системи**

Розроблений програмний продукт повинен забезпечувати реалізацію основних функцій, необхідних для вивчення технології Stream API у мові програмування Java.

Основними функціональними вимогами є:

- надання користувачу доступу до теоретичних матеріалів щодо використання Stream API;
- демонстрація прикладів використання основних потокових операцій;
- організація тестування знань користувача;

- надання практичних вправ для закріплення навчального матеріалу;
- автоматична перевірка відповідей користувача;
- відображення результатів тестування та практичних вправ;
- обчислення загальної успішності навчання;
- забезпечення зручної навігації між функціональними модулями системи.

Для реалізації поставлених вимог програмний продукт містить такі функціональні модулі:

Модуль теоретичних матеріалів.

Модуль демонстраційних прикладів Stream API.

Модуль тестування знань.

Модуль практичних вправ.

Модуль відображення результатів навчання.

**Опис функціональних можливостей**

Модуль теоретичних матеріалів призначений для ознайомлення користувача з основними поняттями Stream API.

У даному модулі розглядаються:

• поняття Stream API;

• принцип lazy evaluation;

• pipeline обробки даних;

• проміжні операції;

• термінальні операції;

• особливості використання потоків у Java.

Модуль демонстраційних прикладів містить приклади використання найбільш поширених операцій Stream API:

• filter();

• map();

• sorted();

• count();

• reduce();

• collect().

Кожен приклад містить фрагмент програмного коду та результат його виконання.

Модуль тестування забезпечує перевірку рівня знань користувача за допомогою тестових запитань із вибором однієї правильної відповіді.

Приклади запитань:

• Що таке Stream?

• Що робить filter()?

• Що повертає map()?

• Що таке lazy evaluation?

• Для чого використовується reduce()?

Після завершення тестування система автоматично визначає кількість правильних відповідей та відображає результат у форматі:

Score: X / 10.

Модуль практичних вправ використовується для закріплення знань щодо використання Stream API.

Користувачу пропонуються завдання на:

• використання filter();

• використання map();

• використання count();

• побудову ланцюжків потокових операцій;

• використання collect(Collectors.toList()).

Після введення відповіді система автоматично виконує перевірку та повідомляє користувача про правильність виконання завдання.

Модуль результатів навчання відображає:

• результат останнього тестування;

• результат останньої практики;

• загальний бал;

• відсоток успішності.

**Нефункціональні вимоги**

До основних нефункціональних вимог належать:

• простота використання інтерфейсу;

• швидке виконання основних операцій;

• коректна обробка дій користувача;

• стабільна робота застосунку;

• можливість запуску на персональному комп’ютері без підключення до мережі Інтернет;

• можливість подальшого розширення функціональності.

**Критерії ефективності**

Ефективність розробленого програмного продукту визначається за такими критеріями:

• зручність використання інтерфейсу;

• повнота подання навчального матеріалу;

• коректність перевірки відповідей;

• швидкість виконання функцій системи;

• ефективність засвоєння знань користувачем;

• можливість використання програмного продукту під час самостійного навчання.

Отже, результатом виконання роботи є створення навчального програмного продукту StreamLearn, який забезпечує вивчення технології Stream API за допомогою поєднання теоретичних матеріалів, демонстраційних прикладів, тестування знань та практичних вправ у межах єдиного програмного середовища.

**Основні модулі програмного продукту**

З метою забезпечення логічної структури та зручності використання система повинна складатися з таких основних модулів:

**Модуль теоретичного навчання**

Модуль забезпечує доступ до структурованого теоретичного матеріалу щодо використання Stream API. У його межах користувач може ознайомитися з основними поняттями, принципами роботи потоків даних, особливостями використання лямбда-виразів та функціональних інтерфейсів.

**Модуль демонстраційних прикладів**

- Призначений для демонстрації прикладів використання основних операцій Stream API, зокрема:
- filter();
- map();
- sorted();
- distinct();
- limit();
- reduce(); • collect();

• groupingBy().

**Модуль**

**тестування**

Забезпечує проведення контролю знань шляхом проходження тестових завдань. Після завершення тестування система автоматично визначає результат та відображає кількість правильних відповідей.

**Модуль практичних вправ**

Дозволяє користувачу виконувати завдання з використанням Stream API, аналізувати приклади програмного коду та перевіряти правильність власних рішень.

**Модуль оцінювання результатів**

Призначений для підрахунку балів, визначення рівня підготовки користувача та формування статистики навчання.

**Обмеження системи**

Під час розробки програмного продукту враховуються такі обмеження:
використання мови програмування Java; орієнтація на навчальні цілі;
локальне використання без обов’язкового підключення до мережі Інтернет;
обмежений обсяг навчальних матеріалів у межах демонстраційної версії; відсутність інтеграції із зовнішніми інформаційними системами.

# 2 ІНФОРМАЦІЙНИЙ ОГЛЯД

## 2.1 Огляд навчального програмного забезпечення

Сучасний розвиток інформаційних технологій сприяв активному впровадженню електронних засобів навчання у освітній процес. Навчальне програмне забезпечення (НПЗ) відіграє важливу роль у підготовці фахівців у галузі комп'ютерних наук, оскільки забезпечує інтерактивну взаємодію користувача з навчальним матеріалом, сприяє формуванню практичних навичок та підвищує ефективність засвоєння знань.

Навчальне програмне забезпечення можна визначити як сукупність програмних засобів, призначених для підтримки навчального процесу, що включає подання теоретичного матеріалу, організацію практичних занять, контроль знань та самостійну роботу користувачів. Основною особливістю сучасного НПЗ є його орієнтація на активну участь користувача в навчальному процесі, що реалізується через інтерактивні елементи, практичні завдання та миттєвий зворотний зв'язок.

### Класифікація навчального програмного забезпечення

Залежно від функціонального призначення, навчальне програмне забезпечення можна поділити на такі основні типи:

- **навчальні системи (tutorial systems)** — призначені для подання теоретичного матеріалу у структурованому вигляді;
- **тренажери (simulators)** — забезпечують відпрацювання практичних навичок;

- **контролюючі системи** — використовуються для перевірки знань та оцінювання результатів навчання;
- **інтелектуальні навчальні системи** — адаптуються до рівня підготовки користувача;
- **інтерактивні середовища програмування** — дозволяють виконувати та аналізувати програмний код у реальному часі.

У контексті навчання програмуванню особливу роль відіграють інтерактивні середовища, які поєднують теоретичні пояснення з практичними завданнями та можливістю негайного виконання коду.

## Сучасні підходи до розробки НПЗ

Серед основних підходів до створення навчального програмного забезпечення можна виділити:

- **компетентнісно-орієнтований підхід**, який передбачає формування практичних навичок поряд із засвоєнням теоретичних знань;
- **модульний підхід**, що забезпечує поділ навчального матеріалу на логічно завершені блоки;
- **інтерактивність**, яка реалізується через активну взаємодію користувача з системою;
- **адаптивність**, що дозволяє враховувати індивідуальний рівень підготовки користувача;
- **поєднання теорії та практики**, що є особливо важливим для дисциплін, пов’язаних із програмуванням.

Застосування цих підходів дозволяє створювати ефективні навчальні системи, які відповідають сучасним вимогам освітнього процесу.

## Порівняння онлайн та локального навчального програмного забезпечення

Навчальні програмні системи можуть реалізовуватися у вигляді онлайн-платформ або локальних застосунків. Кожен із підходів має свої переваги та недоліки.

Таблиця 2.1 - Порівняння онлайн та локального навчального програмного забезпечення

| **Критерій** | **Онлайн-системи** | **Локальні застосунки** |
|---|---|---|
| Доступність | доступ з будь-якого пристрою | обмежена встановленою системою |
| Оновлення | автоматичне | потребує перевстановлення |
| Продуктивність | залежить від мережі | стабільна |
| Інтерактивність | висока | висока |
| Залежність від інтернету | висока | відсутня |

У межах даної роботи обрано реалізацію у вигляді локального застосунку, що дозволяє забезпечити стабільність роботи, простоту використання та незалежність від мережевих ресурсів.

## Особливості навчального ПЗ для програмування

Навчальні програмні продукти у сфері програмування мають специфічні вимоги, зокрема:

- можливість виконання програмного коду;

- наочна демонстрація алгоритмів;
- підтримка різних стилів програмування;
- автоматична перевірка результатів;
- аналіз помилок.

Особливо важливим є використання інтерактивних прикладів, які дозволяють користувачеві змінювати параметри програм та спостерігати результати виконання. Це сприяє глибшому розумінню принципів роботи програмного коду.

**Висновки до підрозділу**

Таким чином, сучасне навчальне програмне забезпечення орієнтоване на інтерактивність, практичну спрямованість та адаптивність. Найбільш ефективними є системи, що поєднують теоретичний матеріал із практичними завданнями та забезпечують можливість безпосередньої взаємодії користувача з програмним кодом.

У контексті теми даної роботи доцільним є створення навчального програмного застосунку, який дозволяє демонструвати можливості Stream API у Java, забезпечуючи поєднання теоретичних пояснень із практичною реалізацією операцій обробки даних.

**Огляд існуючого навчального програмного забезпечення**

Як аналог під час розробки програмного забезпечення було розглянуто навчальний тренажер з теми «Теореми додавання та множення ймовірностей випадкових подій» дистанційного навчального курсу «Теорія ймовірностей і математична статистика», розроблений Д. А. Дудником та Т. О. Парфьоновою.[18]

Програмний засіб призначений для підтримки навчального процесу та формування практичних навичок розв’язування задач з теорії ймовірностей. Основною метою тренажера є допомога студентам у засвоєнні правил знаходження ймовірностей випадкових

подій, а також закріплення теоретичних знань шляхом виконання практичних завдань.

Особливістю даної розробки є використання покрокового алгоритму розв’язання задач. Робота тренажера базується на послідовному виконанні етапів знаходження ймовірності суми несумісних подій. Такий підхід дозволяє студенту не лише отримати правильний результат, а й зрозуміти логіку розв’язання задачі. Під час виконання завдань користувач взаємодіє з програмою, отримує проміжні результати та контролює правильність виконання окремих кроків.

До переваг програмного засобу можна віднести наочність навчання, покроковий контроль процесу розв’язання, можливість самостійного опрацювання матеріалу та автоматизацію перевірки правильності виконання завдань. Використання тренажера сприяє кращому засвоєнню складних математичних понять і підвищує ефективність самостійної роботи студентів.

Проведений аналіз показав, що використання навчальних тренажерів є ефективним засобом підтримки освітнього процесу. Досвід реалізації даного програмного продукту доцільно врахувати під час розробки власної системи, зокрема щодо організації покрокового виконання завдань, контролю правильності дій користувача та забезпечення інтерактивної взаємодії з програмою.

## **2.2** Призначення та можливості Stream API

У сучасній розробці програмного забезпечення важливу роль відіграють інструменти, що дозволяють ефективно працювати з великими обсягами даних. У мові програмування Java одним із таких інструментів є Stream API, який було введено починаючи з версії Java 8 як частину підтримки функціонального програмування.

### **Призначення Stream API**

Stream API призначений для декларативної обробки послідовностей даних, зокрема колекцій, і дозволяє описувати операції над даними без деталізації механізму їх виконання [5, 6]. Основна ідея

полягає у тому, щоб описати, які операції необхідно виконати над даними, не вказуючи детально, як саме ці операції реалізуються.

На відміну від традиційного імперативного підходу, де використовуються цикли та змінні стану, Stream API дозволяє будувати ланцюжки операцій, що виконуються над потоком даних. Це забезпечує більш компактний, читабельний та гнучкий код.

Відповідно до документації Oracle, потік (Stream) є послідовністю елементів, яка підтримує сукупність операцій для виконання обчислень над даними [6]. При цьому потоки не зберігають дані, а лише визначають спосіб їх обробки.

**Основні можливості Stream API**

Stream API надає широкий набір функціональних можливостей для роботи з даними:

***1. Фільтрація даних***

Дозволяє відбирати елементи відповідно до заданих умов за допомогою операції filter().

***2. Перетворення елементів***

Операція map() використовується для перетворення кожного елемента потоку в інше значення та є однією з основних проміжних операцій Stream API [6].

***3. Агрегація результатів***

За допомогою операції reduce() або collect() можна об’єднувати елементи потоку в єдиний результат або структуру даних.

***4. Сортування***

Метод sorted() дозволяє впорядковувати елементи потоку.

***5. Групування даних***

Застосування Collectors.groupingBy() забезпечує класифікацію елементів за певними ознаками.

***6. Паралельна обробка***

Stream API підтримує паралельну обробку даних за допомогою методу parallelStream(), що дозволяє використовувати багатоядерні процесори без явного керування потоками [18].

**Переваги використання Stream API**

Використання Stream API має низку суттєвих переваг:

- **декларативний стиль програмування** — описується результат, а не процес;
- **зменшення обсягу коду** — відсутність необхідності використання циклів та допоміжних змінних;
- **покращення читабельності** — логіка обробки даних представлена у вигляді послідовності операцій;
- **можливість паралелізації** — спрощене використання багатопотоковості;
- **відсутність побічних ефектів** — потоки не змінюють вихідні дані.

**Недоліки та обмеження**

Попри значні переваги, Stream API має певні обмеження:

- складність сприйняття для початківців;
- у деяких випадках зниження продуктивності через додаткові накладні витрати;
- складність налагодження при довгих ланцюжках операцій;
- неефективність при простих задачах, де використання циклів є більш доцільним.

**Порівняння Stream API та імперативного підходу**

Одним із важливих аспектів дослідження є порівняння функціонального підходу, реалізованого через Stream API, та класичного імперативного підходу.

Таблиця 2.2 - Порівняння онлайн та локального навчального програмного забезпечення

| **Критерій** | **Імперативний підхід** | **Stream API** |
|---|---|---|
| Стиль програмування | покроковий | декларативний |
| Кількість коду | більша | менша |
| Читабельність | середня | висока |
| Паралелізація | складна | проста |
| Гнучкість | обмежена | висока |

***Приклад порівняння***

Імперативний підхід:

```
List<Integer> result = new ArrayList<>(); for (Integer n : numbers)
{   if (n % 2 ==
0) {      result.add(n * n);
  }
}
```

Stream API:

```
List<Integer> result = numbers.stream()
    .filter(n -> n % 2 == 0)
    .map(n -> n * n)
    .collect(Collectors.toList());
```

У другому випадку код є більш компактним і зрозумілим, оскільки відображає логіку обробки даних у вигляді послідовності операцій.

**Освітнє значення Stream API**

Stream API є важливим інструментом для навчання сучасних підходів до програмування, оскільки дозволяє:

- ознайомити студентів із функціональною парадигмою;
- сформувати навички роботи з потоками даних;
- розвинути абстрактне мислення;
- підготувати до використання сучасних технологій у професійній діяльності.

**Висновки до підрозділу**

Таким чином, Stream API є потужним інструментом для обробки даних у мові Java, що реалізує принципи функціонального програмування та забезпечує ефективну роботу з колекціями. Його використання дозволяє підвищити читабельність коду, скоротити його обсяг та спростити реалізацію складних операцій обробки даних.

У навчальному процесі застосування Stream API є доцільним, оскільки сприяє формуванню сучасного програмного мислення та розвитку практичних навичок програмування.

# 3 ТЕОРЕТИЧНА ЧАСТИНА

## 3.1 Обґрунтування вибору стеку технологій

Одним із важливих етапів розробки програмного забезпечення є вибір технологій, які забезпечують ефективну реалізацію поставлених завдань, зручність використання та можливість подальшого розвитку програмного продукту. Під час створення навчального застосунку для вивчення технології Stream API в Java вибір програмних засобів здійснювався з урахуванням навчального призначення системи, функціональних вимог та сучасних тенденцій розвитку програмної інженерії.

**Вибір мови програмування**

Для реалізації програмного продукту обрано мову програмування Java, оскільки вона підтримує об’єктно-орієнтовану та функціональну парадигми програмування, а також забезпечує платформонезалежність завдяки використанню JVM [1, 3, 4].

Вибір Java є особливо доцільним, оскільки саме ця мова містить вбудовану реалізацію Stream API, що є основним об’єктом вивчення у розробленому програмному продукті.

**Використання Stream API**

Центральним елементом навчального застосунку є технологія Stream API, яка була представлена в Java 8 для підтримки функціонального стилю обробки даних [9, 10, 28].

У межах програмного продукту передбачено демонстрацію таких операцій Stream API:

- filter();
- map();
- sorted();
- distinct();
- limit();
- skip();
- reduce();
- collect();
- groupingBy();
- counting().

Використання Stream API дозволяє на практичних прикладах продемонструвати принципи декларативного програмування та особливості побудови конвеєрів обробки даних.

**Вибір технології JavaFX**

Для створення графічного інтерфейсу користувача використано технологію JavaFX.

Основними перевагами JavaFX є:

- підтримка сучасного графічного інтерфейсу;
- наявність великої кількості готових елементів керування;
- простота створення навчальних застосунків;
- можливість використання CSS для стилізації інтерфейсу;
- інтеграція з платформою Java.

Використання графічного інтерфейсу дозволяє зробити процес навчання більш зручним та наочним порівняно з консольними застосунками.

**Використання Maven**

Для автоматизації процесу складання проєкту використано систему керування залежностями Maven, що забезпечує стандартизовану структуру проєкту та автоматичне керування бібліотеками [17].

Використання Maven дозволяє забезпечити зручну організацію програмного коду та підвищити ефективність розробки.

**Архітектура програмного продукту**

Під час розробки застосунку використано модульну архітектуру, що передбачає поділ системи на окремі функціональні компоненти. Основними модулями системи є:

- модуль теоретичних матеріалів;
- модуль демонстраційних прикладів;
- модуль тестування знань;
- модуль практичних вправ;
- модуль оцінювання результатів;
- модуль збереження даних.

Такий підхід забезпечує незалежність окремих частин системи та спрощує її подальший розвиток.

**Середовище розробки**

Для створення програмного продукту використано інтегроване середовище розробки IntelliJ IDEA.
Перевагами даного середовища є:

- підтримка Java та JavaFX;
- автоматичне форматування коду;
- вбудовані засоби налагодження;
- підтримка Maven-проєктів; • інструменти рефакторингу.

Використання IntelliJ IDEA дозволяє підвищити продуктивність розробки та якість програмного коду.

**Висновки до підрозділу**

У результаті аналізу було обрано стек технологій, який повністю відповідає поставленим завданням розробки навчального програмного продукту. Використання Java, Stream API, JavaFX та Maven забезпечує створення сучасного програмного застосунку, що дозволяє ефективно вивчати принципи потокової обробки даних та функціонального програмування у середовищі Java.

## 3.2 Алгоритмізація роботи застосунку

Алгоритмізація є важливим етапом розробки програмного забезпечення, оскільки дозволяє формалізувати логіку роботи системи та визначити послідовність виконання основних операцій. У межах даної роботи було розроблено навчальний програмний продукт StreamLearn, призначений для вивчення технології Stream API в мові програмування Java.

Застосунок реалізовано з використанням технології JavaFX. Основна логіка роботи системи зосереджена у класі MainController, який забезпечує взаємодію користувача з усіма функціональними модулями програмного продукту.

***Алгоритм запуску застосунку***

Робота програми починається із запуску головного класу StreamLearnApp.

Послідовність виконання алгоритму:

1. Виконується запуск JavaFX-застосунку.
2. Завантажується файл графічного інтерфейсу main.fxml.
3. Створюється об’єкт Scene розміром 1000×700 пікселів.
4. Підключається файл стилів style.css.
5. Відображається головне вікно програми.
6. Користувач отримує доступ до функціональних можливостей системи.

Фрагмент реалізації запуску застосунку:

```
FXMLLoader loader = new FXMLLoader(
        getClass().getResource("/main.fxml")
);

Scene scene = new Scene(loader.load(), 1000, 700);

stage.setTitle("StreamLearn - Java Stream API");
stage.setScene(scene); stage.show();
```

Такий підхід забезпечує коректну ініціалізацію графічного інтерфейсу та підготовку середовища до подальшої роботи користувача.

### *Алгоритм роботи модуля теоретичних матеріалів*

Модуль теоретичних матеріалів призначений для ознайомлення користувача з основами Stream API.

Алгоритм роботи модуля:

1. Користувач обирає розділ «Теорія».
2. Викликається метод showTheory().
3. Відображається текстовий матеріал.
4. Користувач переглядає інформацію щодо Stream API.
5. За необхідності виконується перехід до інших модулів системи.

У теоретичному модулі користувач отримує інформацію про:

• поняття Stream API;

• lazy evaluation;

• pipeline;

• проміжні операції;

• термінальні операції;

• особливості потокової обробки даних.

Матеріал відображається у компоненті TextArea без виконання додаткових обчислень, тому часова складність роботи модуля наближена до O(1).

### *Алгоритм роботи модуля демонстраційних прикладів*

Модуль прикладів демонструє практичне використання Stream API.

Послідовність роботи алгоритму:

1. Користувач відкриває розділ «Приклади».

2. Викликається метод showExamples().
3. Система формує набір прикладів використання Stream API.
4. Приклади відображаються у текстовому полі.
5. Користувач аналізує результати виконання операцій.

У модулі демонструються приклади використання:

• filter();

• map();

• sorted();

• count();

• reduce(); • collect().

Приклад операції filter():

```
numbers.stream()
      .filter(n -> n > 3)
      .forEach(System.out::println);
```

Приклад операції map():

```
names.stream()
    .map(String::toUpperCase)
    .forEach(System.out::println);
```

Приклад агрегування результатів:

```
int sum = numbers.stream()
            .reduce(0, Integer::sum);
```

Використання прикладів дозволяє сформувати практичне розуміння механізму роботи потоків даних.

### *Алгоритм роботи модуля тестування*

Одним із основних елементів системи є модуль контролю знань.

Модуль містить десять тестових запитань щодо використання Stream API.

Алгоритм роботи:

1. Користувач обирає режим тестування.
2. Викликається метод showQuiz().
3. Ініціалізуються змінні index та score.
4. Завантажується поточне запитання.
5. Користувач обирає один із варіантів відповіді.
6. Виконується перевірка правильності відповіді.
7. У разі правильної відповіді збільшується значення score.
8. Завантажується наступне запитання.
9. Після завершення тесту відображається підсумковий результат.

Перевірка відповіді реалізована таким чином:

```
if (selectedIndex == correct[index]) {
score++;
}
```

Часова складність проходження тесту становить O(n), де n – кількість запитань.

### *Алгоритм роботи модуля практичних вправ*

Для закріплення теоретичних знань реалізовано модуль практичних завдань.

Алгоритм роботи:

1. Користувач відкриває розділ практичних вправ.

2. Генерується набір завдань.
3. Формується випадкове значення для завдань фільтрації.
4. Відображається поточне завдання.
5. Користувач вводить відповідь.
6. Виконується перевірка правильності введених даних.
7. Оновлюється статистика виконання.
8. Відображається наступне завдання.
9. Після завершення вправ формується підсумковий результат.

Приклади завдань:

• Відфільтруй числа більше заданого значення;

• Помнож кожен елемент на 2;

• Порахуй кількість елементів; • Перетвори елементи у квадрат;

• Збери результат у список.

Перевірка відповіді здійснюється методом:

```
public boolean check(String userAnswer) {
return correctAnswer.equalsIgnoreCase(
userAnswer.trim()
   );
}
```

Даний підхід дозволяє автоматизувати процес перевірки відповідей користувача.

***Алгоритм формування результатів навчання***

Після проходження тестування або практичних вправ система формує статистику успішності користувача.

Алгоритм роботи:

1. Отримуються результати тестування.

2. Отримуються результати практичних вправ.
3. Обчислюється загальна кількість балів.
4. Визначається максимальна можлива кількість балів.
5. Обчислюється відсоток успішності.
6. Формується текстовий звіт.
7. Результати відображаються користувачу.

Обчислення успішності виконується за формулою:
успішність = (отримані бали / максимальні бали) × 100 %

Отримані результати дозволяють оцінити рівень засвоєння навчального матеріалу та ефективність проходження курсу.

***Висновки до підрозділу***

У результаті алгоритмізації було визначено логіку роботи всіх основних модулів програмного продукту StreamLearn. Розроблені алгоритми забезпечують послідовне вивчення теоретичних матеріалів, проходження тестування, виконання практичних вправ та аналіз отриманих результатів. Використання модульного підходу спрощує підтримку програмного забезпечення та створює можливості для його подальшого розвитку.

## 3.3 Проєктування застосунку

Проєктування програмного забезпечення є одним із найважливіших етапів розробки, оскільки саме на цьому етапі визначаються структура системи, взаємодія між компонентами та принципи організації програмного коду. Для реалізації навчального застосунку StreamLearn було обрано модульний підхід, який забезпечує простоту супроводу, зрозумілу структуру та можливість подальшого розширення функціональності.

## Загальна архітектура системи

Програмний продукт реалізовано з використанням технології JavaFX та архітектури, заснованої на розділенні графічного інтерфейсу та логіки керування.

До складу застосунку входять такі основні компоненти:

• модуль відображення теоретичних матеріалів;

• модуль демонстраційних прикладів Stream API;

• модуль тестування знань;

• модуль практичних вправ;

• модуль формування результатів навчання;

• графічний інтерфейс користувача.

Взаємодія між модулями здійснюється через центральний клас керування MainController відповідно до принципів розподілу відповідальності між компонентами програмної системи [11].

Завдяки такому підходу кожен модуль відповідає за окрему функціональну частину системи, що позитивно впливає на підтримуваність програмного коду.

## Структура програмного продукту

Логічна структура програмного забезпечення представлена сукупністю взаємопов’язаних класів.

### *Клас StreamLearnApp*

Клас StreamLearnApp є головним класом програми та відповідає за запуск JavaFX-застосунку.

Основні функції:

• запуск програми;

• завантаження інтерфейсу main.fxml;

• підключення файлу стилів style.css;
• створення головного вікна застосунку; • запуск графічного інтерфейсу користувача.

Фрагмент коду запуску:

```
FXMLLoader loader =
new FXMLLoader(
        getClass().getResource("/main.fxml")
    );

Scene scene = new Scene(loader.load(), 1000, 700);

stage.setTitle("StreamLearn - Java Stream API");
stage.setScene(scene); stage.show();
```

***Клас MainController***

Клас MainController є центральним елементом системи та реалізує основну бізнес-логіку застосунку.

Основні функції:

• керування режимами роботи інтерфейсу;
• відображення теоретичних матеріалів;
• відображення прикладів використання Stream API;
• проведення тестування;
• генерація практичних вправ;
• перевірка відповідей користувача;
• формування результатів навчання;
• обчислення статистики успішності.

У даному класі реалізовано масиви тестових запитань, варіанти відповідей та алгоритми перевірки знань. Приклад тестового запитання:

"Що таке Stream?"

Приклад правильних відповідей: private

final int[] correct =

{1,0,0,1,0,0,0,0,0,1};

***Клас Exercise***

Клас Exercise використовується для представлення практичних завдань.

Основні поля:

• question — текст завдання;

• correctAnswer — правильна відповідь.

Основні методи:

• getQuestion(); • getCorrectAnswer();

• check().

Перевірка відповіді реалізована таким

чином: public boolean check(String

userAnswer) { return

correctAnswer.equalsIgnoreCase(

userAnswer.trim()

);

}

Такий підхід забезпечує просту та надійну перевірку введених користувачем даних.

***Клас Question***

Клас Question використовується для представлення тестових завдань та може містити текст запитання, перелік варіантів відповідей і правильну
відповідь.

Використання окремого класу для тестових запитань дозволяє спростити подальше розширення системи та збільшення кількості тестових
матеріалів.

**Проєктування графічного інтерфейсу**

Інтерфейс користувача реалізовано за допомогою JavaFX.
Основними елементами інтерфейсу є:

- Label;
- TextArea;
- TextField;
- RadioButton;
- ToggleGroup;
- VBox.

Для організації роботи застосунку використовуються три основні режими:

1. Режим теоретичних матеріалів.
2. Режим тестування.
3. Режим практичних вправ.

Перемикання між режимами реалізовано за допомогою методів:

showTheoryMode() showQuizMode()

showPracticeMode()

Дані методи змінюють видимість відповідних елементів інтерфейсу та забезпечують зручну навігацію між функціональними модулями системи.

**Організація тестування знань**

Для контролю знань реалізовано набір із десяти тестових запитань щодо використання Stream API.

Система містить питання про:

• поняття Stream;

• операцію filter();

• операцію map();

• lazy evaluation;

• collect();

• reduce();

• pipeline;

• проміжні та термінальні операції.

Для кожного запитання передбачено чотири варіанти відповіді. Після завершення тестування користувач отримує результат у форматі: Score: X / 10

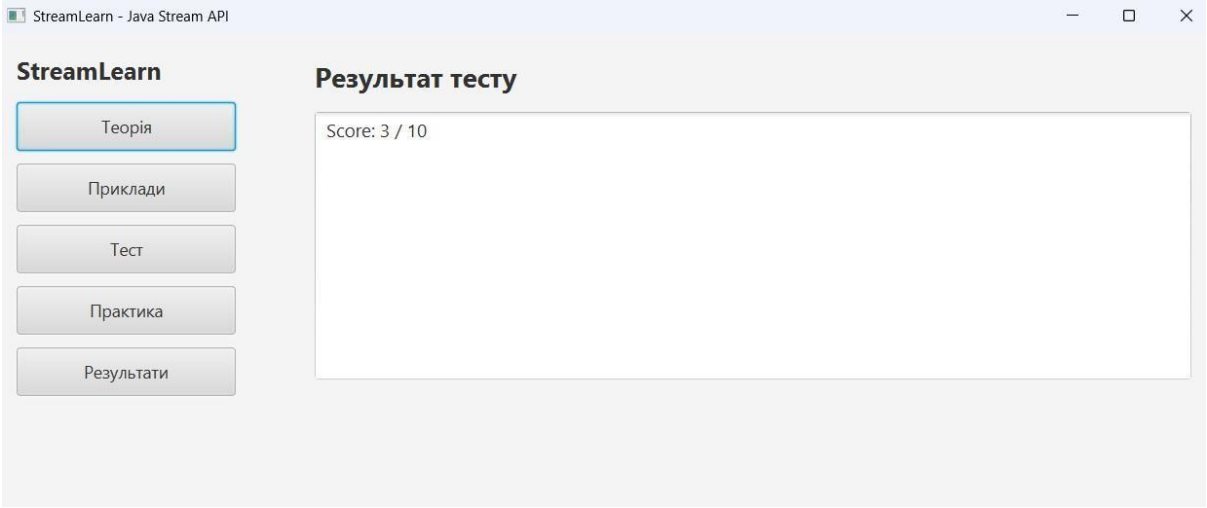

Рисунок 3.1 - приклад відображення результату

**Організація практичних вправ**

Для закріплення знань реалізовано модуль практичних вправ.

Система автоматично формує набір завдань, пов’язаних із використанням Stream API.

Приклади вправ:

• фільтрація чисел;

• використання map();

• використання count();

• побудова ланцюжка filter() → map() → sorted();

• використання collect(Collectors.toList()).

Для окремих вправ використовується випадкове значення, що генерується класом Random.

Фрагмент реалізації:

```
int base = random.nextInt(6) + 5;
```

Застосування випадкових значень підвищує різноманітність завдань та робить процес навчання більш цікавим.

**Організація підрахунку результатів**

Після завершення тестування або практичних вправ система автоматично формує статистику успішності.

Для цього визначаються:

• кількість правильних відповідей;

• кількість виконаних завдань;

• загальний бал;

• відсоток успішності.

Обчислення успішності виконується за формулою: успішність =

(отримані бали / максимальна кількість балів) × 100 %

Отримані результати відображаються користувачу у текстовому вигляді.

**Масштабованість системи**

Запропонована архітектура передбачає можливість подальшого розвитку програмного продукту.

Основні напрями модернізації:

• збільшення кількості теоретичних матеріалів;

• розширення бази тестових запитань;

• додавання нових практичних вправ;

• інтеграція з базою даних;

• збереження статистики користувачів;

• підтримка декількох мов інтерфейсу;

• створення вебверсії застосунку.

Модульна структура програми дозволяє реалізовувати нові функції без суттєвого перепроєктування системи.

**Висновки до підрозділу**

У процесі проєктування було сформовано архітектуру навчального застосунку StreamLearn та визначено структуру його основних компонентів. Використання JavaFX забезпечило створення сучасного графічного інтерфейсу користувача, а модульна організація програмного коду спростила реалізацію функціональних можливостей системи. Спроєктована структура повністю відповідає поставленим вимогам та забезпечує ефективну підтримку процесу вивчення Stream API.

# 4 ПРАКТИЧНА ЧАСТИНА

## 4.1 Програмна реалізація застосунку

Під час реалізації програмного продукту було використано мову програмування Java, бібліотеку JavaFX для створення графічного інтерфейсу користувача та систему керування залежностями Maven.

Основною метою розробки було створення інтерактивного навчального середовища для вивчення технології Stream API, що поєднує теоретичний матеріал, демонстраційні приклади, тестування та практичні вправи.

### Реалізація запуску застосунку

Запуск програмного продукту здійснюється за допомогою класу StreamLearnApp, який успадковує клас Application із бібліотеки JavaFX.

Основним завданням даного класу є ініціалізація графічного інтерфейсу та створення головного вікна програми.

Фрагмент програмного коду:

```
public class StreamLearnApp extends Application {

   @Override       public void start(Stage stage)
throws Exception {

       FXMLLoader loader = new FXMLLoader(
getClass().getResource("/main.fxml")
       );

       Scene scene = new Scene(loader.load(), 1000, 700);
```

```
        scene.getStylesheets().add(
                getClass().getResource("/style.css")
                        .toExternalForm()
        );

        stage.setTitle(
                "StreamLearn - Java Stream API"
        );

        stage.setScene(scene);
stage.show();
    }

    public static void main(String[] args) {
launch();
    }
}
```

Після запуску користувач отримує доступ до всіх функціональних можливостей системи через графічний інтерфейс.

**Реалізація графічного інтерфейсу**

Для створення інтерфейсу використано технологію JavaFX. Основними елементами інтерфейсу є:

• Label;

• TextArea;

• TextField;

• RadioButton;

• VBox;

• ToggleGroup.

Керування інтерфейсом здійснюється через клас MainController. Для перемикання між режимами роботи реалізовано три окремі методи:

showTheoryMode() showQuizMode() showPracticeMode()

Використання окремих режимів дозволяє приховувати або відображати необхідні елементи інтерфейсу залежно від обраного функціонального модуля.

Приклад реалізації:

```
private void showTheoryMode() {
quizBox.setVisible(false);     practiceBox.setVisible(false);
contentArea.setVisible(true);
}
```

Такий підхід забезпечує просту навігацію між різними розділами застосунку.

**Реалізація модуля теоретичних матеріалів**

Для вивчення основ Stream API у системі реалізовано окремий теоретичний модуль.

Відображення навчального матеріалу здійснюється методом showTheory().

У даному модулі користувач може ознайомитися з такими поняттями:

• Stream API;
• pipeline;
• lazy evaluation;
• проміжні операції;
• термінальні операції;
• особливості функціонального програмування.

Матеріал відображається у компоненті TextArea та містить структурований опис основних можливостей Stream API.

**Реалізація модуля демонстраційних прикладів**

Для формування практичних навичок роботи з потоками даних реалізовано модуль прикладів.

Модуль демонструє використання найбільш поширених операцій Stream API.

Приклад використання filter():

```
numbers.stream()
      .filter(n -> n > 3)
      .forEach(System.out::println);
```

Приклад використання map(): names.stream()

```
    .map(String::toUpperCase)
    .forEach(System.out::println);
```

Приклад використання sorted():

```
nums.stream()
   .sorted()
   .forEach(System.out::println);
```

Приклад використання count():

```
long count = numbers.stream()
                .filter(n -> n > 2)
                .count();
```

Приклад використання reduce(): int

```
sum = numbers.stream()
            .reduce(0, Integer::sum);
```

Приклад використання collect():

```
List<Integer> result =
numbers.stream()
            .filter(n -> n % 2 == 0)
            .collect(Collectors.toList());
```

Таким чином користувач отримує можливість ознайомитися з практичним застосуванням потокової обробки даних.

**Реалізація модуля тестування**

Для контролю рівня знань реалізовано систему тестування.

У програмі використовується масив тестових запитань:

```
private final String[] questions
```

Кожне запитання має чотири варіанти відповіді.

Приклади тестових запитань:

• Що таке Stream?

• Що робить filter()?

• Що повертає map()?

• Що таке lazy evaluation?

• Що робить collect()?

• Що таке pipeline?

• Для чого використовується reduce()?

Завантаження поточного запитання здійснюється методом loadQuestion().

Перевірка правильності відповіді виконується методом checkAnswer().

Фрагмент програмного коду:

```
if (selectedIndex == correct[index]) {
score++;
}
```

Після проходження всіх десяти запитань користувачу відображається підсумковий результат.

Приклад результату:

Score: 8 / 10

**Реалізація модуля практичних вправ**

Для закріплення знань користувач може виконувати практичні завдання.

Практичні вправи формуються динамічно під час запуску відповідного режиму роботи.

Створення набору вправ здійснюється методом showExercise(). Для урізноманітнення завдань використовується генератор випадкових чисел:

```
int base = random.nextInt(6) + 5;
```

Приклади вправ:

• Відфільтруй числа більше заданого значення;

• Помнож кожен елемент на 2;

• Порахуй елементи більше заданого значення;

• Перетвори елементи у квадрат;

• Збери результат у список;

• Побудуй ланцюжок filter() → map() → sorted().

Для зберігання інформації про вправу використовується клас Exercise.

Структура класу:

```
private final String question; private
final String correctAnswer;
```

Перевірка відповіді виконується методом:

```
public boolean check(String userAnswer) {
return correctAnswer.equalsIgnoreCase(
userAnswer.trim()
    );
```

Якщо відповідь правильна, збільшується значення змінної practiceScore.

Після завершення всіх вправ користувачу відображається загальний результат.

**Реалізація модуля результатів навчання**

Для аналізу успішності користувача реалізовано окремий модуль результатів.

Відображення статистики здійснюється методом showResults().

Система формує такі показники:

• результат останнього тестування;

• результат останньої практики;

• загальний бал;

• відсоток успішності.

Обчислення успішності виконується за формулою:

успішність = (отримані бали / максимальні бали) × 100 %

Отримані результати відображаються у зручному текстовому вигляді.

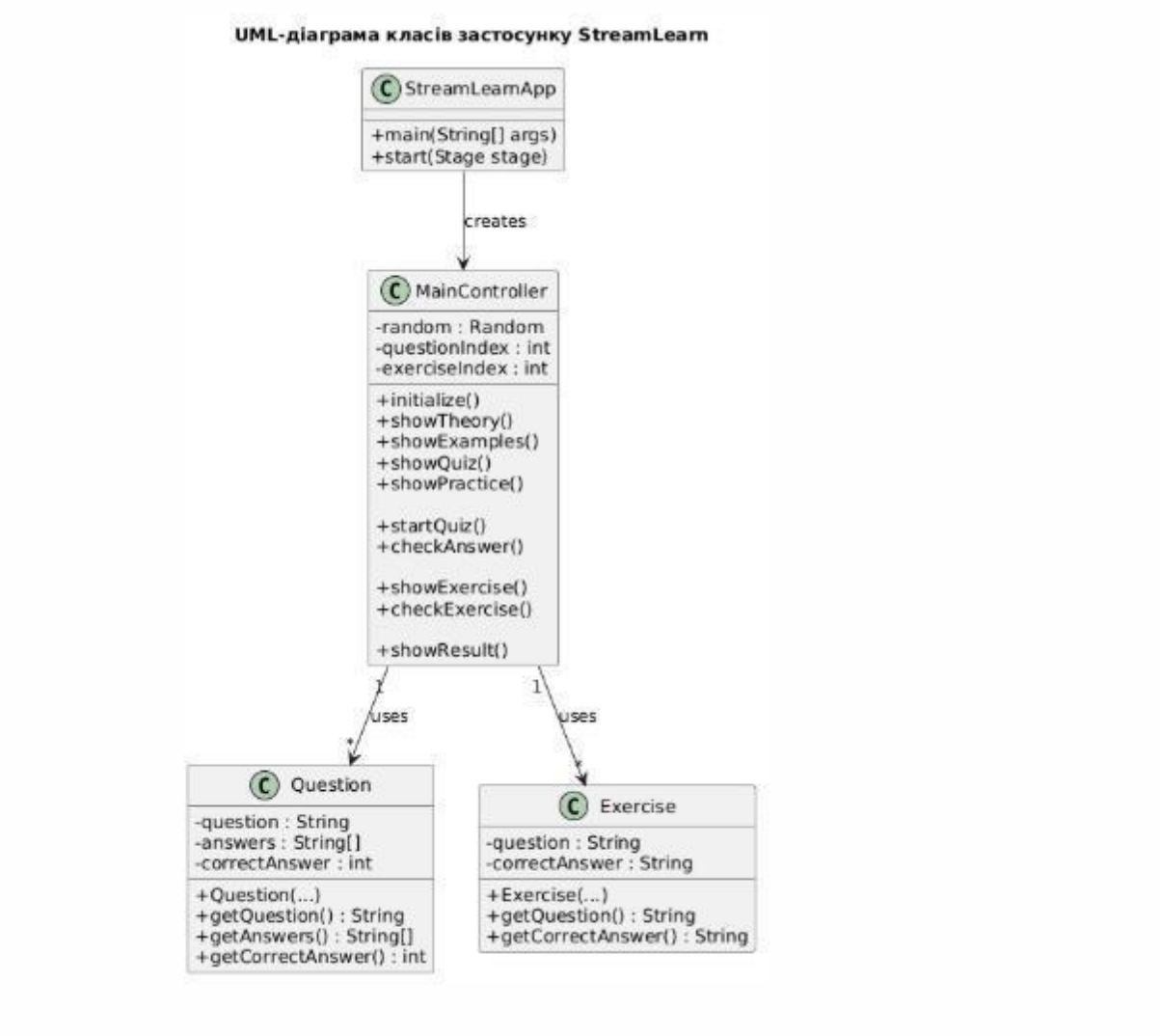


Рисунок 4.1 - UML-діаграма класів застосунку StreamLearn

**Висновки до підрозділу**

У процесі програмної реалізації було створено повноцінний навчальний застосунок StreamLearn, який забезпечує вивчення

теоретичних основ Stream API, виконання демонстраційних прикладів, проходження тестування та виконання практичних вправ. Використання JavaFX дозволило реалізувати сучасний графічний інтерфейс користувача, а модульна структура програмного коду забезпечила простоту супроводу та можливість подальшого розвитку програмного продукту.

## 4.2 Тестування програмного продукту

Після завершення розробки програмного продукту було проведено функціональне тестування всіх основних модулів системи. Метою тестування була перевірка коректності роботи програмного забезпечення, відповідності реалізованого функціоналу поставленим вимогам та виявлення можливих помилок під час взаємодії користувача із системою.

Тестування виконувалося шляхом послідовної перевірки всіх функціональних можливостей застосунку StreamLearn.

### Тестування запуску програми

На першому етапі було перевірено коректність запуску застосунку.

Послідовність тестування:

1. Запустити програму.
2. Перевірити завантаження головного вікна.
3. Перевірити відображення всіх елементів інтерфейсу.
4. Перевірити коректність підключення стилів оформлення.

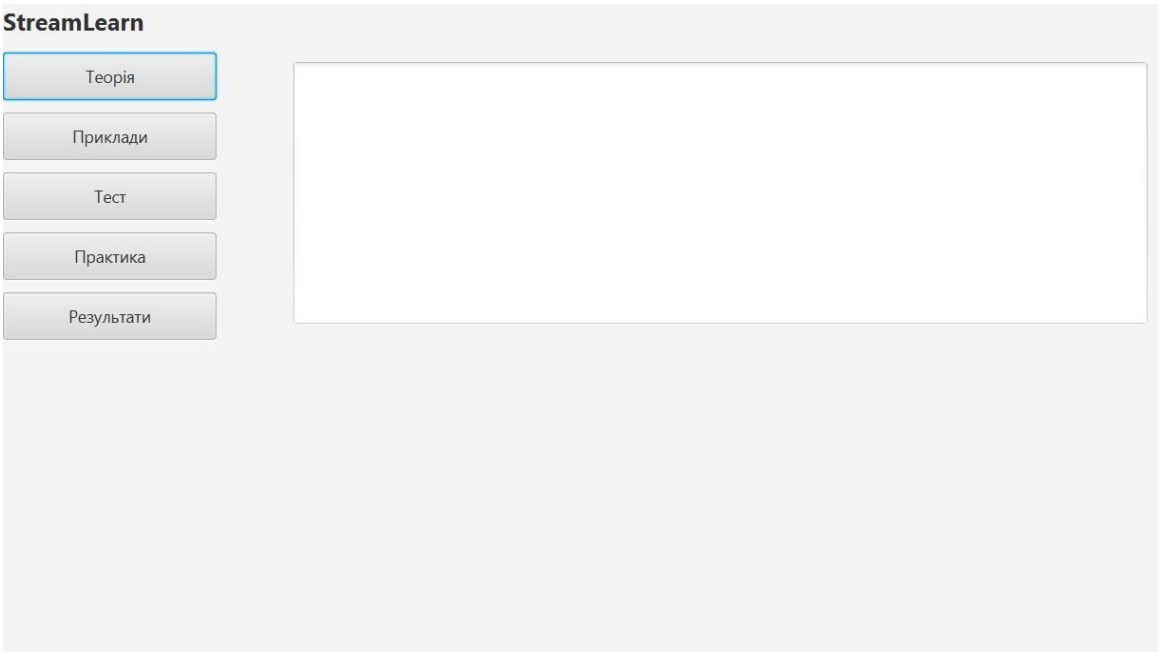


Рисунок 4.2 - Приклад відображення головного вікна

Результат тестування наведено у таблиці 4.1.

Таблиця 4.1 – Тестування запуску програми

| **Дія** | **Очікуваний результат** | **Результат** |
|---|---|---|
| Запуск застосунку | Відображення головного вікна | Успішно |
| Завантаження FXML | Коректне формування інтерфейсу | Успішно |
| Підключення стилів | Коректне оформлення елементів | Успішно |

Під час тестування помилок не виявлено.

**Тестування модуля теоретичних матеріалів**

Для перевірки роботи модуля теорії було виконано відкриття відповідного розділу.

Послідовність тестування:

1. Натиснути кнопку переходу до теоретичного матеріалу.
2. Перевірити відображення тексту.
3. Перевірити коректність форматування інформації.
4. Перевірити можливість переходу до інших модулів.

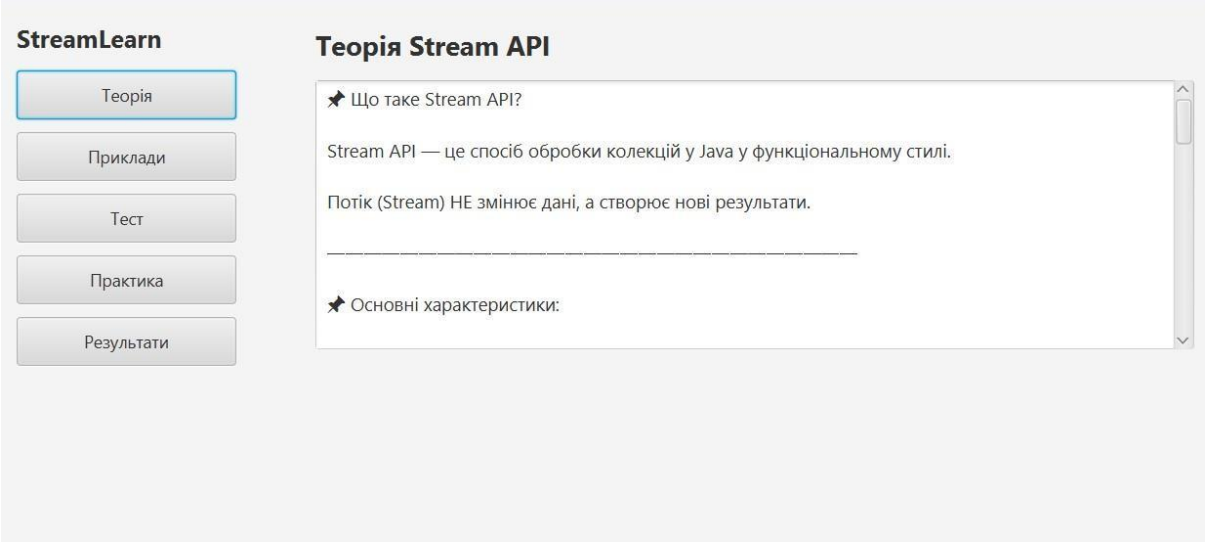


Рисунок 4.3 - приклад відображення тексту теоритичного матеріалу

Результати наведено у таблиці 4.2.

Таблиця 4.2 – Тестування теоретичного модуля

| **Дія** | **Очікуваний результат** | **Результат** |
|---|---|---|
| Відкриття теорії | Відображення матеріалу | Успішно |
| Відображення змісту | Коректне відображення тексту | Успішно |

| | | |
|---|---|---|
| Перехід між розділами | Відсутність помилок | Успішно |

Модуль працює відповідно до поставлених вимог.

**Тестування модуля прикладів Stream API**

Наступним етапом було перевірено модуль демонстраційних прикладів.

Перевірялися приклади:

• filter();

• map();

• sorted();

• count();

• reduce(); • collect().

Таблиця 4.3 – Тестування модуля прикладів

| **Дія** | **Очікуваний результат** | **Результат** |
|---|---|---|
| Відкриття прикладів | Відображення прикладів коду | Успішно |
| Перегляд прикладів | Коректне відображення тексту | Успішно |
| Перехід до інших розділів | Відсутність помилок | Успішно |

Під час тестування порушень роботи не виявлено.

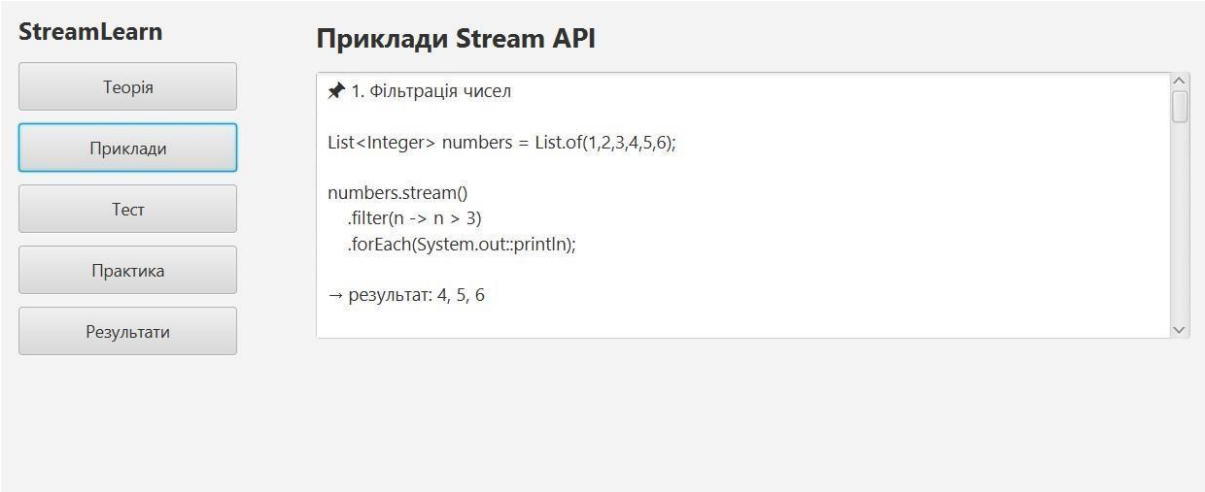


Рисунок 4.4 - Приклад відображення модуля “приклади”

**Тестування модуля контролю знань**

Для перевірки тестового режиму було виконано повне проходження тестування.

Перевірялися:

• завантаження запитань;

• відображення варіантів відповідей;

• коректність підрахунку балів;

• відображення результатів.

Таблиця 4.4 – Тестування модуля тестування

| **Дія** | **Очікуваний результат** | **Результат** |
|---|---|---|
| Початок тесту | Завантаження першого питання | Успішно |
| Вибір відповіді | Перехід до наступного питання | Успішно |

| | | |
|---|---|---|
| Завершення тесту | Відображення результату | Успішно |
| Підрахунок балів | Коректний результат | Успішно |

Під час тестування було підтверджено правильність роботи алгоритму підрахунку балів.

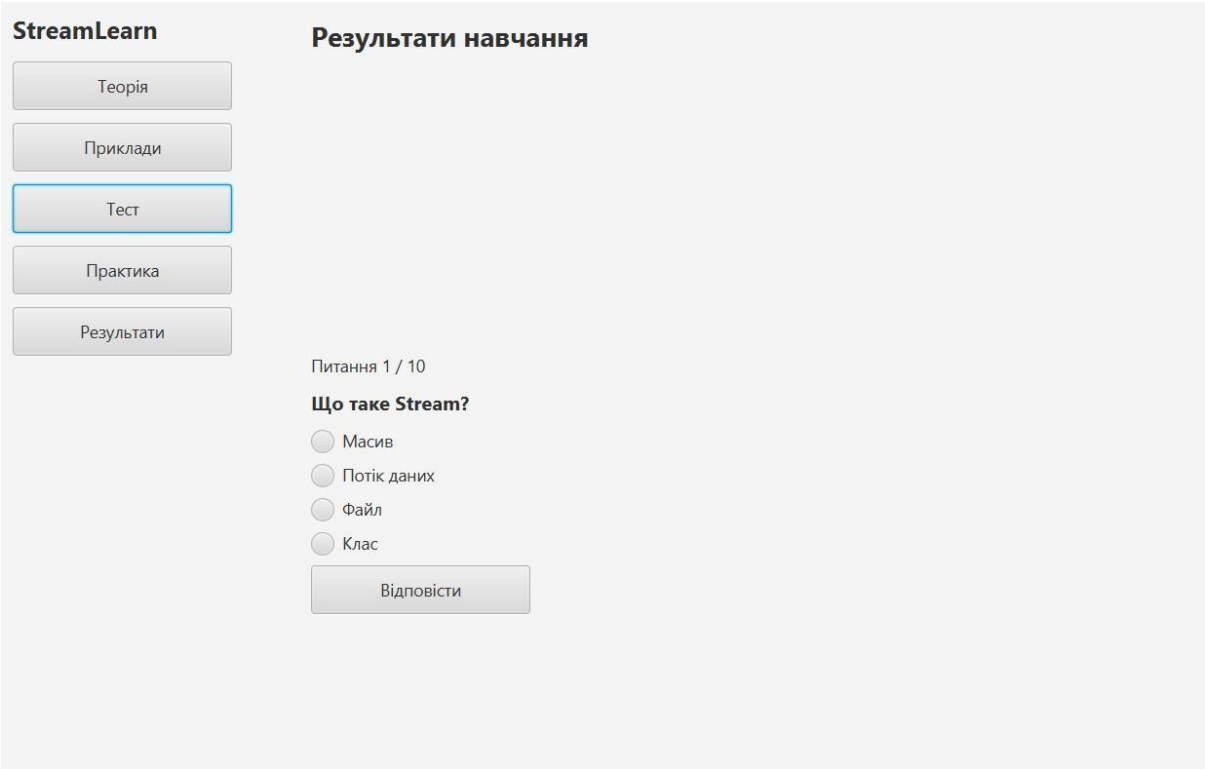


Рисунок 4.5 - Приклад відображення модуля тестування Приклад отриманого результату:

Score: 8 / 10

Також було перевірено ситуацію, коли користувач не обирає жодного варіанта відповіді.

У цьому випадку система не виконує перехід до наступного запитання, що запобігає виникненню помилок під час тестування.

**Тестування модуля практичних вправ**

Для перевірки роботи практичного модуля було виконано декілька циклів проходження вправ.

Перевірялися:

• генерація завдань;

• відображення тексту завдань;

• перевірка введених відповідей;

• формування результатів.

Таблиця 4.5 – Тестування практичних вправ

| **Дія** | **Очікуваний результат** | **Результат** |
|---|---|---|
| Створення вправ | Генерація набору завдань | Успішно |
| Введення правильної відповіді | Зарахування балу | Успішно |
| Введення неправильної відповіді | Виведення повідомлення про помилку | Успішно |
| Завершення вправ | Формування результату | Успішно |

Було перевірено правильність роботи методу перевірки відповідей та коректність оновлення значення змінної practiceScore.

**Тестування модуля результатів навчання**

Для перевірки роботи модуля результатів було проведено декілька тестових сценаріїв.

Перевірялися:

- відображення результатів тестування; • відображення результатів практики;• підрахунок загального балу;
- визначення відсотка успішності.

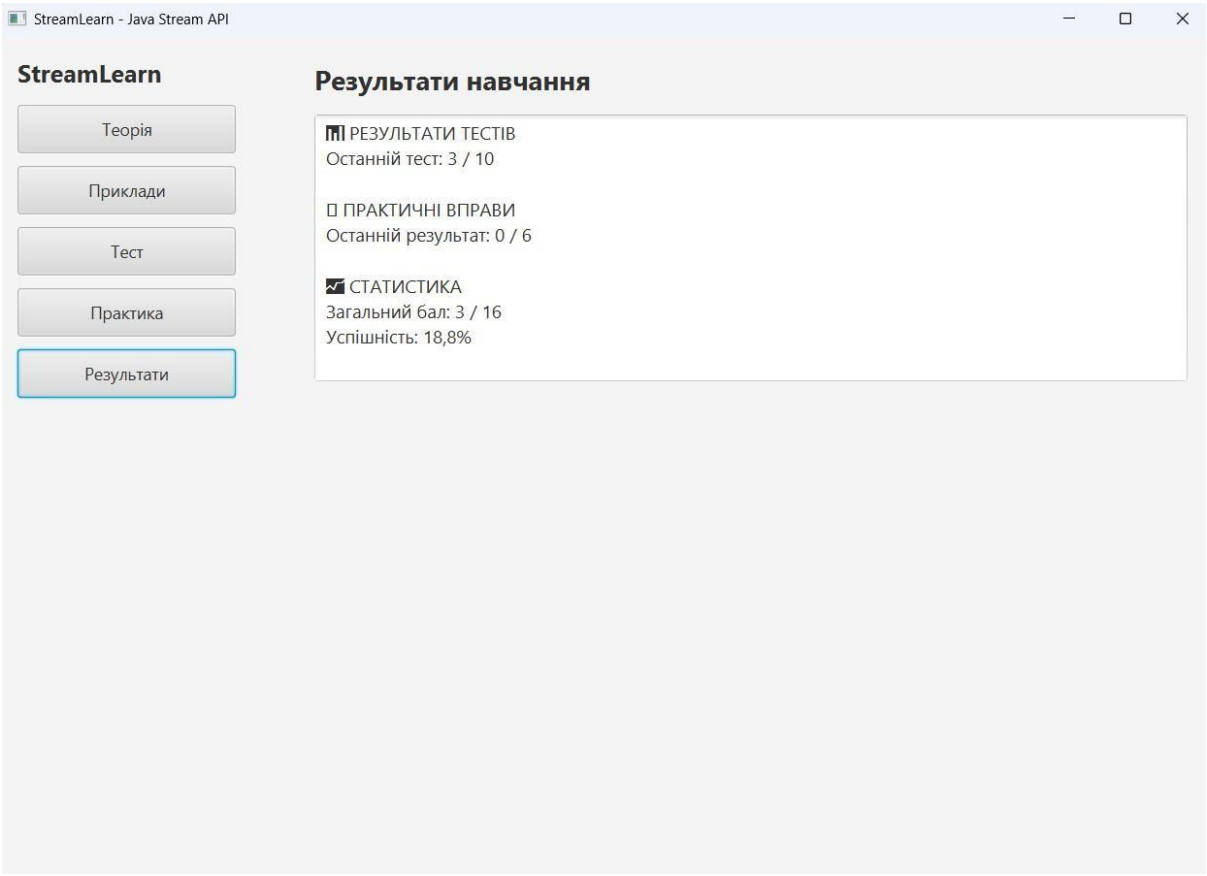


Рисунок 4.6 - Приклад відображення результатів

Таблиця 4.6 – Тестування модуля результатів

| **Дія** | **Очікуваний результат** | **Результат** |
|---|---|---|
| Відображення статистики | Коректний звіт | Успішно |

| | | |
|---|---|---|
| Підрахунок балів | Правильний результат | Успішно |
| Обчислення успішності | Коректне значення | Успішно |

У процесі тестування було підтверджено правильність обчислення підсумкових результатів.

**Аналіз результатів тестування**

Результати проведеного тестування підтвердили коректність роботи всіх основних модулів системи.

Перевірка показала:

• стабільну роботу графічного інтерфейсу;

• коректне відображення теоретичних матеріалів;

• правильне функціонування тестового режиму;

• правильну роботу практичних вправ;

• коректне формування результатів навчання.

Критичних помилок, які могли б призвести до порушення роботи програмного забезпечення, під час тестування не виявлено.

**Висновки до підрозділу**

У результаті проведеного функціонального тестування було підтверджено працездатність програмного продукту StreamLearn та коректність реалізації його основних функцій. Усі перевірені модулі успішно пройшли тестування та продемонстрували відповідність поставленим вимогам. Отримані результати свідчать про готовність програмного продукту до практичного використання у навчальному процесі.

## 4.3 Інструкція користувача

Розроблений програмний продукт StreamLearn призначений для вивчення технології Stream API в мові програмування Java. Застосунок має графічний інтерфейс користувача, реалізований засобами JavaFX, що забезпечує простоту роботи та зручність використання.

### Призначення програмного продукту

Програмний продукт призначений для:

• вивчення теоретичних основ Stream API;

• ознайомлення з прикладами використання потоків даних;

• перевірки знань за допомогою тестування;

• виконання практичних вправ;

• аналізу результатів навчання.

Застосунок може використовуватися студентами, викладачами та всіма користувачами, які вивчають функціональне програмування в Java.

### Вимоги до програмного забезпечення

Для роботи застосунку необхідно:

• операційна система Windows 10 або новіша;

• Java Development Kit (JDK) версії 17 або вище;

• JavaFX;

• не менше 4 ГБ оперативної пам’яті;

• близько 100 МБ вільного місця на диску.

### Запуск програми

Для запуску застосунку необхідно:

1. Відкрити проєкт у середовищі IntelliJ IDEA.

2. Дочекатися завершення завантаження Maven-залежностей.
3. Запустити клас StreamLearnApp.
4. Дочекатися відкриття головного вікна програми.

Після запуску користувач отримує доступ до всіх функціональних можливостей системи.

**Головне вікно програми**

Головне вікно застосунку містить елементи навігації та область відображення інформації.

Основні розділи програми:

• Теорія;

• Приклади Stream API;

• Тестування знань;

• Практичні вправи;

• Результати навчання.

Перехід між розділами здійснюється за допомогою відповідних кнопок інтерфейсу.

**Робота з теоретичними матеріалами**

Для ознайомлення з теоретичним матеріалом необхідно:

1. Відкрити розділ «Теорія».
2. Ознайомитися з навчальними матеріалами.
3. Переглянути опис основних понять Stream API.
4. За необхідності перейти до прикладів або тестування.

У теоретичному модулі розглядаються:

• поняття Stream API;

• lazy evaluation;

• pipeline;

• проміжні операції;

• термінальні операції;

• особливості потокової обробки даних.

**Робота з демонстраційними прикладами**

Для перегляду прикладів використання Stream API необхідно:

1. Відкрити розділ «Приклади».
2. Ознайомитися з наведеними фрагментами програмного коду.
3. Проаналізувати результати виконання операцій.
4. Порівняти різні способи використання Stream API.

У програмі наведено приклади використання:

• filter();

• map();

• sorted();

• count();

• reduce(); • collect().

Приклад:

```
numbers.stream()
    .filter(n -> n > 3)
    .forEach(System.out::println);
```

Результат виконання:

4

5

6

**Проходження тестування**

Для перевірки рівня знань необхідно:

1. Відкрити розділ «Тестування».
2. Ознайомитися з поточним запитанням.
3. Вибрати один із запропонованих варіантів відповіді.
4. Перейти до наступного запитання.
5. Повторити дії до завершення тесту.

Тест містить десять запитань щодо використання Stream API.

Приклади запитань:

• Що таке Stream?

• Що робить filter()?

• Що повертає map()?

• Що таке lazy evaluation?

• Що робить collect()?

Після завершення тестування система автоматично відображає результат у форматі:

Score: X / 10

де X - кількість правильних відповідей.

**Виконання практичних вправ**

Для закріплення знань користувач може виконувати практичні вправи.

Порядок роботи:

1. Відкрити розділ «Практичні вправи».
2. Ознайомитися з умовою завдання.
3. Ввести відповідь у текстове поле.
4. Натиснути кнопку перевірки.

5. Переглянути результат перевірки.

У програмі реалізовано вправи на використання:

• filter();

• map();

• count();

• sorted();

• collect().

Приклади завдань:

• Відфільтруй числа більше заданого значення.

• Помнож кожен елемент на 2.

• Порахуй кількість елементів.

• Перетвори елементи у квадрат.

• Збери результат у список.

Після перевірки система повідомляє про правильність введеної відповіді та автоматично переходить до наступного завдання.

Після завершення всіх вправ відображається підсумковий результат.

**Перегляд результатів навчання** Для

перегляду статистики необхідно:

1. Відкрити розділ «Результати навчання».
2. Ознайомитися з отриманими показниками.

У даному розділі відображаються:

• результат останнього тестування;

• результат останньої практики;

• загальний бал;

• відсоток успішності.

На основі цих даних користувач може оцінити рівень засвоєння навчального матеріалу.

**Типові помилки користувача**

Під час роботи із системою можуть виникати окремі ситуації, пов’язані з неправильними діями користувача.

Таблиця 4.7 – Типові помилки

| **Ситуація** | **Причина** | **Спосіб усунення** |
|---|---|---|
| Не обрано відповідь у тесті | Користувач не вибрав варіант відповіді | Обрати відповідь та продовжити тест |
| Неправильна відповідь у вправі | Помилка під час введення відповіді | Перевірити синтаксис Stream API |
| Нульовий результат тесту | Неправильні відповіді на всі запитання | Повторити вивчення теоретичного матеріалу |

У більшості випадків система коректно обробляє подібні ситуації та продовжує роботу без збоїв.

**Рекомендації щодо використання**

Для досягнення максимального результату рекомендується:

• спочатку вивчити теоретичний матеріал;

• ознайомитися з прикладами використання Stream API;

• пройти тестування;

• виконати практичні вправи;

• проаналізувати отримані результати;

• повторити теми, які викликали труднощі.

Такий підхід дозволяє сформувати стійкі знання щодо використання Stream API та навички практичного застосування потокової обробки даних.

**Висновки до підрозділу**

Розроблений програмний продукт має зрозумілий графічний інтерфейс та простий порядок використання. Інструкція користувача охоплює всі основні можливості системи та описує порядок роботи з кожним функціональним модулем. Використання застосунку дозволяє ефективно організувати процес вивчення Stream API та контролювати результати навчання.

# ВИСНОВОК

У результаті виконання кваліфікаційної роботи було розроблено навчальний програмний продукт для вивчення технології Stream API в мові програмування Java. Створений застосунок призначений для підтримки навчального процесу та забезпечує комплексний підхід до засвоєння теоретичних знань і формування практичних навичок використання потокової обробки даних.

Під час виконання роботи було проведено аналіз сучасних підходів до створення навчального програмного забезпечення, досліджено особливості функціонального програмування в Java та розглянуто можливості використання Stream API для реалізації сучасних програмних рішень. Проведений аналіз підтвердив актуальність використання інтерактивних програмних засобів для підвищення ефективності навчання програмуванню.

У ході виконання кваліфікаційної роботи було досягнуто поставлену мету та вирішено основні завдання дослідження. Зокрема, було визначено функціональні вимоги до системи, спроєктовано архітектуру програмного продукту та реалізовано його основні модулі.

У розробленому застосунку реалізовано:

- модуль теоретичних матеріалів;
- модуль демонстраційних прикладів використання Stream API;
- модуль тестування знань користувача;
- модуль практичних вправ;
- модуль оцінювання результатів навчання;

Розроблений програмний продукт забезпечує можливість послідовного вивчення тематики Stream API, виконання практичних завдань та проходження контролю знань в межах єдиного програмного середовища. Завдяки використанню графічного інтерфейсу

користувача та інтерактивних елементів навчання система є зручною у використанні та придатною для самостійного опрацювання матеріалу.

Під час реалізації програмного продукту було використано сучасні технології розробки програмного забезпечення, зокрема мову програмування Java, бібліотеку JavaFX та систему керування залежностями Maven. Застосування цих технологій дозволило створити масштабований та придатний до подальшого розвитку програмний продукт.

Проведене тестування підтвердило коректність роботи реалізованих функцій, стабільність програмного забезпечення та відповідність результатів поставленим вимогам. Усі основні модулі системи успішно пройшли функціональну перевірку та продемонстрували належний рівень надійності.

Практична цінність роботи полягає у можливості використання розробленого програмного продукту під час навчання студентів основам функціонального програмування та технології Stream API. Застосунок може використовуватися як допоміжний засіб у закладах вищої освіти, під час самостійного навчання або підготовки до практичних занять з дисциплін, пов’язаних із програмуванням мовою Java.

Перспективними напрямами подальшого розвитку програмного продукту є розширення кількості навчальних тем, збільшення бази тестових завдань, реалізація автоматичної генерації практичних вправ, інтеграція з базами даних та створення вебверсії системи для дистанційного навчання.

Отже, поставлена мета кваліфікаційної роботи досягнута, усі визначені завдання виконані, а розроблений навчальний програмний продукт може бути ефективно використаний для вивчення технології Stream API в мові програмування Java.

## СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ


1. Блох Дж. Effective Java. 3rd Edition. – Boston : Addison-Wesley Professional, 2018. – 416 p.

URL: https://kea.nu/files/textbooks/new/Effective%20Java%20%282017%2C%20Add ison-Wesley%29.pdf

2. Шилдт Г. Java: The Complete Reference. 12th Edition. – New York : McGraw-Hill Education, 2021. – 1248 p.

URL: https://www.accessengineeringlibrary.com/content/book/9781260463415

3. Oracle Corporation. Java Platform Standard Edition Documentation.

URL: https://docs.oracle.com/en/java/

4. Oracle Corporation. Stream API Package Summary.

URL: https://docs.oracle.com/javase/8/docs/api/java/util/stream/package

5. Oracle Corporation. Java Tutorials.

URL: https://docs.oracle.com/javase/tutorial/

6. Sierra K., Bates B. Head First Java. 3rd Edition. – Sebastopol : O'Reilly Media, 2022. – 688 p.

URL: https://www.pdfiles.net/pdf/view/Head-First-Java,-3nd-Edition

7. Urma R., Fusco M., Mycroft A. Java 8 in Action. – Shelter Island : Manning Publications, 2015. – 424 p.

URL:
https://vcfvct.wordpress.com/wpcontent/uploads/2016/06/java8inaction.pdf

8. Urma R.-G. Modern Java in Action. – Shelter Island : Manning Publications, 2019. – 592 p.

URL: https://manning-content.s3.amazonaws.com/download/8/1bf1f0b-c08d-
4884-a873-4a5717e9620d/sample_ch01_Urma_Modern-JavaiA_September20.pdf

9. Evans E. Domain-Driven Design: Tackling Complexity in the Heart of Software. – Addison-Wesley, 2003. – 560 p.

URL: https://fabiofumarola.github.io/nosql/readingMaterial/Evans03.pdf

10. Gamma E., Helm R., Johnson R., Vlissides J. Design Patterns: Elements of Reusable Object-Oriented Software. – Addison-Wesley, 1994. – 395 p.

URL: http://www.uml.org.cn/c++/pdf/DesignPatterns.pdf

11. Fowler M. UML Distilled: A Brief Guide to the Standard Object Modeling Language. 4th Edition. – Addison-Wesley, 2018. – 208 p.

URL:
https://ptgmedia.pearsoncmg.com/images/9780321193681/samplepages/978032 1193681_Sample.pdf

12. Bell D. UML Basics. – 2003 p.

URL: https://ru.scribd.com/document/43843983/UML-Basics-an-Introduction

13. OpenJFX Documentation.

URL: https://openjfx.io/

14. OpenJFX Documentation. Getting Started with JavaFX.

URL: https://openjfx.io/openjfx-docs/

15. Apache Software Foundation. Maven Project Documentation.

URL: https://maven.apache.org/

16. Goetz B. Java Concurrency in Practice. – Addison-Wesley, 2006. – 424 p.

URL:
https://ptgmedia.pearsoncmg.com/images/9780321349606/samplepages/9780321349606.pdf

17. Eckel B. Thinking in Java. 4th Edition. – Prentice Hall, 2006. – 1150 p.

18. Дудник Д. А., Парфьонова Т. О.

Development of the training simulator's software on the topic of "Addition and multiplication theorems of probability" for the distance training course "Probability theory and mathematical statistics"

URL: http://dspace.puet.edu.ua/handle/123456789/10416

## ДОДАТОК А

Код Програми

StreamLearnApp.Java

```
package com.streamlearn;

import
javafx.application.Application;
import    javafx.fxml.FXMLLoader;
import javafx.scene.Scene;
import javafx.stage.Stage;

public class StreamLearnApp extends Application {

   @Override
   public void start(Stage stage) throws Exception {

       FXMLLoader loader = new FXMLLoader(
getClass().getResource("/main.fxml")
       );
       Scene scene = new Scene(loader.load(), 1000, 700);

scene.getStylesheets().add(
getClass().getResource("/style.css").toExternalForm()
       );
         stage.setTitle("StreamLearn - Java Stream
API");
stage.setScene(scene);         stage.show();
   }         public static void
main(String[] args) {
launch();
   }
}
```

Question.java

```
package com.streamlearn.model;

public class Question {

   private final String question;
private final String[] answers;
private final int correctAnswer;
```

```
    public Question(String question,
                    String[] answers,
int correctAnswer) {

        this.question = question;
this.answers = answers;
this.correctAnswer = correctAnswer;
    }        public String
getQuestion() {
return question;
    }        public String[]
getAnswers() {         return
answers;
    }        public int
getCorrectAnswer() {
return correctAnswer;
    }
}
```

pom.xml

```
<project xmlns="http://maven.apache.org/POM/4.0.0">

    <modelVersion>4.0.0</modelVersion>

    <groupId>com.streamlearn</groupId>
    <artifactId>StreamLearn</artifactId>
    <version>1.0</version>

    <properties>
        <maven.compiler.source>17</maven.compiler.source>
        <maven.compiler.target>17</maven.compiler.target>
        <javafx.version>21</javafx.version>
    </properties>

    <dependencies>
```

```
        <dependency>
            <groupId>org.openjfx</groupId>
            <artifactId>javafx-controls</artifactId>
            <version>${javafx.version}</version>
        </dependency>

        <dependency>
            <groupId>org.openjfx</groupId>
            <artifactId>javafx-fxml</artifactId>
            <version>${javafx.version}</version>
        </dependency>

    </dependencies>

</project> main.fxml
<?xml version="1.0" encoding="UTF-8"?>

<?import javafx.scene.control.*?>
<?import javafx.scene.layout.*?>

<BorderPane xmlns:fx="http://javafx.com/fxml"
fx:controller="com.streamlearn.MainController">

    <left>
        <VBox spacing="10"              prefWidth="240"
style="-fx-padding:15; -fx-background-color:#f4f4f4;">

            <Label text="StreamLearn"
                   style="-fx-font-size:20px; -fx-font-
weight:bold;"/>
            <Button text="Теорія" onAction="#showTheory"/>
            <Button text="Приклади" onAction="#showExamples"/>
            <Button text="Тест" onAction="#showQuiz"/>
            <Button text="Практика" onAction="#showExercise"/>
            <Button text="Результати" onAction="#showResults"/>
```

```
    </VBox>
  </left>

  <center>
    <VBox spacing="12" style="-fx-padding:20;">

      <Label fx:id="titleLabel"                    style="-
fx-font-size:22px; -fx-font-weight:bold;"/>
      <TextArea        fx:id="contentArea"
wrapText="true"
prefHeight="220"/>
      <!-- QUIZ -->
      <VBox fx:id="quizBox" spacing="8">

        <Label fx:id="progressLabel"/>

        <Label                        fx:id="questionLabel"
style="-fx-font-size:16px; -fx-font-weight:bold;"/>
        <RadioButton fx:id="option1"/>
        <RadioButton fx:id="option2"/>
        <RadioButton fx:id="option3"/>
        <RadioButton fx:id="option4"/>

        <Button text="Відповісти"
onAction="#checkAnswer"/>
      </VBox>

      <!-- PRACTICE -->
      <VBox fx:id="practiceBox" spacing="8">

        <Label fx:id="exerciseLabel"                  style="-fx-
font-size:15px;"/>
```

```
            <TextField fx:id="answerField"
promptText="Введіть код"/>

            <Button text="Перевірити"
onAction="#checkExercise"/>

        </VBox>

    </VBox>
  </center>

</BorderPane>
```
Style.css
```
.root {
  -fx-font-size: 14px;
}

.button {
  -fx-pref-width: 180px;
  -fx-pref-height:
40px;  }

.text-area {
  -fx-font-size:
15px;  }
```
Exersice.java
```
package
com.streamlearn.
model;

public class Exercise {
    private final String
question;    private final
String correctAnswer;

  public Exercise(String question, String correctAnswer) {
this.question = question;       this.correctAnswer =
correctAnswer;
```

```
    }         public String
getQuestion() {
return question;
    }
      public String
getCorrectAnswer() {
return correctAnswer;
    }


    public boolean check(String userAnswer) {          return
correctAnswer.equalsIgnoreCase(userAnswer.trim());
    }
}
```

MainController.java package

```
com.streamlearn; import
java.util.ArrayList; import
java.util.List; import
java.util.Random; import
com.streamlearn.model.Exercise;
import javafx.fxml.FXML; import
javafx.scene.control.*;  import
javafx.scene.layout.VBox;
import java.util.List;


public class MainController {


    // ===== UI =====
    @FXML private Label titleLabel;
    @FXML private TextArea contentArea;

    @FXML private VBox quizBox;
    @FXML private VBox practiceBox;


    @FXML private Label questionLabel;
    @FXML private Label progressLabel;


    @FXML private RadioButton option1;
```

```
    @FXML private RadioButton option2;
    @FXML private RadioButton option3;
    @FXML private RadioButton option4;

    @FXML private Label exerciseLabel;
@FXML private TextField answerField;
private final Random random = new
Random();     private
ToggleGroup group;

    // ===== QUIZ DATA =====
    private final String[] questions = {
"Що таке Stream?",
            "Що робить filter()?",
            "Що повертає map()?",
            "Що таке lazy evaluation?",
            "Що робить collect()?",
            "Що таке pipeline?",
            "Яка terminal операція?",
            "Що таке intermediate?",
            "reduce використовується для?",
            "Stream змінює колекцію?"
    };

    private final String[][] answers = {
            {"Масив", "Потік даних", "Файл", "Клас"},
            {"Фільтрує", "Сортує", "Копіює", "Видаляє"},
            {"Stream", "int", "void", "char"},
            {"Швидке", "Відкладене виконання", "Помилка",
"Цикл"},
            {"Збирає результат", "Створює", "Зупиняє", "Фільтрує"},
            {"Ланцюг операцій", "Цикл", "Файл", "Клас"},
            {"collect", "map", "filter", "peek"},
            {"Проміжна операція", "Фінальна", "Змінна", "Тип"}, {"Агрегації", "Файлів", "Сортування",
"Створення"},
            {"Так", "Ні", "Іноді", "Завжди"}
```

```
}
;


    private final int[] correct = {1,0,0,1,0,0,0,0,0,1};
      private int
index = 0;
private int score = 0;


    // ===== PRACTICE =====
    private List<Exercise>
exercises;     private int
exIndex = 0;     private int
practiceScore = 0;


    // ===== INIT =====
    @FXML
    public void initialize() {


        group = new ToggleGroup();


        option1.setToggleGroup(group);
option2.setToggleGroup(group);
option3.setToggleGroup(group);
option4.setToggleGroup(group);


showTheoryMode();

}


    // ===== UI MODES ===== private void showTheoryMode() {quizBox.setVisible(false);
practiceBox.setVisible(false);         contentArea.setVisible(true);
    }


    private void showQuizMode() {
quizBox.setVisible(true);
```

```
practiceBox.setVisible(false);
contentArea.setVisible(false);
    }

    private void showPracticeMode() {
quizBox.setVisible(false);          practiceBox.setVisible(true);
contentArea.setVisible(false);
    }

    // ===== THEORY =====
public void showTheory() {

        showTheoryMode();
          titleLabel.setText("Теорія
Stream API ");

        contentArea.setText("""
              Що таке Stream API?

            Stream API — це спосіб обробки колекцій у Java у функціональному стилі.

            Потік (Stream) НЕ змінює дані, а створює нові
результати. ————————————————————————

              Основні характеристики:

✓ Lazy evaluation (відкладене виконання)

✓ Pipeline (ланцюжок операцій)

✓ Не змінює вихідну колекцію

✓ Одноразове використання

          ————————————————————————
```

Типи операцій:

Intermediate (проміжні):

- map()
- filter()
- sorted()
- distinct()
- limit()

Terminal (кінцеві):

- collect()
- forEach()
- reduce()
- count()
- findFirst()

---

Як працює Stream:

collection
→ stream()
→ filter()
→ map()
→ collect()

---

Важливо:

Stream НЕ зберігає дані — він їх обробляє.
""");
}

```
    // ===== EXAMPLES =====
public void showExamples() {

        showTheoryMode();

        titleLabel.setText("Приклади Stream API");          contentArea.setText("""
1. Фільтрація чисел

            List<Integer> numbers = List.of(1,2,3,4,5,6);

            numbers.stream()
               .filter(n -> n > 3)

.forEach(System.out::println);
            → результат: 4, 5, 6

            ──────────────────────────────────────

2. Перетворення (map)

            List<String> names = List.of("a", "b", "c");

            names.stream()
               .map(String::toUpperCase)
               .forEach(System.out::println);

            → A B C

            ──────────────────────────────────────

3. Сортування

            List<Integer> nums = List.of(5,1,3,2);
```

```
nums.stream()
    .sorted()
    .forEach(System.out::println);
```

→ 1 2 3 5 ——————————————————————————

4. Підрахунок
```
long count = numbers.stream()
    .filter(n -> n > 2)
    .count();
```

——————————————————————————

5. reduce (сума)

```
int sum = numbers.stream()
    .reduce(0, Integer::sum);
```

——————————————————————————

6. collect в список

```
List<Integer> result = numbers.stream()
    .filter(n -> n % 2 == 0)
    .collect(Collectors.toList());
```

——————————————————————————

7. Комбінований приклад

```
List<Integer> result = numbers.stream()
    .filter(n -> n > 2)
    .map(n -> n * 2)
```

```
                .sorted()
                .toList();
            """);
    }

    // ===== QUIZ (ВАЖНО: ЕДИНСТВЕННЫЙ МЕТОД) =====
public void showQuiz() {

        showQuizMode();

index =
0;
score = 0;

        loadQuestion();
    }

    private void loadQuestion() {

        progressLabel.setText("Питання " + (index + 1) + " / 10");

        questionLabel.setText(questions[index]);
option1.setText(answers[index][0]);
option2.setText(answers[index][1]);
option3.setText(answers[index][2]);
option4.setText(answers[index][3]);

        group.selectToggle(null);
    }

    public void checkAnswer() {

        RadioButton selected = (RadioButton)
group.getSelectedToggle();          if (selected == null) return;
```

```
        int selectedIndex =
selected == option1 ? 0 :
selected == option2 ? 1 :
selected == option3 ? 2 : 3;
          if (selectedIndex ==
correct[index]) {
score++;

}
        index++;
          if (index <
questions.length) {
loadQuestion();
        } else {

            showTheoryMode();
              titleLabel.setText("Результат
тесту");

            contentArea.setText("Score: " + score + " / 10");
        }
    }

    // ===== PRACTICE =====

    public void showExercise() {

        showPracticeMode();

        exercises = new ArrayList<>();
          int base = random.nextInt(6) + 5;
// 5–10

        exercises.add(new Exercise(
                "Відфільтруй числа > " + base,
```

```
            "list.stream().filter(x -> x > " + base + ")"
        ));

        exercises.add(new Exercise(
"Помнож кожен елемент на 2",
            "map(x -> x * 2)"
        ));

        exercises.add(new Exercise(
            "Порахуй елементи > " + base,
            "filter(x -> x > " + base + ").count()"
        ));

        exercises.add(new Exercise(
"Перетвори в квадрат",
            "map(x -> x * x)"
        ));
exercises.add(new
Exercise(               "Збери
у список",
            "collect(Collectors.toList())"
        ));

        exercises.add(new Exercise(
            "Фільтр > " + base + " + map + sort",
            "filter(x -> x > " + base + ").map(x -> x * 2).sorted()"
        ));
exIndex =
0;
practiceSco
re = 0;

        showEx();
    }

    private void showEx() {
```

```
        Exercise e = exercises.get(exIndex);

        exerciseLabel.setText(
                "Завдання:\n" + e.getQuestion()
);
        answerField.clear();
    }

    public void checkExercise() {

        Exercise e = exercises.get(exIndex);

        String answer = answerField.getText().trim();

        if (e.check(answer)) {
practiceScore++;
titleLabel.setText("✓ Правильно");
        } else {
            titleLabel.setText("✕ Неправильно\nПравильна відповідь: " + e.getCorrectAnswer());
}
        exIndex++;

        if (exIndex < exercises.size()) {
showEx();         } else {
finishPractice();
        }
    }

    private void finishPractice() {

        showTheoryMode();
```

```
        titleLabel.setText("Результат практики");

        contentArea.setText(
                "Score: " + practiceScore + "/" + exercises.size()
        );
    }

    // ===== RESULTS =====
public void showResults() {

        showTheoryMode();

        titleLabel.setText("Результати навчання");

        StringBuilder sb = new StringBuilder();

        // ===== TEST RESULT =====
sb.append("  РЕЗУЛЬТАТИ ТЕСТІВ\n");
if (index == 0 && score == 0) {
sb.append("Тести ще не проходились\n\n");
        } else {
sb.append("Останній тест: ")
                .append(score)
                .append(" / 10\n\n");
        }

        // ===== PRACTICE RESULT =====
sb.append("  ПРАКТИЧНІ ВПРАВИ\n");

        if (exercises == null) {
sb.append("Практика ще не проходилась\n");
        } else {
sb.append("Останній результат: ")
                .append(practiceScore)
                .append(" / ")
```

```
                .append(exercises.size())
                .append("\n");
        }


        // ===== SUMMARY =====
sb.append("\n  СТАТИСТИКА\n");
          int total = score + practiceScore;          int
max = 10 +
(exercises != null ? exercises.size() : 0);


        sb.append("Загальний бал: ")
                .append(total)
                .append(" / ")
.append(max)
                .append("\n");
          double percent = max == 0 ? 0 : (total * 100.0
/ max);          sb.append("Успішність: ")
                .append(String.format("%.1f", percent))
                .append("%");


        contentArea.setText(sb.toString());
    }
}
```