

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»

Факультет комп'ютерних наук і технологій

(повне найменування факультету)

Кафедра програмних засобів

(повне найменування кафедри)

Пояснювальна записка

до дипломного проєкту (роботи)

бакалавр

(ступінь вищої освіти)

на тему

ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ

ДЛЯ МОДИФІКАЦІЇ ВИХІДНОГО КОДУ

SOFTWARE FOR SOURCE CODE MODIFICATION

Виконав(ла): студент(ка) 4 курсу, групи КНТ-122

Спеціальності 121 Інженерія програмного

(код і найменування спеціальності)

забезпечення

Освітня програма (спеціалізація)

Інженерія програмного забезпечення

ТАРАСОВ А.В.

(ПРИЗВИЩЕ та ініціали)

Керівник СКРУПСЬКИЙ С.Ю.

(ПРИЗВИЩЕ та ініціали)

Рецензент КОЦУР М.І.

(ПРИЗВИЩЕ та ініціали)

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»

Факультет КНТ
Кафедра програмних засобів
Ступінь вищої освіти бакалавр
Спеціальність 121 Інженерія програмного забезпечення
(код і найменування)
Освітня програма (спеціалізація) Інженерія програмного забезпечення
(назва освітньої програми (спеціалізації))

ЗАТВЕРДЖУЮ

Завідувач кафедри ПЗ, д.т.н, проф.
Сергій СУББОТІН
“ ” 2026 року

З А В Д А Н Н Я

НА ДИПЛОМНИЙ ПРОЄКТ (РОБОТУ) СТУДЕНТА(КИ)

ТАРАСОВА Артема Володимировича

(ПРІЗВИЩЕ, ім'я, по батькові)

1. Тема проєкту (роботи) Програмне забезпечення для модифікації вихідного коду. Software for Source Code Modification

керівник проєкту (роботи) к.т.н., доцент, СКРУПСЬКИЙ Степан Юрійович,
(науковий ступінь, вчене звання, ПРІЗВИЩЕ, ім'я, по батькові)

затверджені наказом закладу вищої освіти від “ 7 ” квітня 2026 року № 139

2. Строк подання студентом проєкту (роботи) 03 червня 2026 року

3. Вихідні дані до проєкту (роботи) рекомендована література, технічне завдання

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити) 1. Аналіз предметної області. 2. Матеріали і методи. 3. Опис програми. 4. Експлуатація, тестування та експериментальне дослідження програми.

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень, кількість слайдів, плакатів) Слайди презентації

6. Консультанти розділів проєкту (роботи)

Розділ	ПРИЗВИЩЕ, ініціали та посада консультанта	Підпис, дата	
		завдання видав	прийняв виконане завдання
1-4 Основна частина	СКРУПСЬКИЙ С.Ю., доцент		
Нормоконтроль	ДЕЙНЕГА Л.Ю., ст. викладач		

7. Дата видачі завдання “ 7 ” квітня 2026 року.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломного проєкту (роботи)	Строк виконання етапів проєкту (роботи)	Примітка
1	Постановка завдання роботи.	1 тиждень	Завдання, ТЗ
2	Аналіз предметної області.	1 тиждень	Розділ 1
3	Вибір мови програмування та інших технологій розробки.	2 тиждень	Розділ 2
4	Розробка структури програми.	2 тиждень	Розділ 3
5	Розробка програми.	3-4 тижні	Розділи 3, 4
6	Тестування та експериментальне дослідження програмного забезпечення.	5 тиждень	Розділ 4
7	Оформлення пояснювальної записки та документів до неї.	6 тиждень	Додатки
8	Нормоконтроль та рецензування.	7 тиждень	
9	Захист роботи.	8 тиждень	

Студент(ка)

_____ Артем ТАРАСОВ
(підпис) (Ім'я ПРИЗВИЩЕ)

Керівник проєкту (роботи)

_____ Степан СКРУПСЬКИЙ
(підпис) (Ім'я ПРИЗВИЩЕ)

РЕФЕРАТ

Пояснювальна записка до дипломної кваліфікаційної роботи бакалавра:
85 с., 3 табл., 24 рис., 3 дод., 21 джерел.

JAVA, ВИХІДНИЙ КОД, МОВА ПРОГРАМУВАННЯ, ПРОГРАМНЕ
ЗАБЕЗПЕЧЕННЯ, РЕФАКТОРИНГ.

Об'єкт дослідження – процес розробки програмного забезпечення для модифікації вихідного коду.

Предмет дослідження – програмні засоби для модифікації вихідного коду.

Мета роботи – розробка програмного забезпечення для модифікації вихідного коду, що забезпечує скорочення трудомісткості супроводу та рефакторингу програм.

Матеріали, методи та технічні засоби: мова програмування Java, середовище розробки IntelliJ IDEA.

Результати. Розроблено проєктні рішення для створення програмного забезпечення для модифікації вихідного коду. Створено програмне забезпечення для модифікації вихідного коду. Проведено тестування розробленої програмної системи для модифікації вихідного коду.

Висновки. Мету роботи досягнуто. Розроблено програмне забезпечення для модифікації вихідного коду за допомогою мови програмування Java та середовища розробки IntelliJ IDEA.

Галузь використання – супровід програмного забезпечення.

ABSTRACT

Explanatory note to the diploma qualifying work of the bachelor: 85 pages, 3 tables, 24 figures, 3 appendixes, 21 sources.

JAVA, SOURCE CODE, PROGRAMMING LANGUAGE, SOFTWARE, REFACTORING.

The object of research is the process of developing software for modifying source code.

The subject of the research is software for modifying source code.

The purpose of this work is to develop the software for modifying source code.

Materials, methods and technical tools: Java programming language, IntelliJ IDEA development environment.

Results. Project solutions for the creation of software for modifying source code has been designed. The software to support the process of modifying source code has been developed. The developed software system for modifying source code has been tested.

Conclusions. The goal of the work has been achieved. The software tools for modifying source code using the Java programming language and the IntelliJ IDEA development environment have been developed.

Scope of use – software support.

ЗМІСТ

	С.
Перелік скорочень та умовних позначок	8
Вступ.....	9
1 Аналіз предметної області.....	12
1.1 Програмне забезпечення для модифікації вихідного коду	12
1.2 Аналіз програмного забезпечення для модифікації вихідного коду	18
1.3 Висновки за розділом 1	27
2 Матеріали і методи	29
2.1 Вибір мови програмування	29
2.2 Вибір середовища розробки для створення програмного забезпечення для модифікації вихідного коду	32
2.3 Висновки за розділом 2	36
3 Опис програми	37
3.1 Структура програмного забезпечення для модифікації вихідного коду	37
3.2 Функціонування програмного забезпечення для модифікації вихідного коду.....	39
3.3 Особливості реалізації програмного забезпечення для модифікації вихідного коду.....	42
3.4 Опис організації зберігання даних програмного забезпечення для модифікації вихідного коду	44
3.5 Проєктування інтерфейсу програмного забезпечення для модифікації вихідного коду.....	46
3.6 Висновки за розділом 3	50
4 Експлуатація, тестування та експериментальне дослідження програми.....	52
4.1 Призначення й умови застосування програми	52
4.2 Характеристики програми для модифікації вихідного коду.....	54
4.3 Інструкція по експлуатації програми.....	55
4.3.1 Звернення до програми.....	55
4.3.2 Вхідні й вихідні дані	55

4.3.3 Повідомлення.....	55
4.4 Виконання програмного забезпечення для модифікації вихідного коду	56
4.5 Тестування програмного забезпечення для модифікації вихідного коду.....	58
4.6 Висновки за розділом 4	58
Висновки	59
Перелік джерел посилання	63
Додаток А Технічне завдання	65
Додаток Б Фрагмент тексту програми	70
Додаток В Слайди презентації	78

ПЕРЕЛІК СКОРОЧЕНЬ ТА УМОВНИХ ПОЗНАК

MVC – Model-View-Controller;

ВХ – вихідний код;

БД – база даних;

ПЗ – програмне забезпечення.

ВСТУП

Актуальність програмного забезпечення для модифікації вихідного коду зумовлюється стрімким розвитком інформаційних технологій, зростанням складності програмних систем та підвищенням вимог до їх якості, безпеки й адаптивності. У сучасних умовах програмні продукти рідко створюються «з нуля» і дедалі частіше базуються на вже існуючих кодових базах, фреймворках і бібліотеках, що потребують постійного доопрацювання, оптимізації та приведення у відповідність до нових технічних або бізнес-вимог. Саме тому інструменти, орієнтовані на зміну, аналіз і перетворення вихідного коду, відіграють ключову роль у процесах супроводу та розвитку програмного забезпечення [1], [2].

Значна увага до таких програмних засобів пояснюється необхідністю скорочення часу на внесення змін і зменшення ймовірності помилок, які можуть виникати під час ручного редагування коду. Модифікація вихідного коду часто пов'язана з великими обсягами даних, складними взаємозв'язками між модулями та залежностями, що ускладнює розуміння наслідків навіть незначних змін. Використання спеціалізованого програмного забезпечення дозволяє автоматизувати частину цих процесів, підвищити прозорість структури коду та забезпечити контроль цілісності програмної системи. Актуальність таких програмних рішень також посилюється вимогами до довготривалої підтримки програмних продуктів. У багатьох організаціях експлуатуються системи, розроблені роки або навіть десятиліття тому, які необхідно адаптувати до нових платформ, стандартів або умов використання. Програмне забезпечення для модифікації вихідного коду дає можливість працювати з застарілими або слабо документованими проєктами, полегшуючи процес їх модернізації та знижуючи ризики, пов'язані з повною заміною таких систем [3], [4].

Таким чином, програмне забезпечення для модифікації вихідного коду є невід'ємною складовою сучасної програмної інженерії, оскільки забезпечує

гнучкість, керованість і стійкість процесів розвитку програмних систем. Його актуальність визначається потребою швидко реагувати на зміни, ефективно використовувати наявні ресурси та підтримувати високий рівень якості програмних рішень упродовж усього життєвого циклу [4]-[6].

Проте деякі програми для модифікації вихідного коду можуть бути орієнтовані на роботу з великими та структурно складними кодовими базами, що вимагає глибокого розуміння архітектури програмного продукту, мов програмування та принципів побудови програмних систем. За відсутності належних знань або досвіду використання подібного програмного забезпечення може призводити до помилкових змін, які складно виявити та усунути на пізніших етапах розробки. Ще одним недоліком є ризик порушення логіки роботи програмної системи внаслідок автоматизованих або напівавтоматизованих змін. Інструменти для модифікації вихідного коду не завжди здатні коректно враховувати всі контекстні зв'язки між модулями, особливо у випадку нестандартних рішень або неочевидних залежностей. У результаті навіть формально коректні зміни можуть призводити до прихованих дефектів, які проявляються лише під час експлуатації або під значним навантаженням. Недоліком також є обмежена універсальність багатьох інструментів, які підтримують лише певні мови програмування, фреймворки або стилі кодування. Це знижує можливість їх застосування в різномірних проєктах і створює залежність від конкретних технологій або постачальників програмного забезпечення [1]-[6].

Тому актуальною є розробка програмного забезпечення для модифікації вихідного коду. У дипломній кваліфікаційній роботі бакалавра розв'язується актуальне завдання розробки програмного забезпечення для модифікації вихідного коду, що забезпечує скорочення трудомісткості супроводу та рефакторингу програм.

Для досягнення поставленої мети у кваліфікаційній роботі бакалавра необхідно розв'язати такі задачі:

- виконати аналіз предметної області та програмних засобів для модифікації вихідного коду;
- здійснити проєктування програмного забезпечення для модифікації вихідного коду;
- створити програмне забезпечення для модифікації вихідного коду;
- виконати тестування розробленого програмного забезпечення для модифікації вихідного коду.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Програмне забезпечення для модифікації вихідного коду

Модифікація вихідного коду являє собою процес цілеспрямованої зміни програмного тексту з метою поліпшення його властивостей без порушення загальної логіки роботи програмного продукту. У межах цього процесу вихідний код може трансформуватися шляхом перейменування ідентифікаторів, перетворення структур, оптимізації окремих фрагментів або застосування спеціальних технік, спрямованих на підвищення стійкості програмного забезпечення до аналізу та несанкціонованого використання. Така діяльність не обмежується лише виправленням помилок, а охоплює більш широкий спектр змін, що впливають на внутрішню організацію програмної системи [1]-[7].

У загальному випадку модифікація вихідного коду дозволяє адаптувати програмне забезпечення до нових вимог, умов експлуатації або обмежень середовища виконання. Вона може застосовуватися для підвищення рівня захищеності інтелектуальної власності, ускладнення зворотної розробки, зменшення залежності від конкретних реалізацій або для приведення коду у відповідність до встановлених стандартів якості. У результаті таких перетворень програмний продукт зберігає свою функціональність, але набуває нових властивостей, які є важливими з точки зору супроводу, масштабування або безпеки [1].

Особливістю процесу модифікації вихідного коду є необхідність точного аналізу синтаксичної та семантичної структури програми, оскільки будь-які некоректні зміни можуть призвести до порушення працездатності системи. Тому цей процес зазвичай спирається на формалізовані алгоритми, автоматизовані інструменти та чітко визначені правила трансформації. Важливу роль відіграє збереження балансу між глибиною змін і стабільністю

роботи програмного забезпечення, що вимагає ретельного контролю та перевірки результатів [2], [3].

Перевагою модифікації вихідного коду є можливість значно підвищити керованість програмного продукту та подовжити його життєвий цикл без необхідності повного переписування системи. Такий підхід сприяє зниженню витрат на підтримку, полегшує адаптацію до змінних вимог і дозволяє більш ефективно захищати програмні рішення від копіювання або несанкціонованого аналізу. У сукупності це робить модифікацію вихідного коду важливим інструментом у сучасній практиці розроблення та супроводу програмного забезпечення [2], [3].

Управління вихідним кодом являє собою організований процес контролю та відстеження змін, що вносяться до програмного коду протягом усього циклу розробки програмного продукту. Воно забезпечує систематичне ведення змін, дозволяючи відслідковувати будь-які модифікації в кодовій базі та зберігати історію їхнього внесення [1].

Репозиторій вихідного коду виступає центральним сховищем для програмного коду та супутніх ресурсів, таких як тестові сценарії, документація та допоміжні скрипти. Він використовується для організації коду, його структуризації та полегшення колективної роботи над проектом. У цьому сховищі фіксуються всі зміни, що вносяться різними учасниками розробки, і відбувається координація роботи над кодовою базою [1], [2].

Постійне ведення обліку змін дозволяє підтримувати код у актуальному стані, забезпечуючи розробникам, тестувальникам та аналітикам доступ до точних версій програми. Це також сприяє ефективному вирішенню конфліктів, які можуть виникнути при паралельній роботі над різними частинами коду [1], [2].

Крім цього, системи управління вихідним кодом підвищують ефективність командної роботи, запобігаючи випадковому перезапису або втраті змін, внесених іншими учасниками проекту. Часто поняття управління вихідним кодом розглядається як синонім контролю версій, оскільки обидва

підходи виконують взаємопов'язані функції збереження історії та впорядкування змін у програмному забезпеченні [1], [2].

Програмне забезпечення для модифікації вихідного коду розглядається як спеціалізований клас програмних засобів, призначених для автоматизованого або напівавтоматизованого перетворення програмного тексту без зміни його функціонального призначення. Такі системи працюють з вихідним кодом на рівні мовних конструкцій, аналізуючи його структуру, логіку та залежності між окремими елементами, після чого виконують визначені перетворення відповідно до заданих правил або налаштувань користувача. Основною метою використання подібного програмного забезпечення є зміна внутрішнього представлення коду з урахуванням вимог безпеки, супроводу або адаптації до певних умов експлуатації [3], [4].

Особливістю програмного забезпечення для модифікації вихідного коду є орієнтація на глибокий аналіз програмного тексту з урахуванням синтаксичних та семантичних особливостей мови програмування. Це дозволяє виконувати перетворення вибірково, не порушуючи коректність виконання програми та не створюючи помилок компіляції чи виконання. У таких системах зазвичай реалізуються механізми контролю допустимих змін, фільтрації ідентифікаторів, обробки рядкових констант та інших чутливих елементів коду, що потребують обережного підходу. Завдяки цьому забезпечується збереження логіки програми при суттєвій зміні її зовнішнього вигляду [3], [4].

Переваги використання програмного забезпечення для модифікації вихідного коду полягають у підвищенні ефективності роботи з великими обсягами програмних текстів та зменшенні впливу людського фактору. Автоматизація процесів дозволяє виконувати складні та повторювані перетворення швидко і з однаковим рівнем якості, що є особливо важливим під час роботи з масштабними проєктами. Крім того, такі програмні засоби сприяють підвищенню рівня захисту програмного забезпечення, ускладнюючи

аналіз коду та його несанкціоноване використання, що має значення для збереження інтелектуальної власності [4], [5].

Ще однією важливою рисою програмного забезпечення для модифікації вихідного коду є гнучкість та можливість адаптації до різних сценаріїв застосування. Користувач може налаштовувати характер і глибину перетворень відповідно до власних потреб, поєднуючи базові та розширені методи модифікації. Це робить такі системи універсальним інструментом, який може використовуватися як у процесі розроблення, так і на етапах супроводу або підготовки програмного продукту до розповсюдження. У сукупності зазначені властивості визначають програмне забезпечення для модифікації вихідного коду як важливий елемент сучасних технологій розроблення та захисту програмних систем [4], [5].

Методи модифікації вихідного коду розглядаються як сукупність підходів, що застосовуються для цілеспрямованої зміни програмного тексту з метою досягнення певних технічних або організаційних результатів без порушення коректності роботи програмної системи. У межах таких підходів основна увага приділяється аналізу структури коду, взаємозв'язків між його елементами та особливостей конкретної мови програмування, що дозволяє виконувати перетворення на різних рівнях абстракції. Вибір методу модифікації зазвичай визначається завданнями, які стоять перед розробником, а також вимогами до надійності, безпеки та подальшого супроводу програмного продукту [6], [7].

Одним із поширених підходів є структурна модифікація, яка ґрунтується на зміні внутрішньої організації програмного коду. У межах такого підходу відбувається перетворення ієрархії класів, методів або логічних блоків із збереженням їх функціонального призначення. Подібні дії дозволяють ускладнити сприйняття коду або, навпаки, адаптувати його до нових вимог середовища виконання. Значна увага при цьому приділяється коректності синтаксису та семантики, оскільки навіть незначні помилки можуть призвести до порушення роботи програми [6], [7].

Іншим важливим напрямом є лексична та символна модифікація, що передбачає зміну імен змінних, методів, класів або інших ідентифікаторів. Такий підхід широко застосовується у випадках, коли необхідно ускладнити аналіз коду або приховати його внутрішню логіку від стороннього перегляду. При цьому зберігається загальна структура програми, а зміни торкаються лише текстового представлення окремих елементів. Ефективність цього методу значною мірою залежить від якості аналізу вихідного коду та здатності системи коректно відрізнити службові конструкції від користувацьких [6], [7].

Окрему групу становлять методи, орієнтовані на трансформацію програмної логіки без зміни кінцевого результату виконання. У межах такого підходу можуть застосовуватися альтернативні конструкції, додаткові проміжні обчислення або перетворення виразів, що призводить до ускладнення логічного аналізу програми. Подібні методи потребують глибокого розуміння семантики мови програмування, оскільки будь-яке відхилення від коректної логіки може негативно вплинути на стабільність програмного продукту [6], [7].

Також застосовуються підходи, пов'язані з обробкою допоміжних елементів вихідного коду, зокрема текстових констант, коментарів або діагностичних повідомлень. Модифікація таких компонентів дозволяє зменшити обсяг надлишкової інформації, змінити спосіб представлення даних або підвищити рівень захисту програмного забезпечення. У поєднанні з іншими методами ці підходи формують комплексну систему модифікації вихідного коду, яка забезпечує гнучкість, універсальність та відповідність сучасним вимогам до розроблення і захисту програмних систем [6], [7].

Модифікація вихідного коду шляхом обфускації розглядається як спеціалізований процес перетворення програмного тексту, метою якого є ускладнення його сприйняття людиною без втрати функціональної правильності програми. У межах цього процесу структура та логіка коду залишаються незмінними для компілятора або середовища виконання, однак змінюється спосіб подання інформації у вихідному коді. Використання

обфускації дозволяє захищати інтелектуальну власність розробника, ускладнює несанкціонований аналіз програми та підвищує рівень безпеки програмного продукту в умовах потенційної загрози копіювання або зворотної інженерії [6], [7].

Обфускація включає різноманітні методи, що можуть застосовуватися окремо або у комбінації, залежно від поставлених завдань і рівня необхідного захисту. До таких методів відноситься зміна імен змінних, методів та класів на довільні або випадково згенеровані послідовності символів, що ускладнює читання та розуміння коду сторонніми особами. Крім того, можуть застосовуватися трансформації структурних блоків, перетворення виразів та алгоритмів на еквівалентні за результатом виконання, але більш складні для аналізу. Додатково здійснюється обробка допоміжних елементів коду, таких як коментарі, константні рядки або повідомлення, що дозволяє приховати інформацію про логіку роботи програми [6], [7].

Особливістю обфускації є баланс між захистом та працездатністю програмного продукту. Перетворення виконуються так, щоб не порушити компіляцію та виконання коду, при цьому максимально ускладнюючи його розбір для людини. Використання автоматизованих інструментів для обфускації дозволяє досягати високого рівня захисту без потреби ручного втручання в код, скорочує час на обробку великих проєктів і забезпечує повторюваність результатів. У результаті програмне забезпечення набуває додаткової стійкості до зворотної розробки та несанкціонованого використання, що робить обфускацію одним із ключових підходів у сучасній практиці захисту програмних систем [6], [7].

Визначено, що модифікація вихідного коду являє собою процес цілеспрямованої зміни програмного тексту з метою поліпшення його властивостей без порушення загальної логіки роботи програмного продукту. У межах цього процесу вихідний код може трансформуватися шляхом перейменування ідентифікаторів, перетворення структур, оптимізації окремих фрагментів або застосування спеціальних технік, спрямованих на підвищення

стійкості програмного забезпечення до аналізу та несанкціонованого використання.

1.2 Аналіз програмного забезпечення для модифікації вихідного коду

Проаналізуємо програмне забезпечення для модифікації вихідного коду [8]-[17].

Програмне забезпечення DeepCode (рис. 1.1) функціонує як складова екосистеми Snyk для інтелектуальної модифікації та вдосконалення вихідного коду. Його робота ґрунтується на застосуванні методів штучного інтелекту, що дає змогу виконувати глибокий аналіз програмних текстів безпосередньо під час процесу розробки. У результаті користувач отримує рекомендації щодо покращення коду та повідомлення про можливі недоліки ще на етапі написання або редагування, що сприяє підвищенню загальної якості програмного продукту [8], [15].

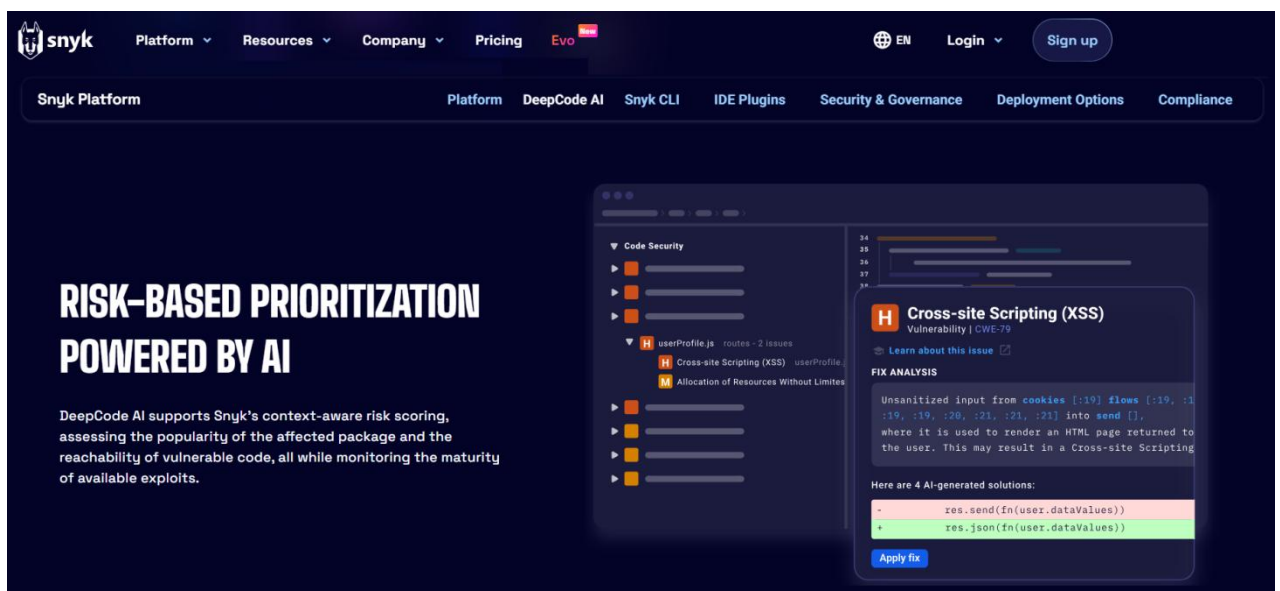


Рисунок 1.1 – Програмне забезпечення DeepCode [15]

Особлива увага в межах програмного забезпечення DeepCode приділяється аспектам безпеки. Аналіз здійснюється з урахуванням накопиченого досвіду та знань про типові помилки, уразливі конструкції та

поширені сценарії атак. Завдяки цьому формується контекстно обґрунтована інформація, яка допомагає виявляти ризики, пов'язані з порушенням захищеності програмних рішень, ще до моменту їх розгортання або використання у робочому середовищі. Такий підхід дозволяє програмному забезпеченню DeepCode зменшити ймовірність виникнення критичних проблем і підвищує довіру до кінцевого програмного продукту [8], [15].

Програмне забезпечення DeepCode орієнтоване на роботу з різними мовами програмування, що робить його універсальним інструментом для команд із різним технологічним стеком. Воно здатне не лише сигналізувати про потенційно небезпечні фрагменти коду, а й пропонувати варіанти їх коригування, що спрощує процес усунення недоліків і знижує навантаження на розробників. Крім того, рішення органічно поєднується з процесами автоматизованої збірки та розгортання програмних продуктів, забезпечуючи безперервний контроль якості та безпеки коду в межах сучасних підходів до розробки програмного забезпечення [8], [15].

Основними характеристиками програмного забезпечення DeepCode є такі [9], [10], [12]:

- аналіз коду в реальному часі: програмне забезпечення DeepCode здійснює безперервну перевірку вихідного коду під час його написання або редагування, що дозволяє оперативно виявляти потенційні помилки та небезпечні конструкції [8], [15];

- орієнтація на безпеку: у процесі аналізу основний акцент робиться на пошуку вразливостей, логічних помилок і шаблонів, які можуть призвести до порушення захищеності програмного продукту [8], [15];

- використання штучного інтелекту: механізми перевірки коду у програмному забезпеченні DeepCode базуються на машинному навчанні та аналізі великої кількості відкритих і промислових проєктів, що підвищує точність рекомендацій [8], [15];

- підтримка різних мов програмування: у програмному забезпеченні DeepCode підтримується можливість роботи з широким спектром сучасних

мов, що робить інструмент придатним для застосування в багатомовних проєктах [8], [15];

- інтеграція з середовищами розробки: програмне забезпечення DeepCode легко поєднується з популярними редакторами коду та платформами автоматизації розробки, не порушуючи звичного робочого процесу [8], [15];

- автоматичні рекомендації з виправлення: користувачеві надаються не лише повідомлення про проблеми, а й пропозиції щодо їх усунення, що скорочує час на доопрацювання коду [8], [15];

- масштабованість використання: програмне забезпечення DeepCode може застосовуватися як у невеликих проєктах, так і в складних корпоративних системах з великою кодовою базою [8], [15].

Серед недоліків програмного забезпечення DeepCode можна відзначити залежність якості результатів від повноти підтримки конкретної мови програмування, а також обмежену гнучкість налаштувань аналізу для нетипових або вузькоспеціалізованих сценаріїв розробки, що в окремих випадках може потребувати додаткових інструментів перевірки [8], [15].

Програмне забезпечення CodeAnt AI (рис. 1.2) представляє собою інтелектуальний інструмент для модифікації та перевірки вихідного коду, що виконує роль віртуального помічника розробника у процесі командної роботи. Функціонування платформи CodeAnt AI ґрунтується на поєднанні алгоритмів штучного інтелекту з методами статичного аналізу, завдяки чому забезпечується глибоке опрацювання програмного коду та своєчасне виявлення потенційних помилок, недоліків структури або проблемних фрагментів. Інструмент вбудовується у популярні системи керування репозиторіями та процесами розробки, що дозволяє отримувати оперативний зворотний зв'язок безпосередньо під час роботи з кодом. У результаті взаємодія з платформою не потребує суттєвих змін звичного робочого середовища та органічно доповнює існуючі процеси розробки. Особливістю CodeAnt AI є автоматизація типових операцій, пов'язаних із перевіркою та

вдосконаленням вихідного коду, що знижує навантаження на розробників і мінімізує рутинну діяльність [8], [16].

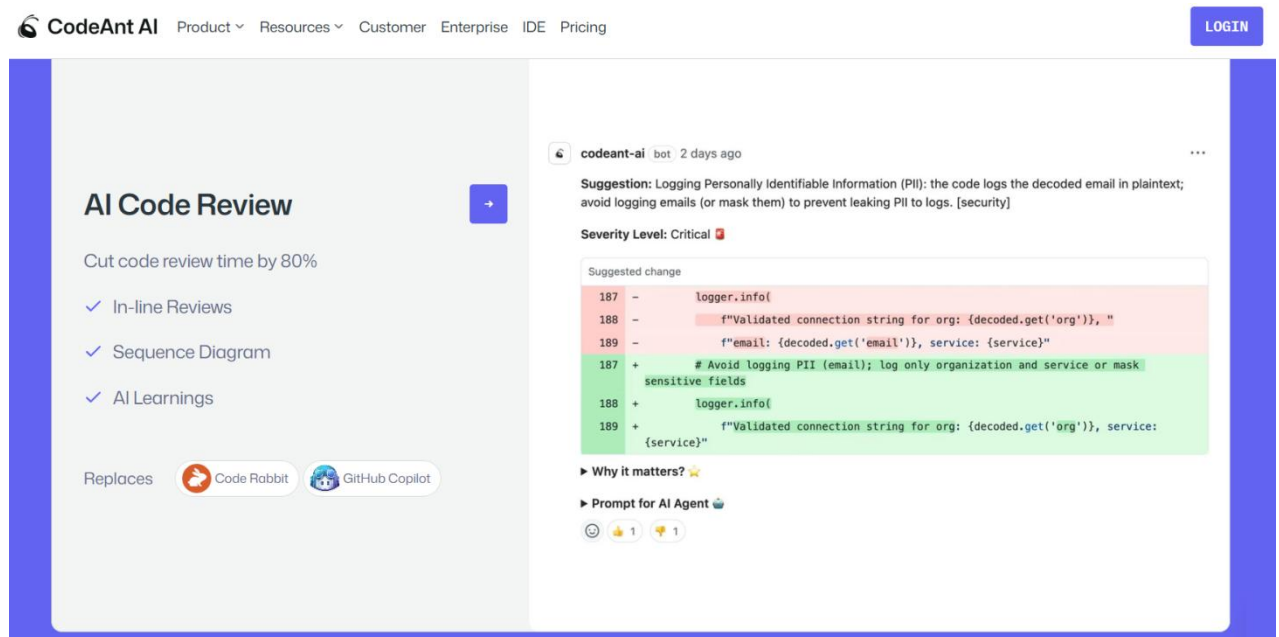


Рисунок 1.2 – Програмне забезпечення CodeAnt AI [16]

Під час аналізу програмних проєктів система формує зрозумілі рекомендації щодо можливих виправлень і оптимізацій, які можуть бути застосовані без тривалого ручного доопрацювання. Окрема увага приділяється перевірці змін, що вносяться у кодову базу, оскільки платформа автоматично аналізує оновлення та формує узагальнені звіти з позначенням проблемних місць. Такий підхід сприяє підвищенню якості коду, поліпшенню командної взаємодії та більш усвідомленому прийняттю рішень під час розвитку програмного продукту. Крім того, можливість постійного аналізу кодової бази дозволяє підтримувати її у стабільному та впорядкованому стані протягом усього життєвого циклу програмного забезпечення [8], [16].

Програмне забезпечення CodeAnt AI має такі особливості [8], [16]:

- інтелектуальний аналіз коду: платформа CodeAnt AI застосовує алгоритми штучного інтелекту для глибокого опрацювання вихідного коду з метою виявлення помилок, потенційних ризиків та можливостей для покращення структури програмних рішень [8], [16];

- інтеграція з репозиторіями: забезпечується тісна взаємодія з поширеними системами керування версіями, що дозволяє аналізувати зміни безпосередньо у процесі командної розробки та отримувати миттєвий зворотний зв'язок [8], [16];

- автоматизація перевірок: у програмному забезпеченні CodeAnt AI реалізується автоматичне опрацювання змін у кодовій базі з формуванням узагальнених результатів перевірки коду, що зменшує потребу у ручному аналізі [8], [16];

- рекомендації щодо модифікації: програмне забезпечення CodeAnt AI формує зрозумілі підказки та пропозиції з удосконалення коду, які можуть бути застосовані безпосередньо під час роботи з проєктом [8], [16];

- підтримка командної роботи: функціонування платформи CodeAnt AI орієнтоване на колективну розробку, що сприяє узгодженості рішень та підвищенню якості програмного продукту [8], [16];

- безперервний контроль якості: забезпечується постійний аналіз кодової бази протягом усього процесу розробки, що дозволяє своєчасно реагувати на накопичення технічних недоліків [8], [16].

Серед недоліків програмного забезпечення CodeAnt AI можна відзначити залежність якості результатів від специфіки проєкту та структури коду, а також потребу у певному часі для адаптації інструмента до внутрішніх стандартів і стилю розробки конкретної команди [8], [16].

Програмне забезпечення Refact.ai (рис. 1.3) позиціонується як інтелектуальна система для модифікації та аналізу вихідного коду, побудована з акцентом на автономність і захист даних розробника. Функціонування цієї платформи ґрунтується на відкритій архітектурі, що дозволяє виконувати всі операції безпосередньо в межах локальної інфраструктури організації. Такий підхід забезпечує високий рівень контролю над кодовою базою та мінімізує ризики, пов'язані з передаванням чутливої інформації стороннім сервісам. У процесі роботи інструмент застосовує методи штучного інтелекту для аналізу структури програм, виявлення потенційних недоліків і формування

рекомендацій щодо вдосконалення коду, не порушуючи вимог до конфіденційності [8], [17].

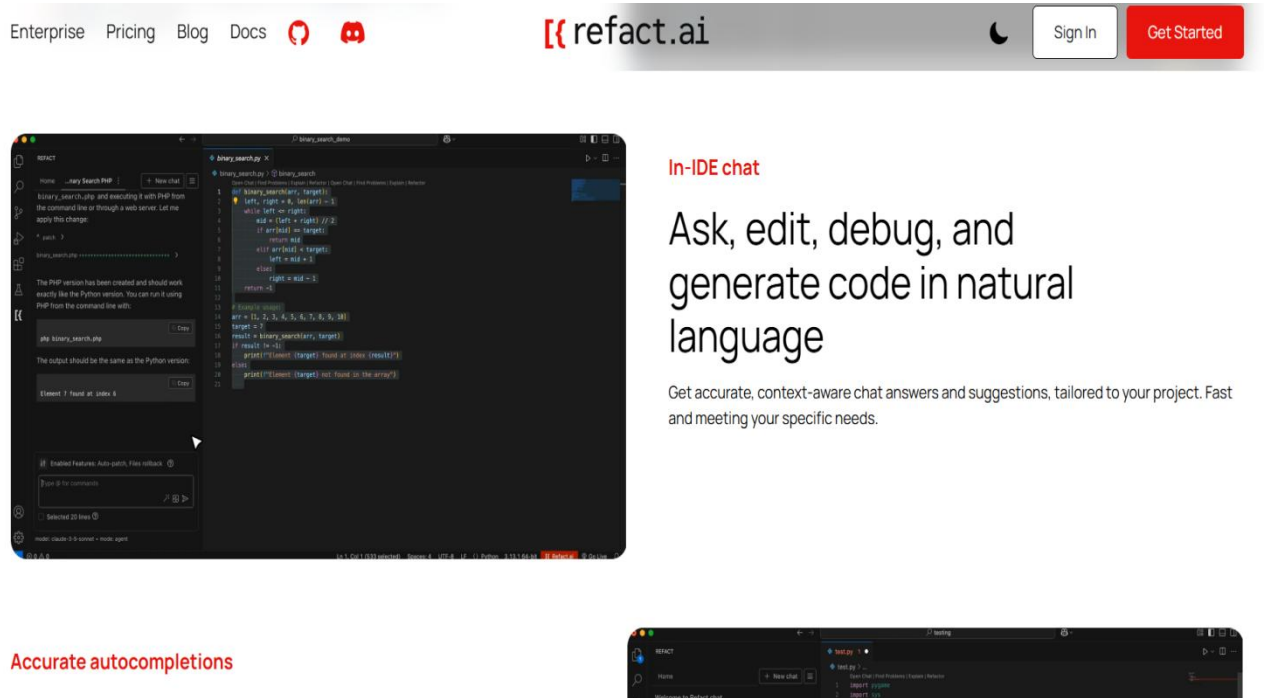


Рисунок 1.3 – Програмне забезпечення Refact.ai [17]

Особливістю Refact.ai є орієнтація на захист інтелектуальної власності, що робить його придатним для використання у корпоративному середовищі з підвищеними вимогами до безпеки. Платформа побудована таким чином, що всі обчислення та перевірки виконуються локально, а вихідний код не залишає внутрішніх ресурсів організації. При цьому за рівнем аналітичних можливостей система не поступається рішенням, які базуються на хмарних технологіях. Інтеграційні механізми дозволяють гнучко вбудовувати Refact.ai у наявні процеси розробки, адаптуючи його до специфіки конкретного проєкту та внутрішніх стандартів команди [8], [17].

Завдяки підтримці різних мов програмування та розширюваності через програмні інтерфейси, Refact.ai може використовуватися як універсальний інструмент для аналізу й модифікації коду у різноманітних програмних продуктах. Особливу увагу приділено аспектам безпеки, зокрема під час роботи з програмами, орієнтованими на мобільні платформи, де вимоги до захисту даних є особливо жорсткими. У сукупності це дозволяє розглядати Refact.ai як

комплексне рішення, що поєднує інтелектуальний аналіз коду з високим рівнем автономності та довіри до середовища виконання [8], [17].

Програмне забезпечення Refact.ai має такі особливості [8], [17]:

- орієнтація на конфіденційність: обробка та аналіз вихідного коду виконуються у програмному забезпеченні Refact.ai локально в межах інфраструктури організації, що мінімізує ризики витоку даних і забезпечує контроль над інтелектуальною власністю [8], [17];

- відкрита архітектура платформи: програмне забезпечення Refact.ai побудоване на основі відкритого коду, що спрощує аудит, розширення функціональності та адаптацію до внутрішніх стандартів розробки [8], [17];

- використання штучного інтелекту: у процесі аналізу застосовуються інтелектуальні алгоритми для виявлення проблем, оптимізації структури коду та формування рекомендацій щодо його покращення [8], [17];

- підтримка різних мов програмування: програмне забезпечення Refact.ai здатна працювати з кількома популярними мовами, що дозволяє використовувати її у різнопрофільних програмних проєктах [8], [17];

- інтеграція через програмні інтерфейси: наявність API дає змогу вбудовувати Refact.ai у наявні процеси розробки та автоматизувати аналіз коду у межах внутрішніх робочих потоків [8], [17];

- корпоративний рівень безпеки: у програмному забезпеченні Refact.ai реалізовано механізми захисту, які відповідають вимогам організацій з підвищеними стандартами інформаційної безпеки [8], [17];

- спеціалізований аналіз прикладної безпеки: програмне забезпечення Refact.ai приділяє увагу перевірці програм на наявність уразливостей, зокрема у застосунках для різних середовищ виконання [8], [17];

- альтернатива хмарним рішенням: функціональні можливості аналізу наближені до хмарних сервісів, але без необхідності передавання коду зовнішнім постачальникам [8], [17].

Серед недоліків програмного забезпечення Refact.ai можна відзначити підвищені вимоги до обчислювальних ресурсів при локальному розгортанні, а

також необхідність самостійного налаштування й супроводу системи, що може ускладнювати впровадження для невеликих команд або проєктів з обмеженими технічними можливостями [8], [17].

Порівняльну характеристику програмного забезпечення для модифікації вихідного коду наведено у таблиці 1.1.

Таблиця 1.1 – Порівняння програмного забезпечення для модифікації вихідного коду

Критерій порівняння	DeepCode [15]	CodeAnt AI [16]	Refact.ai [17]
Автоматизація перевірок та виправлень	+	+	+—
Простота впровадження	+	+	+—
Інтеграція з системами контролю версій	+—	+	+—
Орієнтація на безпеку	+	+—	+
Можливість локального розгортання	—	—	+
Гнучкість налаштувань під проєкт	+—	+	+

За результатами проведеного аналізу можна зробити висновок, що у наш час існує досить багато програмних засобів для модифікації вихідного коду. Проте деякі програми для модифікації вихідного коду можуть бути орієнтовані на роботу з великими та структурно складними кодовими базами, що вимагає глибокого розуміння архітектури програмного продукту, мов програмування та принципів побудови програмних систем. За відсутності належних знань або досвіду використання подібного програмного забезпечення може призводити до помилкових змін, які складно виявити та

усунути на пізніших етапах розробки. Ще одним недоліком є ризик порушення логіки роботи програмної системи внаслідок автоматизованих або напівавтоматизованих змін. Інструменти для модифікації вихідного коду не завжди здатні коректно враховувати всі контекстні зв'язки між модулями, особливо у випадку нестандартних рішень або неочевидних залежностей. У результаті навіть формально коректні зміни можуть призводити до прихованих дефектів, які проявляються лише під час експлуатації або під значним навантаженням. Недоліком також є обмежена універсальність багатьох інструментів, які підтримують лише певні мови програмування, фреймворки або стилі кодування. Це знижує можливість їх застосування в різномірних проєктах і створює залежність від конкретних технологій або постачальників програмного забезпечення. Тому актуальною є розробка програмного забезпечення для модифікації вихідного коду.

При розробці програмного забезпечення для модифікації вихідного коду необхідно забезпечити такі функціональні вимоги:

- підтримка можливості завантаження та обробки файлів вихідного коду програмних проєктів з файлової системи користувача;
- підтримка можливості аналізу синтаксичної структури вихідного коду з метою коректного виявлення ідентифікаторів, константних рядків та інших елементів, придатних до модифікації;
- забезпечення можливості налаштування параметрів модифікації вихідного коду відповідно до вимог користувача через графічний інтерфейс;
- підтримка можливості обфускації назв класів, методів та змінних без порушення логіки виконання програми;
- підтримка можливості вибіркового застосування додаткових методів модифікації, зокрема видалення коментарів, консольних повідомлень та конструкцій обробки винятків;
- забезпечення стабільної роботи графічного інтерфейсу під час виконання ресурсоємних операцій шляхом використання окремих потоків виконання;

- підтримка можливості формування окремих файлів з модифікованим вихідним кодом без перезапису оригінальних файлів;
- забезпечення інформування користувача про стан виконання процесу модифікації та результати обробки файлів;
- підтримка можливості подальшого використання згенерованих файлів у складі нового або існуючого програмного проєкту.

1.3 Висновки за розділом 1

Визначено, що модифікація вихідного коду являє собою процес цілеспрямованої зміни програмного тексту з метою поліпшення його властивостей без порушення загальної логіки роботи програмного продукту. У межах цього процесу вихідний код може трансформуватися шляхом перейменування ідентифікаторів, перетворення структур, оптимізації окремих фрагментів або застосування спеціальних технік, спрямованих на підвищення стійкості програмного забезпечення до аналізу та несанкціонованого використання.

За результатами проведеного аналізу зроблено висновок, що у наш час існує досить багато програмних засобів для модифікації вихідного коду. Проте деякі програми для модифікації вихідного коду можуть бути орієнтовані на роботу з великими та структурно складними кодовими базами, що вимагає глибокого розуміння архітектури програмного продукту, мов програмування та принципів побудови програмних систем. За відсутності належних знань або досвіду використання подібного програмного забезпечення може призводити до помилкових змін, які складно виявити та усунути на пізніших етапах розробки. Ще одним недоліком є ризик порушення логіки роботи програмної системи внаслідок автоматизованих або напіваавтоматизованих змін. Інструменти для модифікації вихідного коду не завжди здатні коректно враховувати всі контекстні зв'язки між модулями, особливо у випадку нестандартних рішень або неочевидних залежностей. У результаті навіть

формально коректні зміни можуть призводити до прихованих дефектів, які проявляються лише під час експлуатації або під значним навантаженням. Недоліком також є обмежена універсальність багатьох інструментів, які підтримують лише певні мови програмування, фреймворки або стилі кодування. Це знижує можливість їх застосування в різномірних проєктах і створює залежність від конкретних технологій або постачальників програмного забезпечення. Тому актуальною є розробка програмного забезпечення для модифікації вихідного коду.

Сформульовано функціональні вимоги до програмного забезпечення для модифікації вихідного коду.

2 МАТЕРІАЛИ І МЕТОДИ

2.1 Вибір мови програмування

При розробці програмного забезпечення для модифікації вихідного коду було використано мову програмування Java [18], [19].

Java характеризується високим рівнем стандартизації та передбачуваності поведінки, що має особливе значення під час розробки складних програмних систем і інструментальних засобів. Наявність чітко визначеної специфікації мови та віртуальної машини забезпечує однакове трактування коду в різних середовищах виконання, що знижує ризик виникнення неочікуваних помилок і спрощує тестування програмного забезпечення. Така стабільність сприяє довготривалому використанню розроблених рішень без критичної залежності від конкретної платформи або апаратної конфігурації [18], [19].

Важливою особливістю Java є її орієнтація на багатопотокове виконання, що реалізується на рівні самої мови та стандартної бібліотеки. Підтримка потоків, механізмів синхронізації та сучасних засобів паралельної обробки даних дозволяє ефективно працювати з ресурсоемними задачами, зокрема з аналізом і обробкою великих обсягів вихідного коду. Це відкриває можливості для підвищення продуктивності програмних інструментів за рахунок паралельного виконання операцій без суттєвого ускладнення архітектури [18], [19].

Регулярне оновлення платформи Java дозволяє використовувати функціональні підходи, лямбда-вирази та розширені засоби роботи з колекціями, не втрачаючи можливості підтримки раніше створених програмних продуктів. Це робить Java привабливою для проєктів, орієнтованих на тривалий життєвий цикл і поступове розширення функціональності [18], [19].

Значну роль відіграє й широка спільнота розробників, яка формує велику кількість навчальних матеріалів, прикладів реалізацій і відкритих програмних рішень. Активна підтримка мови з боку спільноти та корпоративних користувачів сприяє швидкому виявленню й усуненню недоліків, а також розвитку інструментів для налагодження, тестування та аналізу коду. У результаті знижується поріг входження для нових розробників і підвищується якість створюваного програмного забезпечення [18], [19].

Отже, мова програмування Java вирізняється поєднанням стабільності, масштабованості та технологічної зрілості, що дозволяє ефективно застосовувати її в різних сферах розробки програмного забезпечення. Її можливості виходять далеко за межі базового програмування, забезпечуючи надійну основу для створення складних, довготривалих і функціонально насичених програмних систем [18], [19].

Доцільність використання мови програмування Java для розробки програмного забезпечення для модифікації вихідного коду визначається поєднанням її технологічних особливостей, стабільності та широких можливостей для роботи зі складними програмними структурами. Java спочатку проєктувалася як мова для створення масштабних і надійних систем, що безпосередньо відповідає вимогам до інструментів, які виконують аналіз, перетворення та контроль змін у вихідному коді. Такі програмні продукти мають працювати з великими обсягами даних, складними ієрархіями та численними залежностями, що ефективно реалізується засобами об'єктоорієнтованої моделі Java [18], [19].

Важливою перевагою Java є її платформонезалежність, яка забезпечується використанням віртуальної машини. Це дозволяє розробленому програмному забезпеченню функціонувати на різних операційних системах без необхідності адаптації вихідного коду, що є особливо актуальним для інструментів модифікації, які можуть застосовуватися в різних середовищах розробки. Завдяки цьому спрощується розповсюдження програмного продукту та зменшуються витрати на його

супровід. Доцільність вибору Java також пов'язана з наявністю розвиненої стандартної бібліотеки та великої кількості перевірених сторонніх фреймворків. Засоби роботи з текстом, файловими системами, абстрактними синтаксичними деревами, регулярними виразами та потоками даних дають змогу ефективно реалізовувати механізми аналізу та трансформації вихідного коду. Крім того, існування численних бібліотек для статичного аналізу та парсингу спрощує створення складних функціональних можливостей без необхідності реалізації їх з нуля [18], [19].

Окрему роль відіграє висока надійність і безпека мови Java, що досягається завдяки строгій типізації, автоматичному керуванню пам'яттю та розвиненим механізмам обробки винятків. Це знижує ймовірність виникнення критичних помилок у програмному забезпеченні, яке безпосередньо впливає на структуру та коректність вихідного коду інших програмних систем. У контексті модифікації коду така стабільність є принципово важливою, оскільки помилки інструменту можуть мати масштабні наслідки [18], [19].

Обґрунтування вибору мови програмування для розробки програмного забезпечення для модифікації вихідного коду наведено у таблиці 2.1.

Таблиця 2.1 – Обґрунтування вибору мови програмування для розробки програмного забезпечення для модифікації вихідного коду

Критерій порівняння мов програмування	Мова програмування		
	Java	Python	C++
Придатність для великих кодових баз	+	+—	+
Наявність бібліотек для аналізу та трансформації коду	+	+	—
Строга типізація та контроль помилок	+	—	+
Безпека та керування пам'яттю	+	+	—

Підтримка багатопотоковості	+	+—	+
-----------------------------	---	----	---

Отже, для реалізації програмного забезпечення для модифікації вихідного коду обрано мову програмування Java завдяки її універсальності, зрілості екосистеми та здатності забезпечувати надійну роботу зі складними програмними структурами. Важливою перевагою Java є її платформонезалежність, яка забезпечується використанням віртуальної машини. Це дозволяє розробленому програмному забезпеченню функціонувати на різних операційних системах без необхідності адаптації вихідного коду, що є особливо актуальним для інструментів модифікації, які можуть застосовуватися в різних середовищах розробки.

2.2 Вибір середовища розробки для створення програмного забезпечення для модифікації вихідного коду

В якості середовища розробки для створення програмного забезпечення для модифікації вихідного коду було обрано середовище IntelliJ IDEA [20], [21].

Середовище розробки IntelliJ IDEA вирізняється високим рівнем стабільності та орієнтацією на професійне використання в довготривалих програмних проєктах. Архітектура середовища розробки побудована таким чином, щоб забезпечувати коректну роботу навіть із великими та складними кодовими базами, мінімізуючи затримки під час аналізу коду та виконання допоміжних операцій. Це має особливе значення при створенні інструментів, які працюють з вихідним кодом інших програмних систем і потребують постійного аналізу великої кількості файлів та залежностей [20], [21].

Варто також звернути увагу на модульність IntelliJ IDEA, яка реалізується через систему плагінів. Завдяки цьому середовище розробки може бути легко адаптоване під специфічні потреби конкретного проєкту або технологічного стеку. Розширення функціональності за рахунок плагінів

дозволяє інтегрувати додаткові засоби аналізу коду, автоматизації перевірок або взаємодії з іншими інструментами розробки, що підвищує гнучкість і універсальність середовища [20], [21].

Суттєвою характеристикою IntelliJ IDEA є орієнтація на підвищення продуктивності розробника. Продумані механізми автоматизації рутинних дій, підтримка шаблонів коду та швидкі навігаційні можливості зменшують кількість повторюваних операцій і дозволяють зосередитися на логіці програмного забезпечення. У результаті скорочується час розробки та знижується ймовірність механічних помилок, що особливо важливо при створенні складних інструментальних рішень [20], [21].

Окремо слід відзначити можливості IntelliJ IDEA щодо аналізу якості коду та дотримання стандартів програмування. Середовище розробки здатне автоматично виявляти потенційні проблеми, пов'язані з читабельністю, надмірною складністю або неефективними конструкціями. Це сприяє формуванню більш структурованого та підтримуваного коду, що є критично важливим для програмних продуктів, орієнтованих на подальший розвиток і модифікацію [20], [21].

Отже, IntelliJ IDEA є не лише інструментом для написання програмного коду, а комплексним середовищем, яке підтримує повний цикл розробки програмного забезпечення. Її функціональні можливості, гнучкість налаштувань і стабільність роботи створюють надійну основу для реалізації складних програмних проєктів, зокрема тих, що пов'язані з аналізом і модифікацією вихідного коду [20], [21].

Доцільність використання середовища розробки IntelliJ IDEA для створення програмного забезпечення для модифікації вихідного коду обумовлюється його орієнтацією на роботу зі складними програмними проєктами та глибоким розумінням структури програмного коду. Це середовище розробки не обмежується лише текстовим редагуванням, а виконує інтелектуальний аналіз коду, що є принципово важливим під час розробки інструментів, які самі призначені для аналізу та трансформації

вихідних текстів програм. Завдяки цьому забезпечується більш точне проєктування логіки модифікацій і знижується ризик помилок на етапі реалізації [20], [21].

Використання IntelliJ IDEA є доцільним також через її тісну інтеграцію з мовою програмування Java та екосистемою пов'язаних технологій. Середовище надає повноцінну підтримку сучасних можливостей мови, автоматично враховує особливості синтаксису та семантики, що спрощує створення складних механізмів обробки коду. Інтелектуальні підказки, перевірки коректності та миттєве виявлення потенційних проблем сприяють підвищенню якості програмного забезпечення ще на етапі розробки [20], [21].

Суттєвою перевагою IntelliJ IDEA є розвинені засоби навігації та рефакторингу, які дозволяють ефективно працювати з великими кодовими базами. Це особливо актуально для проєктів, пов'язаних із модифікацією вихідного коду, де необхідно постійно аналізувати взаємозв'язки між класами, методами та модулями. Середовище розробки забезпечує швидкий доступ до структури проєкту, історії змін і залежностей, що спрощує підтримку логічної цілісності програмного продукту [20], [21].

Додатковим аргументом на користь IntelliJ IDEA є наявність потужних засобів налагодження, тестування та інтеграції з системами контролю версій. Це дозволяє оперативно перевіряти коректність реалізованих механізмів модифікації коду та контролювати наслідки внесених змін. У контексті розробки інструментального програмного забезпечення така можливість є критично важливою, оскільки помилки можуть мати масштабний вплив на оброблювані програмні проєкти [20], [21].

Отже, використання середовища розробки IntelliJ IDEA є доцільним завдяки поєднанню інтелектуальних можливостей аналізу коду, глибокої підтримки Java та ефективних інструментів роботи з великими проєктами. Це середовище створює умови для підвищення якості, надійності та керованості програмного забезпечення, призначеного для модифікації вихідного коду, що робить його оптимальним вибором для реалізації подібних рішень [20], [21].

Обґрунтування вибору середовища розробки для створення програмного забезпечення для модифікації вихідного коду наведено у таблиці 2.2.

Таблиця 2.2 – Обґрунтування вибору середовища розробки для створення програмного забезпечення для модифікації вихідного коду

Критерій порівняння середовищ розробки	Середовища розробки		
	IntelliJ IDEA	Eclipse	NetBeans
Глибина аналізу вихідного коду	+	+—	+—
Інтелектуальні підказки та автодоповнення	+	+—	+—
Інтеграція з системами контролю версій	+	+	+—
Масштабованість середовища	+	+—	+
Можливості рефакторингу	+	+	+—

Таким чином, для створення програмного забезпечення для модифікації вихідного коду обрано середовище розробки IntelliJ IDEA через поєднання інтелектуальних можливостей аналізу коду, глибокої підтримки мови програмування Java та ефективних інструментів роботи з великими проєктами. Це середовище створює умови для підвищення якості, надійності та керованості програмного забезпечення, призначеного для модифікації вихідного коду, що робить його оптимальним вибором для реалізації подібних рішень. Крім того, суттєвою характеристикою IntelliJ IDEA є її орієнтація на підвищення продуктивності розробника. Продумані механізми автоматизації рутинних дій, підтримка шаблонів коду та швидкі навігаційні можливості

зменшують кількість повторюваних операцій і дозволяють зосередитися на логіці програмного забезпечення.

2.3 Висновки за розділом 2

Для реалізації програмного забезпечення для модифікації вихідного коду обрано мову програмування Java завдяки її універсальності, зрілості екосистеми та здатності забезпечувати надійну роботу зі складними програмними структурами. Важливою перевагою Java є її платформонезалежність, яка забезпечується використанням віртуальної машини. Це дозволяє розробленому програмному забезпеченню функціонувати на різних операційних системах без необхідності адаптації вихідного коду, що є особливо актуальним для інструментів модифікації, які можуть застосовуватися в різних середовищах розробки.

Для створення програмного забезпечення для модифікації вихідного коду обрано середовище розробки IntelliJ IDEA через поєднання інтелектуальних можливостей аналізу коду, глибокої підтримки мови програмування Java та ефективних інструментів роботи з великими проєктами. Це середовище створює умови для підвищення якості, надійності та керованості програмного забезпечення, призначеного для модифікації вихідного коду, що робить його оптимальним вибором для реалізації подібних рішень.

3 ОПИС ПРОГРАМИ

3.1 Структура програмного забезпечення для модифікації вихідного коду

Структуру програмного забезпечення для модифікації вихідного коду наведено на рис. 3.1.

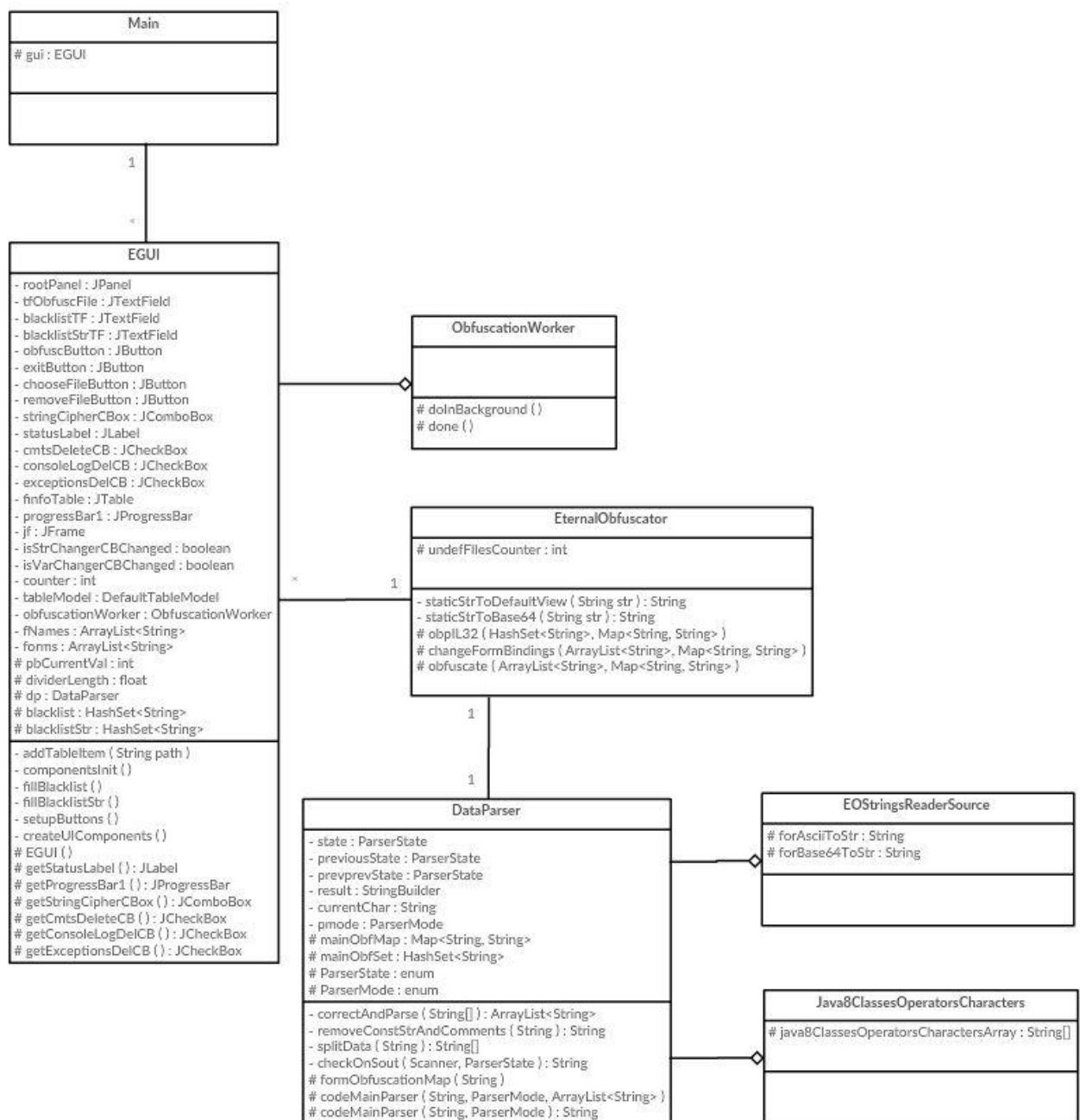


Рисунок 3.1 – Структура програмного забезпечення для модифікації вихідного коду

Структура програмного забезпечення для модифікації вихідного коду розроблена з урахуванням мінімізації надлишкових елементів і зосереджена на реалізації ключових механізмів заплутування програмного коду. Архітектура програмної системи відображена у вигляді діаграми класів, яка демонструє взаємодію основних програмних компонентів, відповідальних за аналіз, обробку та трансформацію вихідного коду. Такий підхід дозволяє чітко окреслити функціональні межі кожного класу та забезпечити узгоджену роботу всієї системи без залучення допоміжних або другорядних модулів.

Центральним елементом програмного продукту виступає головний клас `Main`, через який ініціюється запуск системи та здійснюється взаємодія з користувачем за допомогою графічного інтерфейсу. Саме з цього класу відбувається створення окремого потоку виконання, представленого класом `ObfuscationWorker`, що дозволяє відокремити виконання основного алгоритму модифікації коду від інтерфейсної частини. Така організація забезпечує стабільність роботи програми та запобігає блокуванню користувацького інтерфейсу під час обробки великих обсягів вихідного коду.

Логіка безпосереднього виконання заплутуваних перетворень зосереджена у класі `EternalObfuscator`, який реалізує основні алгоритми обфускації та координує їх застосування до різних елементів програмного коду. Цей клас взаємодіє з модулем аналізу, представленим класом `DataParser`, відповідальним за дослідження структури вихідного коду та визначення доцільності застосування обфускації до конкретних класів, методів або полів. Аналіз виконується з урахуванням синтаксичних і семантичних особливостей мови програмування Java, що дозволяє зберігати коректність модифікованого коду.

Допоміжну, але важливу роль у структурі програмного забезпечення відіграють два незалежні класи, призначені для підтримки процесу аналізу та трансформації. Клас `EOStringsReaderSource` забезпечує формування та використання механізмів розшифрування константних рядків, що є одним із поширених прийомів ускладнення аналізу програмного коду. Водночас клас

Java8ClassesOperatorsCharacters містить повний набір символів, ключових слів та базових конструкцій мови Java Standard Edition восьмої версії, що дозволяє парсеру швидко ідентифікувати структурні елементи коду, необхідні для подальшої обфускації.

Узгоджена взаємодія зазначених класів формує цілісну структуру програмного продукту, орієнтовану на ефективну модифікацію вихідного коду. Представлення цієї структури у вигляді діаграми класів дає змогу наочно продемонструвати логічні зв'язки між компонентами системи та підкреслити їхню роль у реалізації основного алгоритму модифікації коду, забезпечуючи зрозумілість і керованість програмної архітектури.

3.2 Функціонування програмного забезпечення для модифікації вихідного коду

Функціонування програмного забезпечення для модифікації вихідного коду подамо за допомогою схеми, зображеної на рис. 3.2.

Функціонування програмного забезпечення для модифікації вихідного коду реалізується як послідовний процес взаємодії користувача з системою та автоматизованої обробки програмних файлів відповідно до обраних параметрів. Робота програми розпочинається з формування вхідних даних, коли через графічний інтерфейс здійснюється підключення текстових файлів, що містять вихідний код, а також файлів опису інтерфейсу. На цьому етапі програмне забезпечення лише фіксує набір ресурсів, з якими надалі буде виконуватися модифікація, не втручаючись у їх внутрішню структуру.

Після завантаження необхідних файлів користувач визначає параметри майбутньої модифікації шляхом обфускації за допомогою відповідних налаштувань. Задається спосіб перетворення константних рядків, якщо така функціональність потрібна, а також перелік винятків, для яких шифрування не повинно застосовуватися. Окремо враховується можливість збереження незмінними певних ідентифікаторів, що є важливим для підтримки

працездатності програмного продукту. Крім того, користувач може дозволити або заборонити усунення коментарів, інструкцій виведення інформації в консоль та конструкцій, пов'язаних з обробкою виключень. Таким чином формується індивідуальна стратегія модифікації вихідного коду.

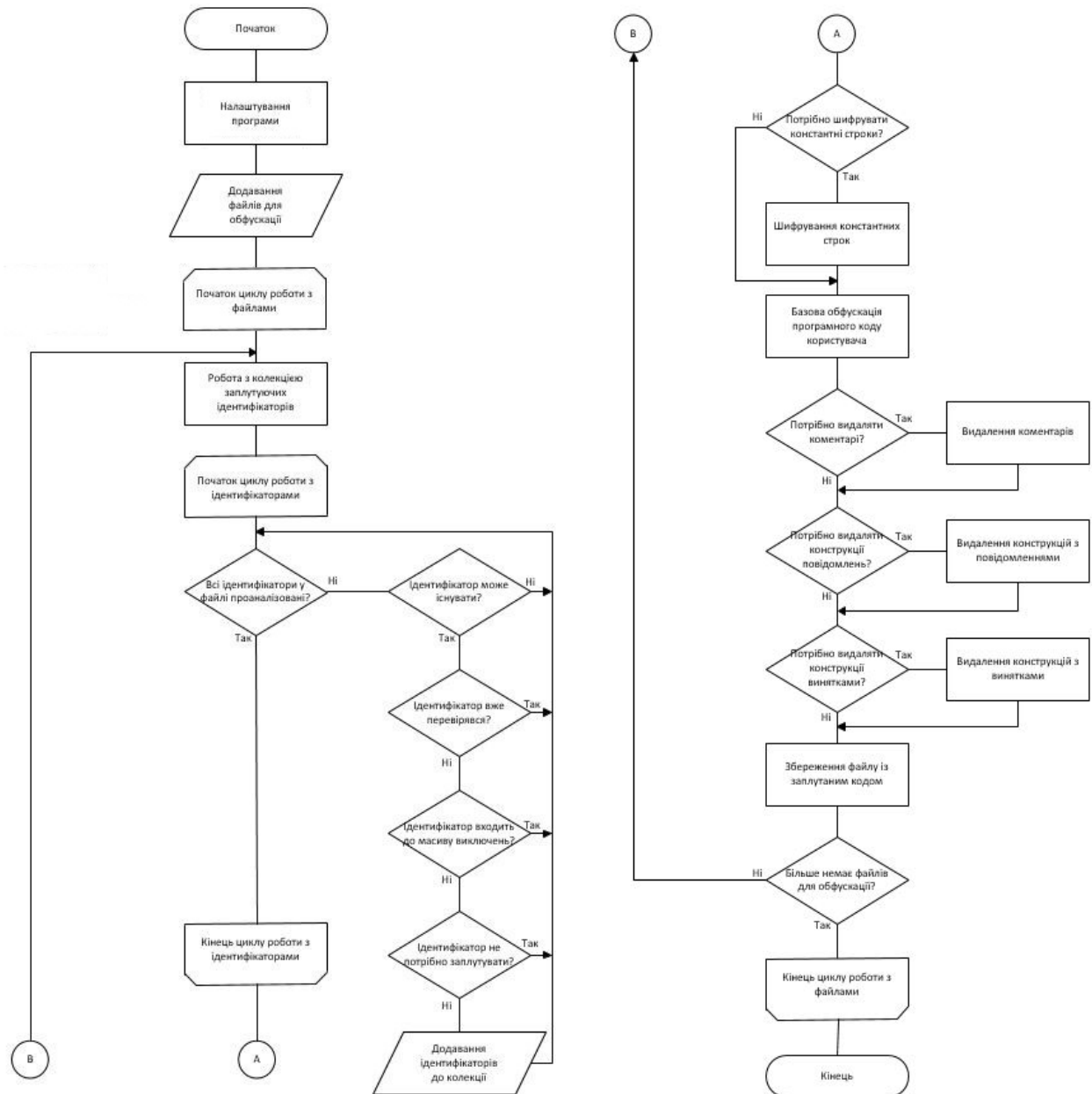


Рисунок 3.2 – Схема функціонування програмного забезпечення для модифікації вихідного коду

Безпосередній процес модифікації шляхом обфускації запускається після підтвердження дій у графічному інтерфейсі. На цьому етапі активується

механізм аналізу, реалізований у класі `DataParser`, який здійснює поглиблене опрацювання вихідного коду. У результаті такого аналізу формується спеціальна структура відповідностей між початковими ідентифікаторами та їх модифікованими аналогами. Згенеровані назви мають фіксовану довжину та складаються з випадкових комбінацій візуально схожих символів, що значно ускладнює сприйняття логіки програми та проведення зворотного аналізу.

Після завершення етапу аналізу керування процесом переходить до класу `EternalObfuscator`, який виконує поетапну обробку кожного файлу вихідного коду. У ході цієї обробки всі знайдені ідентифікатори замінюються відповідними значеннями з попередньо сформованої колекції. У такий спосіб відбувається базова модифікація структури програми, що змінює імена класів, методів і змінних без порушення загальної логіки виконання.

Подальше функціонування програмного забезпечення залежить від обраних користувачем додаткових параметрів. Якщо передбачено обфускацію константних рядків, система виконує їх пошук у коді та перетворення відповідно до заданого алгоритму, зокрема шляхом представлення у вигляді числових кодів або закодованих послідовностей. Після цього, за умови підтвердження відповідних налаштувань, з вихідного коду усуваються коментарі, інструкції для виведення службових повідомлень та конструкції обробки виключень, що додатково зменшує інформативність коду для стороннього аналізу.

Завершальним етапом роботи програмного забезпечення є формування нових файлів, які містять модифікований варіант вихідного коду. Ці файли створюються поруч з оригінальними та можуть бути безпосередньо використані для заміни початкових версій у складі програмного проєкту. Таким чином забезпечується збереження вихідних даних і водночас надається готовий результат обфускації, придатний для подальшого використання або розгортання.

Таким чином, функціонування програмного забезпечення для модифікації вихідного коду реалізує послідовний і логічно впорядкований

процес обробки програмних файлів, починаючи з їх аналізу та завершуючи формуванням модифікованих версій. Запропонований механізм дозволяє зберігати коректність програмного коду, водночас істотно ускладнюючи його подальший аналіз сторонніми особами, що підтверджує ефективність обраного підходу. Реалізована модель функціонування програмного продукту забезпечує гнучке керування процесом модифікації завдяки системі налаштувань і дозволяє адаптувати алгоритми модифікації до конкретних потреб користувача. Це свідчить про практичну придатність програмного забезпечення для застосування у реальних програмних проєктах.

3.3 Особливості реалізації програмного забезпечення для модифікації вихідного коду

Особливості реалізації програмного забезпечення для модифікації вихідного коду визначаються необхідністю поєднання стабільної роботи користувацького інтерфейсу з ресурсоемними алгоритмами аналізу та трансформації програмних текстів. Саме тому виконання перетворень організовано в окремому потоці `ObfuscationWorker`, який функціонує незалежно від головного потоку застосунку. Такий підхід дозволяє уникнути блокування графічного інтерфейсу під час обробки файлів і забезпечує коректну взаємодію користувача з програмою навіть у разі тривалих операцій над великими проєктами.

Реалізація алгоритмів модифікації шляхом обфускації побудована таким чином, що кожен файл, обраний користувачем, обробляється автономно. Центральну роль у цьому процесі відіграє клас `EternalObfuscator`, у якому зосереджено логіку основної функції модифікації коду. Для кожного окремого документа запускається повний цикл аналізу та перетворень, що унеможливорює некоректний вплив змін одного файлу на інші елементи проєкту. Така ізольована обробка підвищує надійність програмного забезпечення та спрощує контроль результатів.

Початковим етапом реалізації заплутування є формування набору ідентифікаторів, які можуть бути змінені без порушення синтаксису та логіки програми. Ця задача покладена на клас `DataParser`, який за допомогою спеціалізованої функції здійснює попередню підготовку тексту вихідного коду. Документ розбивається на окремі лексичні одиниці, при цьому ігноруються пропуски та службові розділювачі. Подальша перевірка кожного потенційного ідентифікатора виконується з урахуванням правил мови Java, що регламентують допустимі символи на початку та в середині назв, а також з урахуванням переліку зарезервованих слів і стандартних класів. Для цього використовується внутрішній набір мовних конструкцій, який дозволяє швидко відсіяти елементи, що не підлягають модифікації.

Важливою особливістю реалізації є врахування користувацьких обмежень. Якщо певні ідентифікатори не повинні змінюватися, наприклад через їх належність до сторонніх бібліотек, система перевіряє відповідні дозволи ще до додавання таких назв у внутрішню колекцію. Крім того, кожен ідентифікатор перевіряється на унікальність у межах уже сформованого набору, що запобігає дублюванню та зайвим операціям. Лише після проходження всіх перевірок ідентифікатор визнається придатним до подальшого заплутування.

За наявності активованої функції шифрування константних рядків програмне забезпечення переходить до окремого етапу обробки текстових літералів. Для забезпечення коректності аналізу з коду тимчасово усуваються коментарі, що дозволяє уникнути помилкового розпізнавання рядків, які фактично не беруть участі у виконанні програми. Після цього виконується пошук фрагментів тексту, обмежених подвійними лапками, які ідентифікуються як константні строки. Система враховує заздалегідь задані винятки та формує окрему колекцію лише з тих значень, які дозволено шифрувати. Подальша заміна відбувається відповідно до обраного користувачем алгоритму, що реалізується через спеціалізовані методи у класі `EternalObfuscator`.

Наступним ключовим аспектом реалізації є заміна ідентифікаторів на їх заплутані аналоги. Для цього застосовується генератор випадкових послідовностей фіксованої довжини, сформованих з візуально подібних символів. Такий підхід значно ускладнює сприйняття структури програми та створює додаткові труднощі для аналізу коду методами зворотної розробки. Реалізація цього механізму інтегрована безпосередньо в основний алгоритм модифікації та застосовується до всіх елементів, які були попередньо включені до відповідної колекції.

Після завершення базових перетворень програмне забезпечення переходить до виконання додаткових операцій, якщо вони були дозволені користувачем. Реалізація цих функцій передбачає повне очищення коду від коментарів із урахуванням можливих крайових випадків, усунення інструкцій для виведення інформації в консоль, а також радикальне спрощення конструкцій обробки виключень шляхом вилучення допоміжних блоків і ключових слів. Усі ці дії виконуються обережно, щоб не порушити синтаксичну цілісність програмного тексту.

Завершальним етапом реалізації є збереження результатів модифікації. Для цього створюється окремий файл у спеціально визначеній директорії, де розміщується повністю модифікований варіант вихідного коду. Назва нового документа відповідає оригінальному файлу, що полегшує його ідентифікацію та подальше використання у складі проєкту. Таким чином реалізація програмного забезпечення забезпечує повний цикл модифікації коду — від аналізу до формування готового результату — з урахуванням вимог до надійності, гнучкості та зручності використання.

3.4 Опис організації зберігання даних програмного забезпечення для модифікації вихідного коду

У програмному забезпеченні для модифікації вихідного коду організація зберігання даних відіграє допоміжну, але важливу роль, оскільки

забезпечує коректну обробку програмних файлів, тимчасових структур та результатів виконання алгоритмів модифікації. Особливістю такого програмного продукту є відсутність потреби у використанні повноцінної системи керування базами даних, оскільки обробка здійснюється над файлами вихідного коду та проміжними представленнями, які існують переважно протягом часу виконання програми.

Схему процесу організації зберігання та обробки даних у програмному забезпеченні для модифікації вихідного коду наведено на рис. 3.3.

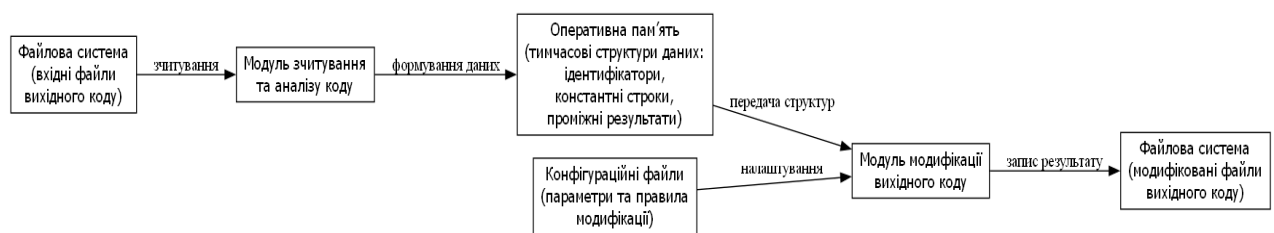


Рисунок 3.3 – Організація зберігання та обробки даних у програмному забезпеченні для модифікації вихідного коду

Основним джерелом даних для програмного забезпечення є файли вихідного коду, що зберігаються у файловій системі. Під час роботи програми ці файли зчитуються, аналізуються та перетворюються у внутрішні структури даних, які використовуються для подальшої модифікації. Зберігання таких даних в оперативній пам'яті дозволяє забезпечити високу швидкість обробки, мінімізувати затримки доступу та спростити реалізацію алгоритмів трансформації коду.

У процесі функціонування програмного забезпечення активно використовуються тимчасові структури даних, призначені для збереження абстрактних синтаксичних представлень, таблиць ідентифікаторів, наборів правил модифікації та проміжних результатів перетворень. Ці структури створюються динамічно, використовуються лише в межах одного сеансу роботи та знищуються після завершення виконання відповідних операцій, що виключає необхідність їх довготривалого збереження.

Для збереження результатів роботи програмного забезпечення використовується файлове сховище. Модифікований вихідний код записується у нові файли або замінює початкові залежно від обраного режиму роботи. Такий підхід забезпечує прозорість процесу модифікації, дозволяє легко контролювати зміни та інтегрувати програмне забезпечення у наявні процеси розробки без потреби у додаткових інструментах зберігання даних.

Налаштування програмного забезпечення, параметри запуску та обрані режими модифікації зберігаються у конфігураційних файлах. Використання текстових форматів для таких файлів спрощує їх редагування, підвищує гнучкість налаштування та дозволяє легко переносити конфігурацію між різними середовищами виконання. При цьому конфігураційні дані зчитуються під час запуску програмного забезпечення та використовуються для формування логіки подальшої обробки.

Обрана організація зберігання даних дозволяє зменшити складність програмного забезпечення, уникнути залежності від зовнішніх систем зберігання та забезпечити високу надійність роботи. Такий підхід є доцільним для програмних засобів, орієнтованих на модифікацію вихідного коду, оскільки відповідає їх функціональному призначенню та вимогам до продуктивності, масштабованості та простоти інтеграції.

3.5 Проєктування інтерфейсу програмного забезпечення для модифікації вихідного коду

Проєктування інтерфейсу взаємодії користувача з програмним забезпеченням для модифікації вихідного коду виконано з урахуванням вимог технічного завдання. При розробленні програмного забезпечення для модифікації вихідного коду враховані функціональні вимоги до програми:

- підтримка можливості завантаження та обробки файлів вихідного коду програмних проєктів з файлової системи користувача;

- підтримка можливості аналізу синтаксичної структури вихідного коду з метою коректного виявлення ідентифікаторів, константних рядків та інших елементів, придатних до модифікації;
- забезпечення можливості налаштування параметрів модифікації вихідного коду відповідно до вимог користувача через графічний інтерфейс;
- підтримка можливості обфускації назв класів, методів та змінних без порушення логіки виконання програми;
- підтримка можливості вибіркового застосування додаткових методів модифікації, зокрема видалення коментарів, консольних повідомлень та конструкцій обробки винятків;
- забезпечення стабільної роботи графічного інтерфейсу під час виконання ресурсоємних операцій шляхом використання окремих потоків виконання;
- підтримка можливості формування окремих файлів з модифікованим вихідним кодом без перезапису оригінальних файлів;
- забезпечення інформування користувача про стан виконання процесу модифікації та результати обробки файлів;
- підтримка можливості подальшого використання згенерованих файлів у складі нового або існуючого програмного проєкту.

Проєктування інтерфейсу програмного забезпечення для модифікації вихідного коду здійснювалося з орієнтацією на зручність сприйняття та мінімізацію складності взаємодії з програмою. Візуальне оформлення побудоване у нейтральній світлій стилістиці з використанням читабельних шрифтів, що сприяє комфортній роботі протягом тривалого часу. Основна ідея інтерфейсу полягає у зосередженні уваги користувача лише на ключових елементах керування, без перевантаження екрану другорядною інформацією, що дозволяє швидко зорієнтуватися у логіці роботи програмного продукту.

Головне вікно інтерфейсу призначене для керування файлами, які підлягають модифікації. Додавання вихідних файлів реалізоване через стандартні механізми операційної системи, що робить цей процес звичним та

зрозумілим. Інформація про вибрані файли автоматично відображається у табличній формі, де легко контролюється їхній склад та за потреби здійснюється видалення непотрібних елементів. Для зручності відображається актуальний файл, з яким у даний момент асоціюється процес обробки, при цьому пряме редагування цього поля обмежене, що запобігає помилковим діям. У нижній частині інтерфейсу розміщено індикатор виконання, який динамічно відображає стан обробки та надає користувачу наочне уявлення про хід модифікації.

Інтерфейс головної форми програмного забезпечення для модифікації вихідного коду наведено на рис. 3.4.

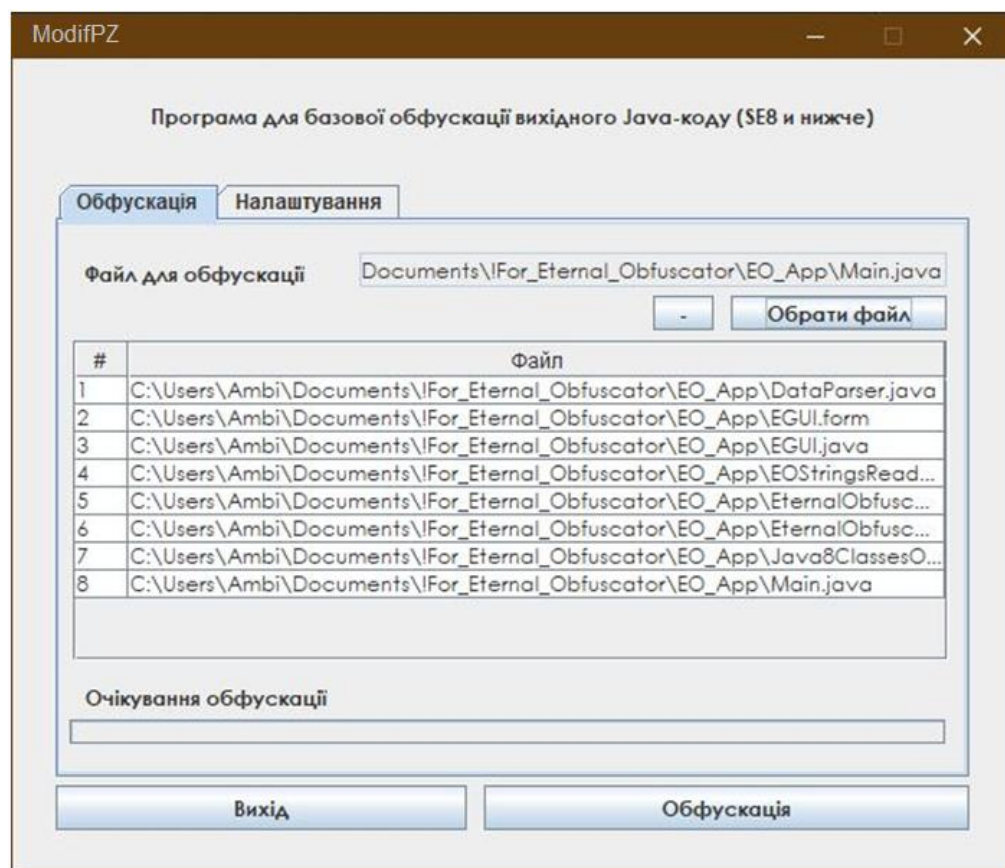


Рисунок 3.4 – Інтерфейс головної форми програмного забезпечення для модифікації вихідного коду

Окремий розділ інтерфейсу відведено для налаштування параметрів перетворення вихідного коду. Тут реалізовано логічно згруповані елементи

керування, що дозволяють визначати спосіб обробки текстових констант, задавати виключення для окремих рядків та ідентифікаторів, а також керувати застосуванням додаткових методів ускладнення коду. Вибір варіантів супроводжується прикладами та підказками, які полегшують прийняття рішень і зменшують ймовірність помилкових налаштувань. Доповненням до основних елементів інтерфейсу є спливаючі повідомлення та допоміжні підказки, що спрямовують користувача у процесі підготовки та запуску модифікації. Такий підхід до проектування інтерфейсу забезпечує поєднання простоти використання з достатньою функціональною гнучкістю програмного забезпечення.

Інтерфейс вкладки Налаштування програмного забезпечення для модифікації вихідного коду наведено на рис. 3.5.

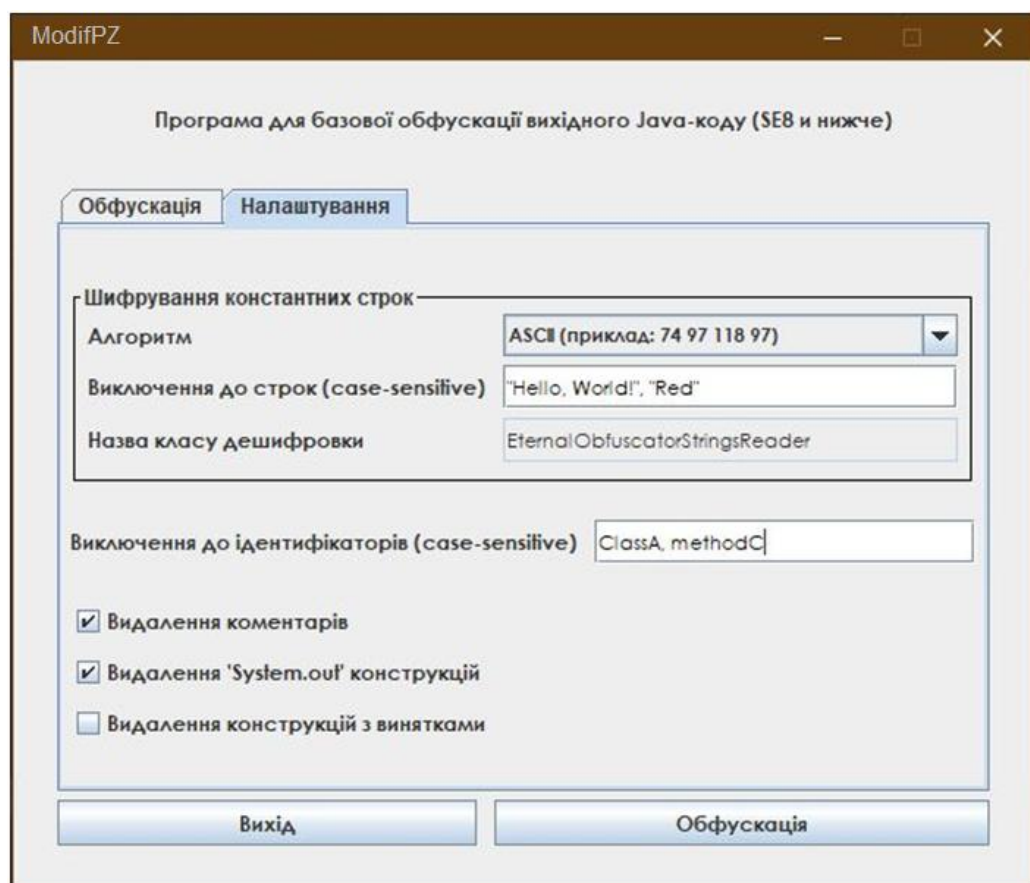


Рисунок 3.5 – Інтерфейс вкладки Налаштування програмного забезпечення для модифікації вихідного коду

3.6 Висновки за розділом 3

Розроблено програмне забезпечення для модифікації вихідного коду.

Структура програмного забезпечення для модифікації вихідного коду розроблена з урахуванням мінімізації надлишкових елементів і зосереджена на реалізації ключових механізмів заплутування програмного коду. Архітектура програмної системи відображена у вигляді діаграми класів, яка демонструє взаємодію основних програмних компонентів, відповідальних за аналіз, обробку та трансформацію вихідного коду. Такий підхід дозволяє чітко окреслити функціональні межі кожного класу та забезпечити узгоджену роботу всієї системи без залучення допоміжних або другорядних модулів.

Описано функціонування програмного забезпечення для модифікації вихідного коду, що реалізує послідовний і логічно впорядкований процес обробки програмних файлів, починаючи з їх аналізу та завершуючи формуванням модифікованих версій. Запропонований механізм дозволяє зберігати коректність програмного коду, водночас істотно ускладнюючи його подальший аналіз сторонніми особами, що підтверджує ефективність обраного підходу. Реалізована модель функціонування програмного продукту забезпечує гнучке керування процесом модифікації завдяки системі налаштувань і дозволяє адаптувати алгоритми модифікації до конкретних потреб користувача. Це свідчить про практичну придатність програмного забезпечення для застосування у реальних програмних проєктах.

Описано особливості реалізації програмного забезпечення для модифікації вихідного коду.

Описано організацію зберігання та обробки даних у програмному забезпеченні для модифікації вихідного коду. Визначено, що у програмному забезпеченні для модифікації вихідного коду організація зберігання даних відіграє допоміжну, але важливу роль, оскільки забезпечує коректну обробку програмних файлів, тимчасових структур та результатів виконання алгоритмів модифікації. Особливістю такого програмного продукту є відсутність потреби

у використанні повноцінної системи керування базами даних, оскільки обробка здійснюється над файлами вихідного коду та проміжними представленнями, які існують переважно протягом часу виконання програми. Обраний підхід до організації зберігання даних дозволяє зменшити складність програмного забезпечення, уникнути залежності від зовнішніх систем зберігання та забезпечити високу надійність роботи.

Виконано проєктування інтерфейсу взаємодії користувача з програмним забезпеченням для модифікації вихідного коду.

4 ЕКСПЛУАТАЦІЯ, ТЕСТУВАННЯ ТА ЕКСПЕРИМЕНТАЛЬНЕ ДОСЛІДЖЕННЯ ПРОГРАМИ

4.1 Призначення й умови застосування програми

Програма призначена для модифікації вихідного коду. Програма написана на мові Java завдяки її універсальності, зрілості екосистеми та здатності забезпечувати надійну роботу зі складними програмними структурами. В якості середовища розробки для створення програмного забезпечення для модифікації вихідного коду обрано IntelliJ IDEA через поєднання інтелектуальних можливостей аналізу коду, глибокої підтримки мови програмування Java та ефективних інструментів роботи з великими проєктами.

Програмне забезпечення для модифікації вихідного коду забезпечує виконання таких функцій:

- підтримка можливості завантаження та обробки файлів вихідного коду програмних проєктів з файлової системи користувача;
- підтримка можливості аналізу синтаксичної структури вихідного коду з метою коректного виявлення ідентифікаторів, константних рядків та інших елементів, придатних до модифікації;
- забезпечення можливості налаштування параметрів модифікації вихідного коду відповідно до вимог користувача через графічний інтерфейс;
- підтримка можливості обфускації назв класів, методів та змінних без порушення логіки виконання програми;
- підтримка можливості вибіркового застосування додаткових методів модифікації, зокрема видалення коментарів, консольних повідомлень та конструкцій обробки винятків;
- забезпечення стабільної роботи графічного інтерфейсу під час виконання ресурсоємних операцій шляхом використання окремих потоків виконання;

- підтримка можливості формування окремих файлів з модифікованим вихідним кодом без перезапису оригінальних файлів;
- забезпечення інформування користувача про стан виконання процесу модифікації та результати обробки файлів;
- підтримка можливості подальшого використання згенерованих файлів у складі нового або існуючого програмного проєкту.

Для роботи програмного забезпечення для модифікації вихідного коду потрібні такі технічні та програмні ресурси: апаратна частина обчислювальної системи повинна забезпечувати стабільне виконання операцій аналізу та перетворення текстових файлів, а також коректну роботу графічного інтерфейсу користувача. Процесор комп'ютера має підтримувати багатопотокову обробку та мати тактову частоту не нижче середнього рівня (2ГГц), достатнього для виконання операцій синтаксичного аналізу та масових текстових заміन без суттєвих затримок. Обсяг оперативної пам'яті повинен бути не меншим за мінімально необхідний (8Гб) для одночасного зберігання вихідних файлів, проміжних структур даних та модифікованих результатів, що забезпечує безперервну роботу програми без перевантаження системи. Для зберігання даних необхідна наявність вільного дискового простору (32Гб), достатнього для розміщення оригінальних файлів проєкту, тимчасових даних та згенерованих модифікованих копій вихідного коду.

Програмне забезпечення має функціонувати під керуванням операційної системи, що підтримує виконання Java-застосунків та роботу з файловою системою на рівні користувача. Обов'язковою є наявність встановленого середовища виконання Java відповідної версії, яке забезпечує коректну роботу всіх модулів програми, зокрема парсера, модуля модифікації та графічного інтерфейсу. Для запуску та використання програмного продукту також необхідні стандартні системні бібліотеки, що входять до складу операційної системи, а також засоби відображення графічного інтерфейсу. Зазначені вимоги дозволяють забезпечити стабільне, продуктивне та коректне

функціонування програмного забезпечення для модифікації вихідного коду в типовому робочому середовищі користувача.

Функціональні характеристики розробленого програмного забезпечення для модифікації вихідного коду наведено у технічному завданні (додаток А). Фрагмент тексту програмного забезпечення для модифікації вихідного коду наведено у додатку Б.

4.2 Характеристики програми для модифікації вихідного коду

Програмне забезпечення для модифікації вихідного коду характеризується орієнтацією на автоматизовану обробку програмних текстів із збереженням їхньої функціональної коректності. Його робота ґрунтується на внутрішньому аналізі структури коду, що дозволяє коректно ідентифікувати елементи мови програмування та виконувати над ними цілеспрямовані перетворення без порушення логіки виконання програми. Архітектура такого програмного продукту побудована з урахуванням розмежування відповідальностей між модулями, завдяки чому забезпечується стабільність, передбачуваність результатів та можливість розширення функціональних можливостей у майбутньому.

Важливою характеристикою є гнучкість налаштувань, яка дозволяє адаптувати процес модифікації до різних вимог користувача та особливостей конкретного програмного проєкту. Програмне забезпечення забезпечує контроль над глибиною та типом перетворень, що дає змогу балансувати між рівнем ускладнення коду та зручністю його подальшого використання. Окрему увагу приділено стабільності виконання, що досягається завдяки ізольованому виконанню обчислювально інтенсивних операцій та коректній роботі з файловою системою. Це забезпечує безпечну обробку вихідних файлів і збереження результатів у формі незалежних копій.

До ключових характеристик також належить універсальність застосування, оскільки програмний продукт може використовуватися для

різних типів проєктів, що мають схожу синтаксичну структуру вихідного коду. Підтримка графічного інтерфейсу підвищує зручність взаємодії з програмою та зменшує поріг входження для користувачів з різним рівнем підготовки. Сукупність зазначених характеристик визначає програмне забезпечення для модифікації вихідного коду як інструмент, що поєднує ефективність автоматизованих перетворень із надійністю та зручністю практичного використання.

4.3 Інструкція по експлуатації програми

4.3.1 Звернення до програми

Для запуску програмного забезпечення для модифікації вихідного коду необхідно запустити відповідний файл.

4.3.2 Вхідні й вихідні дані

Вхідними даними до програмного забезпечення для модифікації вихідного коду є файли вихідного коду програм, які користувач бажає модифікувати, а також додаткові параметри модифікації, включно з правилами обробки ідентифікаторів, шифруванням константних рядків та іншими налаштуваннями користувача.

Вихідними даними програмного забезпечення для модифікації вихідного коду є нові файли, що містять обфусковані версії вихідного коду, а також тимчасові або проміжні структури, які можуть зберігатися для контролю процесу обробки.

4.3.3 Повідомлення

Користувач програмного забезпечення для модифікації вихідного коду може отримати такі повідомлення: повідомлення про успішне завантаження та

аналіз вихідних файлів, про хід виконання процесу модифікації, про виявлені помилки або конфлікти під час обробки файлів, а також про завершення обробки та збереження результатів у відповідних файлах.

4.4 Виконання програмного забезпечення для модифікації вихідного коду

Експлуатація програмного забезпечення для модифікації вихідного коду розпочинається з вибору файлів програмного проекту, які підлягають обробці (рис. 4.1). Для цього користувачу необхідно натиснути кнопку Обрати файл та вибрати файл або множину файлів у діалоговому вікні, що відкрилося.

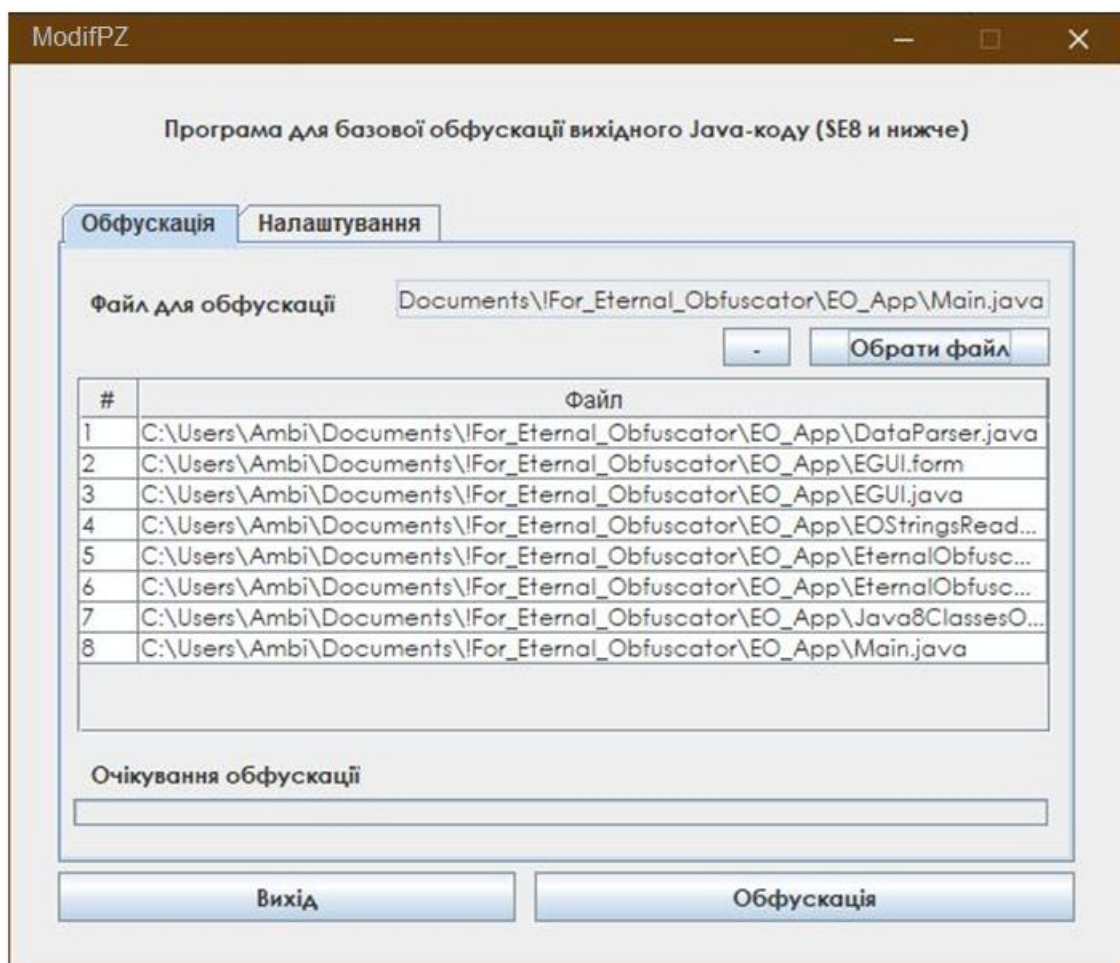


Рисунок 4.1 – Інтерфейс головної форми програмного забезпечення для модифікації вихідного коду

Після цього користувач переходить до етапу налаштування параметрів модифікації (рис. 4.2). На цьому етапі визначаються правила перетворення вихідного коду, зокрема особливості обробки ідентифікаторів, текстових констант та додаткових елементів, які можуть впливати на рівень заплутування. Усі параметри задаються безпосередньо через графічний інтерфейс, що дозволяє гнучко адаптувати роботу програми під конкретні вимоги та особливості обраного програмного проєкту.

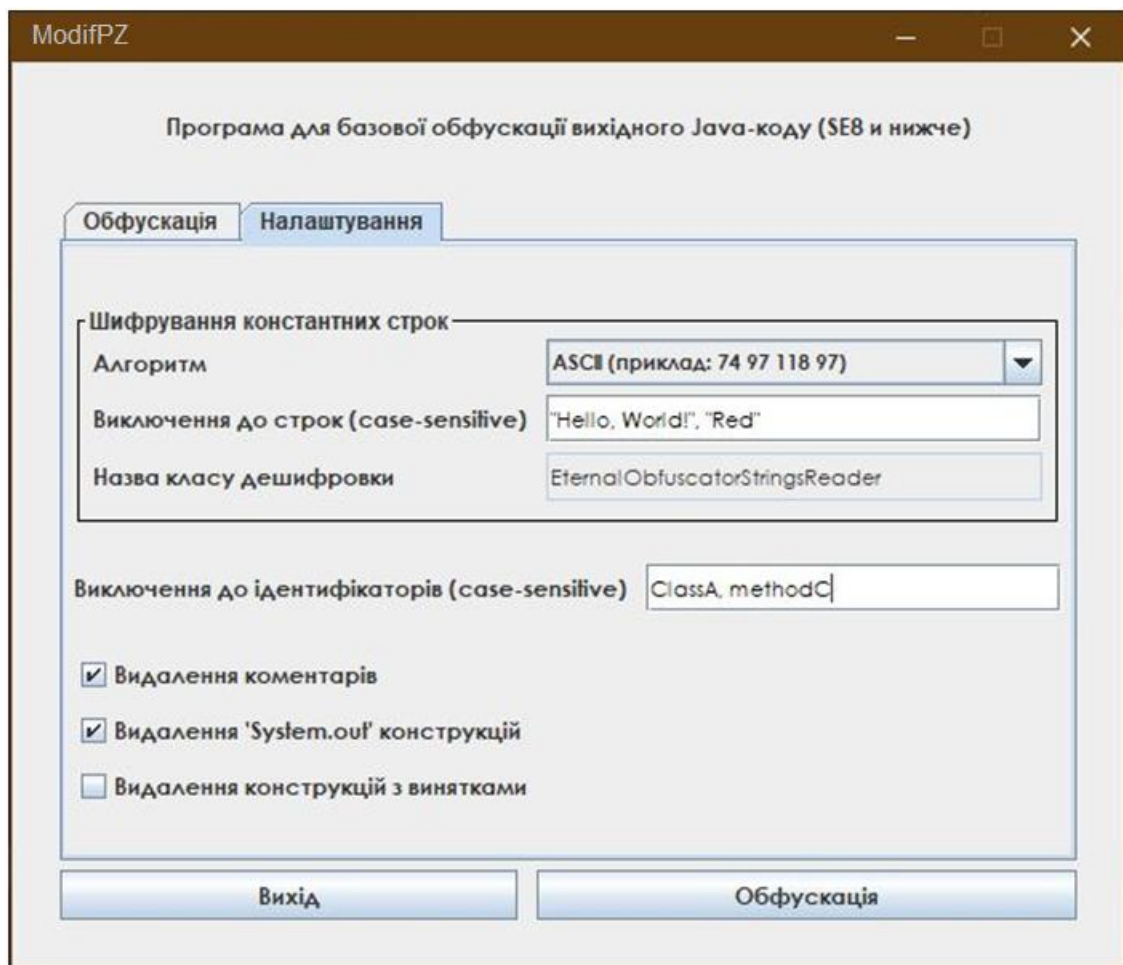


Рисунок 4.2 – Вкладка Налаштування

Після завершення налаштування запускається процес модифікації шляхом натискання відповідної кнопки, під час якого користувач має можливість спостерігати за перебігом виконання операцій (рис. 4.3). По завершенню обробки програмне забезпечення формує окремі файли з модифікованим вихідним кодом та розміщує їх у спеціально визначеній

директорії. Отримані результати можуть бути використані для заміни початкових файлів або для створення окремого проєкту, що завершує цикл використання програмного забезпечення для модифікації вихідного коду.



Рисунок 4.3 – Перебіг процесу модифікації програмного коду

4.5 Тестування програмного забезпечення для модифікації вихідного коду

Виконано тестування програмного забезпечення для модифікації вихідного коду.

Виявлено, що програма для модифікації вихідного коду функціонує правильно та злагоджено. Вона реалізує всі функціональні вимоги та успішно виконує свою основну задачу модифікації вихідного коду.

Розроблене програмне забезпечення дозволяє забезпечувати автоматизацію процесів, пов'язаних з підтримкою процесу модифікації вихідного коду.

4.6 Висновки за розділом 4

Описано програмне забезпечення для модифікації вихідного коду. Виконано тестування розробленого програмного забезпечення для модифікації вихідного коду. Результати тестування програмного забезпечення показали, що розроблена програма дозволяє забезпечити автоматизацію процесів, пов'язаних з підтримкою процесу модифікації вихідного коду.

ВИСНОВКИ

В ході виконання дипломної кваліфікаційної роботи бакалавра було проаналізовано та досліджено процес розробки програмного забезпечення для модифікації вихідного коду.

Визначено, що модифікація вихідного коду являє собою процес цілеспрямованої зміни програмного тексту з метою поліпшення його властивостей без порушення загальної логіки роботи програмного продукту. У межах цього процесу вихідний код може трансформуватися шляхом перейменування ідентифікаторів, перетворення структур, оптимізації окремих фрагментів або застосування спеціальних технік, спрямованих на підвищення стійкості програмного забезпечення до аналізу та несанкціонованого використання.

За результатами проведеного аналізу зроблено висновок, що у наш час існує досить багато програмних засобів для модифікації вихідного коду. Проте деякі програми для модифікації вихідного коду можуть бути орієнтовані на роботу з великими та структурно складними кодовими базами, що вимагає глибокого розуміння архітектури програмного продукту, мов програмування та принципів побудови програмних систем. За відсутності належних знань або досвіду використання подібного програмного забезпечення може призводити до помилкових змін, які складно виявити та усунути на пізніших етапах розробки. Ще одним недоліком є ризик порушення логіки роботи програмної системи внаслідок автоматизованих або напіваавтоматизованих змін. Інструменти для модифікації вихідного коду не завжди здатні коректно враховувати всі контекстні зв'язки між модулями, особливо у випадку нестандартних рішень або неочевидних залежностей. У результаті навіть формально коректні зміни можуть призводити до прихованих дефектів, які проявляються лише під час експлуатації або під значним навантаженням. Недоліком також є обмежена універсальність багатьох інструментів, які

підтримують лише певні мови програмування, фреймворки або стилі кодування. Це знижує можливість їх застосування в різномірних проєктах і створює залежність від конкретних технологій або постачальників програмного забезпечення. Тому актуальною є розробка програмного забезпечення для модифікації вихідного коду.

Сформульовано функціональні вимоги до програмного забезпечення для модифікації вихідного коду.

Для реалізації програмного забезпечення для модифікації вихідного коду обрано мову програмування Java завдяки її універсальності, зрілості екосистеми та здатності забезпечувати надійну роботу зі складними програмними структурами. Важливою перевагою Java є її платформонезалежність, яка забезпечується використанням віртуальної машини. Це дозволяє розробленому програмному забезпеченню функціонувати на різних операційних системах без необхідності адаптації вихідного коду, що є особливо актуальним для інструментів модифікації, які можуть застосовуватися в різних середовищах розробки.

Для створення програмного забезпечення для модифікації вихідного коду обрано середовище розробки IntelliJ IDEA через поєднання інтелектуальних можливостей аналізу коду, глибокої підтримки мови програмування Java та ефективних інструментів роботи з великими проєктами. Це середовище створює умови для підвищення якості, надійності та керованості програмного забезпечення, призначеного для модифікації вихідного коду, що робить його оптимальним вибором для реалізації подібних рішень.

Розроблено програмне забезпечення для модифікації вихідного коду.

Структура програмного забезпечення для модифікації вихідного коду розроблена з урахуванням мінімізації надлишкових елементів і зосереджена на реалізації ключових механізмів заплутовування програмного коду. Архітектура програмної системи відображена у вигляді діаграми класів, яка демонструє взаємодію основних програмних компонентів, відповідальних за аналіз,

обробку та трансформацію вихідного коду. Такий підхід дозволяє чітко окреслити функціональні межі кожного класу та забезпечити узгоджену роботу всієї системи без залучення допоміжних або другорядних модулів.

Описано функціонування програмного забезпечення для модифікації вихідного коду, що реалізує послідовний і логічно впорядкований процес обробки програмних файлів, починаючи з їх аналізу та завершуючи формуванням модифікованих версій. Запропонований механізм дозволяє зберігати коректність програмного коду, водночас істотно ускладнюючи його подальший аналіз сторонніми особами, що підтверджує ефективність обраного підходу. Реалізована модель функціонування програмного продукту забезпечує гнучке керування процесом модифікації завдяки системі налаштувань і дозволяє адаптувати алгоритми модифікації до конкретних потреб користувача. Це свідчить про практичну придатність програмного забезпечення для застосування у реальних програмних проєктах.

Описано особливості реалізації програмного забезпечення для модифікації вихідного коду.

Описано організацію зберігання та обробки даних у програмному забезпеченні для модифікації вихідного коду. Визначено, що у програмному забезпеченні для модифікації вихідного коду організація зберігання даних відіграє допоміжну, але важливу роль, оскільки забезпечує коректну обробку програмних файлів, тимчасових структур та результатів виконання алгоритмів модифікації. Особливістю такого програмного продукту є відсутність потреби у використанні повноцінної системи керування базами даних, оскільки обробка здійснюється над файлами вихідного коду та проміжними представленнями, які існують переважно протягом часу виконання програми. Обраний підхід до організації зберігання даних дозволяє зменшити складність програмного забезпечення, уникнути залежності від зовнішніх систем зберігання та забезпечити високу надійність роботи.

Виконано проєктування інтерфейсу взаємодії користувача з програмним забезпеченням для модифікації вихідного коду.

Виконано тестування розробленого програмного забезпечення для модифікації вихідного коду. Результати тестування програмного забезпечення показали, що розроблена програма дозволяє забезпечити автоматизацію процесів, пов'язаних з підтримкою процесу модифікації вихідного коду.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Source Code Management, Tools, and Best Practices. URL: <https://www.turing.com/blog/source-code-management-tools-best-practices> (date of access: 27.04.2026).
2. Improving Process of Source Code Modification Focusing on Repeated Code. URL: https://www.researchgate.net/publication/299714132_Improving_Process_of_Source_Code_Modification_Focusing_on_Repeated_Code (date of access: 27.04.2026).
3. Source code modification technology based on parameterized code patterns. URL: <https://ieeexplore.ieee.org/document/5783177> (date of access: 27.04.2026).
4. Steps to Improve Code Quality. URL: <https://stackify.com/7-steps-to-improve-code-quality/> (date of access: 27.04.2026).
5. Understanding the Code Modification Process: A Comprehensive Guide. URL: <https://reyabogado.com/us/what-is-the-code-modification-process/> (date of access: 27.04.2026).
6. Code Refactoring: Best Practices, Techniques & Tools. URL: <https://newdigitalstreet.com/code-refactoring/> (date of access: 27.04.2026).
7. Code Refactoring. Major Benefits and Best Techniques. URL: <https://xbsoftware.com/blog/code-refactoring-techniques-in-software-engineering/> (date of access: 27.04.2026).
8. AI Code Review Tools That Find Bugs & Flaws. URL: <https://www.digitalocean.com/resources/articles/ai-code-review-tools> (date of access: 27.04.2026).
9. Best Code Review Tools for Developers. URL: <https://kinsta.com/blog/code-review-tools/> (date of access: 27.04.2026).
10. Source Code Management Tools to Track Changes Effectively. URL: <https://geekflare.com/dev/source-code-management-tools/> (date of access: 27.04.2026).

11. List of the Top Source Code Management Software. URL: <https://topbusinesssoftware.com/categories/source-code-management/> (date of access: 27.04.2026).

12. Elite Software for Source Code Management. URL: <https://buddyxtheme.com/software-for-source-code-management/> (date of access: 27.04.2026).

13. Best Source Code Management Software. URL: <https://sourceforge.net/software/source-code-management/> (date of access: 27.04.2026).

14. Source Code Management Tools for Version Control. URL: <https://www.techjockey.com/blog/source-code-management-tools> (date of access: 27.04.2026).

15. DeepCode. URL: <https://snyk.io/platform/deepcode-ai/> (date of access: 27.04.2026).

16. CodeAnt AI. URL: <https://www.codeant.ai/> (date of access: 27.04.2026).

17. Refact.ai. URL: <https://refact.ai/> (date of access: 27.04.2026).

18. The Java Tutorials. URL: <https://docs.oracle.com/javase/tutorial/> (date of access: 27.04.2026).

19. Getting Started with Java. URL: <https://dev.java/learn/getting-started/> (date of access: 27.04.2026).

20. IntelliJ IDEA Tutorial. URL: <https://www.tpointtech.com/intellij-idea-tutorial> (date of access: 27.04.2026).

21. IntelliJ IDEA overview. URL: <https://www.jetbrains.com/help/idea/discover-intellij-idea.html> (date of access: 27.04.2026).

ДОДАТОК А
Технічне завдання

Вступ

Програмне забезпечення може використовуватися для модифікації вихідного коду.

A.1 Підстава для розробки

Підставою для розробки є завдання на дипломну кваліфікаційну роботу на тему «Програмне забезпечення для модифікації вихідного коду», затверджене наказом Національного університету «Запорізька політехніка» № 139 від 7 квітня 2026 р.

A.2 Призначення розробки

Програмний продукт призначений для зміни тексту вихідного програмного коду.

A.3 Основні вимоги до програми, що розробляється

A.3.1 Вимоги до функціональних характеристик

Програмне забезпечення для модифікації вихідного коду забезпечує виконання таких функцій:

- підтримка можливості завантаження та обробки файлів вихідного коду програмних проєктів з файлової системи користувача;
- підтримка можливості аналізу синтаксичної структури вихідного коду з метою коректного виявлення ідентифікаторів, константних рядків та інших елементів, придатних до модифікації;
- підтримка можливості вибіркового застосування додаткових методів модифікації, зокрема видалення коментарів, консольних повідомлень та конструкцій обробки винятків;

- забезпечення можливості налаштування параметрів модифікації вихідного коду відповідно до вимог користувача через графічний інтерфейс;
- підтримка можливості обфускації назв класів, методів та змінних без порушення логіки виконання програми;
- забезпечення стабільної роботи графічного інтерфейсу під час виконання ресурсоємних операцій шляхом використання окремих потоків виконання;
- підтримка можливості формування окремих файлів з модифікованим вихідним кодом без перезапису оригінальних файлів;
- забезпечення інформування користувача про стан виконання процесу модифікації та результати обробки файлів;
- підтримка можливості подальшого використання згенерованих файлів у складі нового або існуючого програмного проєкту.

A.3.2 Вимоги до інтерфейсу програми

Інтерфейс програмного забезпечення для модифікації вихідного коду повинен бути зручним для користувачів. Повинна бути забезпечена можливість роботи в візуальному режимі.

A.3.3 Вимоги до надійності

Програмне забезпечення для модифікації вихідного коду повинно забезпечити надійне функціонування.

A.3.4 Умови експлуатації

Для експлуатації програмного забезпечення для модифікації вихідного коду необхідна наявність персонального комп'ютера.

A.3.5 Вимоги до складу та параметрів технічний засобів

Для роботи програмного забезпечення для модифікації вихідного коду потрібні такі технічні та програмні ресурси: апаратна частина обчислювальної системи повинна забезпечувати стабільне виконання операцій аналізу та перетворення текстових файлів, а також коректну роботу графічного інтерфейсу користувача. Процесор комп'ютера має підтримувати багатопотокову обробку та мати тактову частоту не нижче середнього рівня (2ГГц), достатнього для виконання операцій синтаксичного аналізу та масових текстових заміन без суттєвих затримок. Обсяг оперативної пам'яті повинен бути не меншим за мінімально необхідний (8Гб) для одночасного зберігання вихідних файлів, проміжних структур даних та модифікованих результатів, що забезпечує безперервну роботу програми без перевантаження системи. Для зберігання даних необхідна наявність вільного дискового простору (32Гб), достатнього для розміщення оригінальних файлів проєкту, тимчасових даних та згенерованих модифікованих копій вихідного коду.

Програмне забезпечення має функціонувати під керуванням операційної системи, що підтримує виконання Java-застосунків та роботу з файловою системою на рівні користувача. Обов'язковою є наявність встановленого середовища виконання Java відповідної версії, яке забезпечує коректну роботу всіх модулів програми, зокрема парсера, модуля модифікації та графічного інтерфейсу. Для запуску та використання програмного продукту також необхідні стандартні системні бібліотеки, що входять до складу операційної системи, а також засоби відображення графічного інтерфейсу.

A.3.6 Вимоги до маркування і пакування

Програма для модифікації вихідного коду може бути записана на будь-якому носії інформації.

На пакуванні повинна бути назва програми – «Програмне забезпечення для модифікації вихідного коду».

A.4 Вхідні дані до роботи

Вхідними даними до програмного забезпечення для модифікації вихідного коду є файли вихідного коду програм, які користувач бажає модифікувати, а також додаткові параметри модифікації, включно з правилами обробки ідентифікаторів, шифруванням константних рядків та іншими налаштуваннями користувача.

Вихідними даними програмного забезпечення для модифікації вихідного коду є нові файли, що містять обфусковані версії вихідного коду, а також тимчасові або проміжні структури, які можуть зберігатися для контролю процесу обробки.

ДОДАТОК Б
Фрагмент тексту програми

```

private void stEvt() {
    btnLd.addActionListener(new ActionListener() {
        public void actionPerformed(ActionEvent evt) {
            ldFls();
        }
    });

    btnRmv.addActionListener(new ActionListener() {
        public void actionPerformed(ActionEvent evt) {
            rmvSlctd();
        }
    });

    btnStt.addActionListener(new ActionListener() {
        public void actionPerformed(ActionEvent evt) {
            sttFrm.setVisible(true);
        }
    });

    btnStrt.addActionListener(new ActionListener() {
        public void actionPerformed(ActionEvent evt) {
            strtWrk();
        }
    });
}

private void ldFls() {
    JFileChooser flChs = new JFileChooser();
    flChs.setMultiSelectionEnabled(true);
    int rslt = flChs.showOpenDialog(this);
    if (rslt == JFileChooser.APPROVE_OPTION) {
        File[] fls = flChs.getSelectedFiles();
        for (File fl : fls) {
            tblMdl.addFl(fl);
        }
    }
}

private void rmvSlctd() {
    int[] rws = tblFls.getSelectedRows();
    if (rws.length > 0) {
        tblMdl.rmvRws(rws);
    }
}

private void strtWrk() {
    if (wrkThrd == null || !wrkThrd.isAlive()) {

```

```

        wrkThrd = new WrkThrd(tblMdl.getFls(), prgBr,
sttFrm.getStt());
        wrkThrd.start();
    }
}

class FlTblMdl extends AbstractTableModel {

    private List<File> flLst;

    public FlTblMdl() {
        flLst = new ArrayList<>();
    }

    public void addFl(File fl) {
        flLst.add(fl);
        fireTableDataChanged();
    }

    public void rmvRws(int[] rws) {
        Arrays.sort(rws);
        for (int i = rws.length - 1; i >= 0; i--) {
            flLst.remove(rws[i]);
        }
        fireTableDataChanged();
    }

    public List<File> getFls() {
        return new ArrayList<>(flLst);
    }

    public int getRowCount() {
        return flLst.size();
    }

    public int getColumnCount() {
        return 1;
    }

    public Object getValueAt(int rw, int cl) {
        return flLst.get(rw).getName();
    }

    public String getColumnName(int cl) {
        return "Fl Nm";
    }
}

```

```

class SttFrm extends JFrame {

    private JCheckBox chkStr;
    private JCheckBox chkCmm;
    private JCheckBox chkCns;
    private JCheckBox chkXcp;

    private JTextArea txtXcl;
    private JButton btnSv;

    private Map<String, Boolean> sttMp;

    public SttFrm(JFrame prnt) {
        stFrm();
        stCmp();
        stEvt();
    }

    private void stFrm() {
        setTitle("Sttngs");
        setSize(400, 400);
        setLocationRelativeTo(null);
        setLayout(new BorderLayout());
    }

    private void stCmp() {
        JPanel pnlCntr = new JPanel(new GridLayout(0, 1));

        chkStr = new JCheckBox("Strng Mdftn");
        chkCmm = new JCheckBox("Rmv Cmmnts");
        chkCns = new JCheckBox("Rmv Cnsl");
        chkXcp = new JCheckBox("Rmv Xcptns");

        txtXcl = new JTextArea();
        btnSv = new JButton("Sv");

        pnlCntr.add(chkStr);
        pnlCntr.add(chkCmm);
        pnlCntr.add(chkCns);
        pnlCntr.add(chkXcp);

        add(pnlCntr, BorderLayout.NORTH);
        add(new JScrollPane(txtXcl), BorderLayout.CENTER);
        add(btnSv, BorderLayout.SOUTH);

        sttMp = new HashMap<>();
    }
}

```

```

private void stEvt() {
    btnSv.addActionListener(new ActionListener() {
        public void actionPerformed(ActionEvent evt) {
            svStt();
            setVisible(false);
        }
    });
}

private void svStt() {
    sttMp.put("str", chkStr.isSelected());
    sttMp.put("cmm", chkCmm.isSelected());
    sttMp.put("cns", chkCns.isSelected());
    sttMp.put("xcp", chkXcp.isSelected());
}

public Map<String, Boolean> getStt() {
    return new HashMap<>(sttMp);
}
}

class WrkThrd extends Thread {

    private List<File> flLst;
    private JProgressBar prgBr;
    private Map<String, Boolean> sttMp;

    public WrkThrd(List<File> flLst, JProgressBar prgBr,
Map<String, Boolean> sttMp) {
        this.flLst = flLst;
        this.prgBr = prgBr;
        this.sttMp = sttMp;
    }

    public void run() {
        int ttl = flLst.size();
        int cnt = 0;

        for (File fl : flLst) {
            prcsFl(fl);
            cnt++;
            int vl = (int)((double)cnt / ttl) * 100;
            prgBr.setValue(vl);
            try {
                Thread.sleep(100);
            } catch (Exception ex) {
            }
        }
    }
}

```

```

    }
}

private void prcsFl(File fl) {
    try {
        List<String> lnLst = Files.readAllLines(fl.toPath());
        List<String> nwLst = new ArrayList<>();

        for (String ln : lnLst) {
            String rs = ln;
            if (sttMp.getOrDefault("cmm", false)) {
                rs = rs.replaceAll("//.*", "");
            }
            if (sttMp.getOrDefault("cns", false)) {
                rs = rs.replaceAll("System.out.print.*", "");
            }
            if (sttMp.getOrDefault("xcp", false)) {
                rs = rs.replaceAll("try|catch|finally", "");
            }
            if (sttMp.getOrDefault("str", false)) {
                rs = rs.replaceAll("\"[^\\"]*\"", "\\\"X\\\"");
            }
            nwLst.add(rs);
        }

        Path pth = Paths.get(fl.getParent(), "mdf_" +
fl.getName());
        Files.write(pth, nwLst);

    } catch (Exception ex) {
    }
}

public class StrtApp {
    public static void main(String[] args) {
        SwingUtilities.invokeLater(new Runnable() {
            public void run() {
                new MnFrm().setVisible(true);
            }
        });
    }
}

private String gnrNwNms() {
    String src = "IlIlIlIlIlIlIlIlIlIlIlIlIlIlIlIlIl";
    StringBuilder bld = new StringBuilder();
    Random rnd = new Random();

```

```

        for (int i = 0; i < src.length(); i++) {
            bld.append(src.charAt(rnd.nextInt(src.length())));
        }
        return bld.toString();
    }

    private String mdftIdn(String ln, Map<String, String> mp) {
        String rs = ln;
        for (Map.Entry<String, String> nt : mp.entrySet()) {
            rs = rs.replaceAll("\\b" + nt.getKey() + "\\b",
nt.getValue());
        }
        return rs;
    }

    private Map<String, String> crtIdnMp(List<String> lnLst) {
        Map<String, String> mp = new HashMap<>();
        for (String ln : lnLst) {
            String[] tkns = ln.split("[^A-Za-z_$]");
            for (String tk : tkns) {
                if (tk.length() > 3 && !mp.containsKey(tk)) {
                    mp.put(tk, gnrNwNms());
                }
            }
        }
        return mp;
    }

    private List<String> aplyIdnMdft(List<String> lnLst) {
        Map<String, String> mp = crtIdnMp(lnLst);
        List<String> nwLst = new ArrayList<>();
        for (String ln : lnLst) {
            nwLst.add(mdftIdn(ln, mp));
        }
        return nwLst;
    }

    private List<String> rmvBlNkLn(List<String> lnLst) {
        List<String> nwLst = new ArrayList<>();
        for (String ln : lnLst) {
            if (ln.trim().length() > 0) {
                nwLst.add(ln);
            }
        }
        return nwLst;
    }

    public Set<String> prcsXcl() {
        Set<String> st = new HashSet<>();
    }

```

```

String[] lns = txtXcl.getText().split("\\n");
for (String ln : lns) {
    if (ln.trim().length() > 0) {
        st.add(ln.trim());
    }
}
return st;
}

class TxtMdftTls {

    public static List<String> rplcStrCn(List<String> lnLst) {
        List<String> nwLst = new ArrayList<>();
        for (String ln : lnLst) {
            nwLst.add(ln.replaceAll("\\\"[^\"]*\\\"", "\\\"XSTR\\\""));
        }
        return nwLst;
    }

    public static List<String> rmvTrSpc(List<String> lnLst) {
        List<String> nwLst = new ArrayList<>();
        for (String ln : lnLst) {
            nwLst.add(ln.trim());
        }
        return nwLst;
    }

    public static List<String> cnctLn(List<String> lnLst) {
        List<String> nwLst = new ArrayList<>();
        String bff = "";
        for (String ln : lnLst) {
            bff += ln;
            if (ln.endsWith(";")) {
                nwLst.add(bff);
                bff = "";
            }
        }
        if (bff.length() > 0) {
            nwLst.add(bff);
        }
        return nwLst;
    }
}

```

ДОДАТОК В
Слайди презентації

Міністерство освіти і науки України
Національний університет «Запорізька політехніка»

Програмне забезпечення для модифікації вихідного коду

Виконав: Артем ТАРАСОВ

ст. групи КНТ-122

Керівник: Степан СКРУПСЬКИЙ

к.т.н., доцент НУ «Запорізька політехніка»

Рисунок В.1 – Слайд 1

Об'єкт, предмет та мета роботи

Об'єкт дослідження – процес розробки програмного забезпечення для модифікації вихідного коду.

Предмет дослідження – програмні засоби для модифікації вихідного коду.

Метою роботи є розробка програмного забезпечення для модифікації вихідного коду, що забезпечує скорочення трудомісткості супроводу та рефакторингу програм.

[2]

Рисунок В.2 – Слайд 2

Завдання роботи

Для досягнення поставленої мети у кваліфікаційній роботі бакалавра необхідно розв'язати такі задачі:

- виконати аналіз предметної області та програмних засобів для модифікації вихідного коду;
- здійснити проектування програмного забезпечення для модифікації вихідного коду;
- створити програмне забезпечення для модифікації вихідного коду;
- виконати тестування розробленого програмного забезпечення для модифікації вихідного коду.

[3]

Рисунок В.3 – Слайд 3

Порівняння існуючих аналогів

Критерій порівняння	DeepCode	CodeAnt AI	Refact.ai
Автоматизація перевірок та виправлень	+	+	+—
Простота впровадження	+	+	+—
Інтеграція з системами контролю версій	+—	+	+—
Орієнтація на безпеку	+	+—	+
Можливість локального розгортання	—	—	+
Гнучкість налаштувань під проєкт	+—	+	+

[4]

Рисунок В.4 – Слайд 4

Вимоги до програми

При розробці програмного забезпечення для модифікації вихідного коду необхідно забезпечити такі функціональні вимоги:

- підтримка можливості завантаження та обробки файлів вихідного коду програмних проєктів з файлової системи користувача;
- підтримка можливості аналізу синтаксичної структури вихідного коду з метою коректного виявлення ідентифікаторів, константних рядків та інших елементів, придатних до модифікації;
- забезпечення можливості налаштування параметрів модифікації вихідного коду відповідно до вимог користувача через графічний інтерфейс;
- підтримка можливості обфускації назв класів, методів та змінних без порушення логіки виконання програми;
- підтримка можливості вибіркового застосування додаткових методів модифікації, зокрема видалення коментарів, консольних повідомлень та конструкцій обробки винятків;
- забезпечення стабільної роботи графічного інтерфейсу під час виконання ресурсоемних операцій шляхом використання окремих потоків виконання;
- підтримка можливості формування окремих файлів з модифікованим вихідним кодом без перезапису оригінальних файлів;
- забезпечення інформування користувача про стан виконання процесу модифікації та результати обробки файлів;
- підтримка можливості подальшого використання згенерованих файлів у складі нового або існуючого програмного проєкту.

[5]

Рисунок В.5 – Слайд 5

Порівняння мов програмування

Критерій порівняння мов програмування	Мова програмування		
	Java	Python	C++
Придатність для великих кодових баз	+	+-	+
Наявність бібліотек для аналізу та трансформації коду	+	+	-
Строга типізація та контроль помилок	+	-	+
Безпека та керування пам'яттю	+	+	-
Підтримка багатопотоковості	+	+-	+

[6]

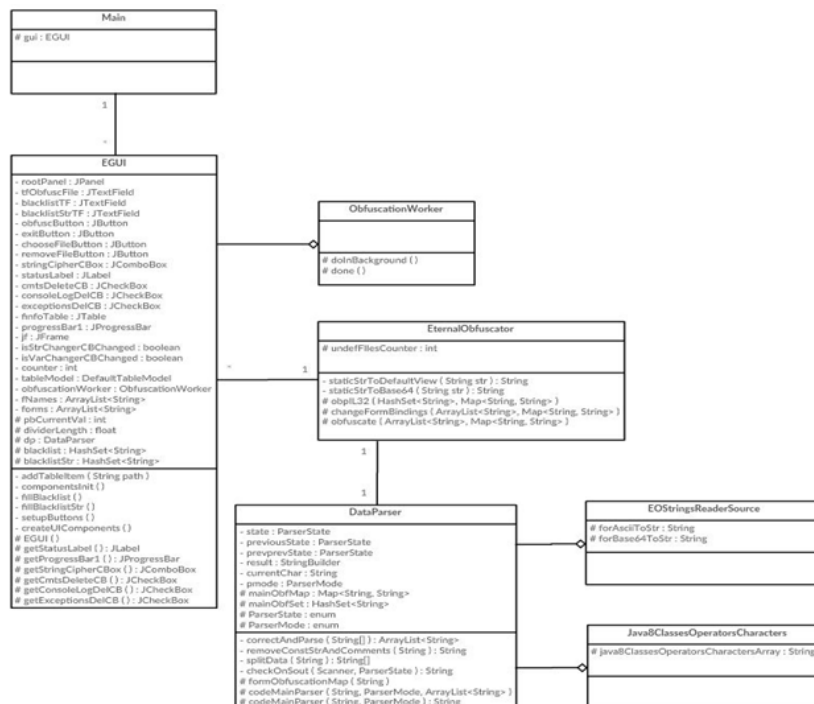
Рисунок В.6 – Слайд 6

Порівняння середовищ розробки

Критерій порівняння середовищ розробки	Середовища розробки		
	IntelliJ IDEA	Eclipse	NetBeans
Глибина аналізу вихідного коду	+	+—	+—
Інтелектуальні підказки та автодоповнення	+	+—	+—
Інтеграція з системами контролю версій	+	+	+—
Масштабованість середовища	+	+—	+
Можливості рефакторингу	+	+	+—

[7]

Рисунок В.7 – Слайд 7



Структура програми

[8]

Рисунок В.8 – Слайд 8

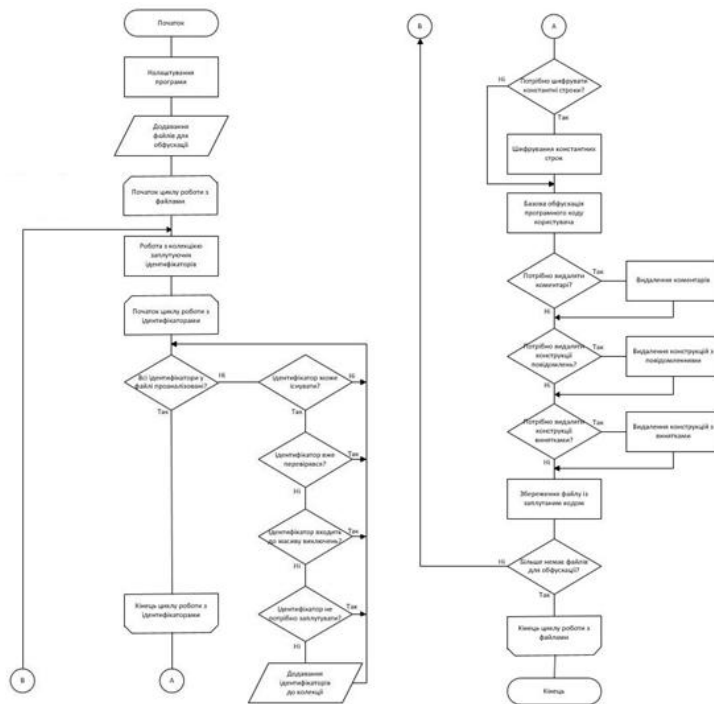


Схема функціонування програми

[9]

Рисунок В.9 – Слайд 9

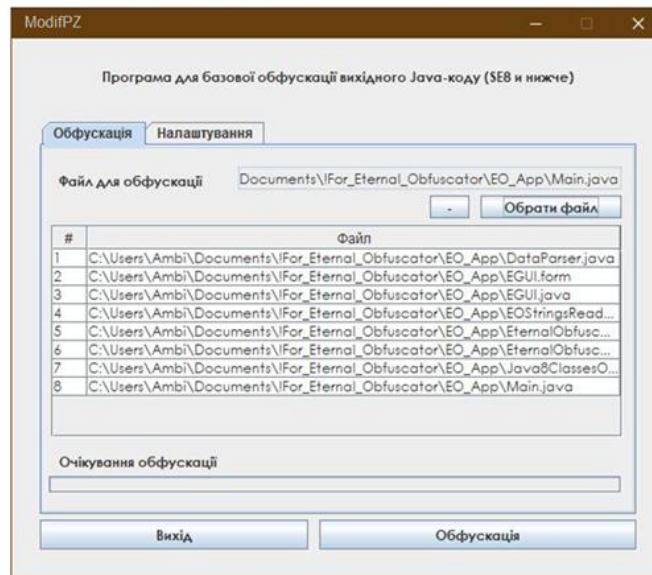
Організація зберігання та обробки даних



[10]

Рисунок В.10 – Слайд 10

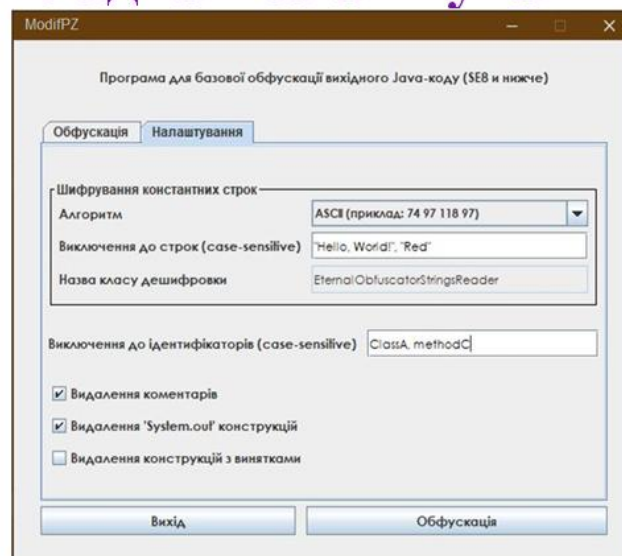
Робота з програмою. Головна форма



[11]

Рисунок В.11 – Слайд 11

Робота з програмою. Вкладка Налаштування



[12]

Рисунок В.12 – Слайд 12

Висновки

У дипломній кваліфікаційній роботі бакалавра було проаналізовано та досліджено процес розробки програмного забезпечення для модифікації вихідного коду.

Для реалізації програмного забезпечення для модифікації вихідного коду обрано мову програмування Java та середовище розробки IntelliJ IDEA.

Розроблено програмне забезпечення для модифікації вихідного коду.

В результаті виконання завдання було отримано систему, що відповідає всім вимогам представленим в технічному завданні, є зручною для використання та виконує всі необхідні функції.

Усі завдання випускної дипломної роботи бакалавра повністю виконано.

[13]

Рисунок В.13 – Слайд 13