МІНІСТЕРСТОВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ АВІАЦІЙНИЙ УНІВЕРСИТЕТ
Факультет кібербезпеки, комп‘ютерної та програмної інженерії
Кафедра комп’ютерних інформаційних технологій

ДОПУСТИТИ ДО ЗАХИСТУ
Завідувач випускової кафедри
_____ А.С. Савченко
«___» _____ 20___р

ДИПЛОМНИЙ ПРОЕКТ

(ПОЯСНЮВАЛЬНА ЗАПИСКА)

ВИПУСКНИКА ОСВІТНЬОГО СПУПЕНЯ «БАКАЛАВР»

Тема: «Методика оцінки якості програмного забезпечення підприємства "Альфа плюс"»

Виконавець: студент УС-411 Олійник Ілля Андрійович

(студент, група, прізвище, ім'я, по батькові)

Керівник: к. т. н., доцент Моденов Ю.Б.

(науковій ступень, вчене звання, прізвище, ім'я, по батькові)

Нормоконтролер: ст. викл. Шевченко О.П.

(П.І.Б.) (підпис)

КИЇВ 2021

НАЦІОНАЛЬНИЙ АВІАЦІЙНИЙ УНІВЕРСИТЕТ

Факультет кібербезпеки, комп‘ютерної та програмної інженерії

Кафедра комп’ютерних інформаційних технологій

Освітній ступінь: Бакалавр

Галузь знань, спеціальність, спеціалізація: 12 “Інформаційні технології”, 122 “Комп’ютерні науки”, “Інформаційні управляючі системи та технології”

ЗАТВЕРДЖУЮ

Завідувач кафедри

А.С. Савченко

“_____”_____ 2021 р.

З А В Д А Н Н Я

на виконання дипломної роботи студента

Олійника Іллі Андрійовича

1. Тема проекту (роботи): «Методика оцінки якості програмного забезпечення підприємства "Альфа плюс"» затверджена наказом ректора № 636/ст. від 24.04.2021р.

2. Термін виконання роботи: з 10.05.2020 по 14.06.2021р.

3. Вихідні данні до роботи: на основі потреб у автоматизованому тестуванні програмного забезпечення, розробити додаток для Jmeter;

4. Зміст пояснювальної записки (перелік питань, що підлягають розробці): якість програмного продукту, основні методи і задачі тестування, огляд сучасних засобів автоматизованого тестування програмного забезпечення.

5. Перелік обов'язкового графічного (ілюстративного) матеріалу: Додаток Apache Jmeter, середовище розробки – Eclipse,вхідні параметри Java запиту, список розроблених бібліотек та імпорт класів в Jmeter.

КАЛЕНДАРНИЙ ПЛАН

	Етапи виконання дипломної роботи	Термін виконання етапів	Примітка
1	Вивчення літературних джерел	11.05.2020р. – 12.05.2020р.	
2	Розробка та затвердження плану дипломного проекту.	13.05.2020р.	
3	Проведення консультації з науковим керівником щодо створення першого розділ.	14.05.2020р.	
4	Аналітичний огляд і постановка задачі.	15.05.2020р. – 18.05.2020р.	
5	Обґрунтування вибору засобу для тестування програмного забезпечення	19.05.2020р. – 22.05.2020р.	
6	Вивчення роботи програми Jmeter	23.06.2020р. – 27.05.2020р.	
7	Проектування сценаріїв тестування	28.05.2020р. – 04.06.2020р.	
8	Інтеграція Java додатку в Jmeter	05.06.2020р. – 08.06.2020р.	
9	Розробка бібліотек	09.06.2020р. – 10.06.2020р.	
10	Оформлення пояснювальної записки та графічної частини проекту	11.06.2020р. – 12.06.2020р.	

Керівник дипломної роботи (проекту) (Моденов Ю. Б.)

(підпис керівника) (П.І.Б.)

Завдання прийняв до виконання (Олійник І.А.)

(підпис випускника) (П.І.Б.)

РЕФЕРАТ

Пояснювальна записка до дипломного проекту «Методика оцінки якості програмного забезпечення підприємства "Альфа плюс"» містить: 99 аркушів з них, 27 малюнків, 2 таблиці.

Об’єкт дослідження: автоматизація програмного забезпечення.

Предмет дослідження: додаток Jmeter.

Мета роботи: автоматизація додатка користувача через інтеграцію бібліотек в Jmeter.

Методи дослідження, технічні та програмні засоби: середовище розробки Eclipse, мова програмування Java, бібліотеки Jmeter.

Отримані результати та їх новизна: розроблені бібліотеки, які можуть бути інтегровані в Jmeter для створення автоматизованих сценаріїв тестування.

ЯКІСТЬ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ, ТЕСТУВАННЯ, ЗАПИТ, АВТОМАТИЗАЦІЯ.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ ТА СКОРОЧЕНЬ

БД	– база даних
ПЗ	– програмне забезпечення
CSV	– Comma-separated values
HTTP	– Hypertext Transfer Protocol
HTTPS	– Hypertext Transfer Protocol Secure
IMAP	– Internet Message Access Protocol
IP	– Internet Protocol
JDBC	– Java Database Connectivity
JMS	– Java Message Service
LDAP	– Lightweight Directory Access Protocol
SMTP	– Simple Mail Transfer Protocol
SQL	– Structured Query Language
SSL	– Secure Sockets Layer
URL	– Uniform resource locator
XML	– eXtensible Markup Language
XPath	– XML Path Language

ВСТУП

Протягом років, розвиток сучасних технологій зумовив створення потрібності в вирішуванні складних, інформаційних завдань за допомогою програмного забезпечення. На початку розвитку обчислювальних машин запуск програмного забезпечення було надзвичайно затратним процесом, який потребував великого рівня професіоналізму спеціалістів та допуск до цінного обладнання, доступ до якого раніше був тільки у найбільших підприємств, в сьогоднішніх реаліях цей доступ має майже кожен. Було зроблено велику кількість спеціальних мов для створення програмного забезпечення, також вони потрібні для дослідження програмування, були створенні підходи та методики, що облегшують формування програмного лише спеціального, складного та коштовного обладнання, сьогодні можуть бути вирішенні шляхом використовування більш бюджетної сукупності програмного обладнання. Саме це і стало початком появи нових лиць в програмуванні.

На даному етапі ми можемо виділити декілька груп людей: професійні програмісти, ті, що використовують програмне забезпечення, як допоміжний аспект в їх роботі, і ті, яким задоволення приносить сам процес програмування, що вивчають його для особистого розвитку. Проте, далеко не всім вдається створити блоки, не кажучи вже про повну реалізацію програмного продукту. Велика частина від загальної кількості створених програм завершують своє існування на стадії зіткнення з програмними помилками. Усі ці помилки можуть привести до непоправних наслідків, які залежать від певного характеру програми. Якщо програма має відповідальний характер, то більш відповідальними та суровими повинні бути критерії її якості. Перш за все, глобальний продукт має відповідати усім стандартам якісної системи, якщо вона буде дотримуватись їх, вона буде

конкурентна з іншими програмними продуктами і принесе відповідний прибуток. Потрібно розуміти, якщо програма не буде видавати очікуваний результат, не відображати коректні вихідні дані, при точно вказаних вхідних даних, або ж повільно виповнювати поставлені задачі, це поставить під питання її функціональність та може привести до відказу потенційного користувача від її використання.

Якість функціоналу програмного продукту визначається деяким набором характеристик, які доносять необхідну інформацію про його ефективність для зацікавлених лиць. Проте кожен потенційний замовник має свій погляд на даний продукт, на його функціональність та дизайн. Саме тому для того, щоб забезпечити його якість, перш за все необхідно взнати погляд замовника на майбутній програмний продукт та знайти рішення його потреби. На жаль, не існує єдиного рішення усіх стандартів якості програмного забезпечення. Під час створення програмного продукту, розробник має можливість редагування на кожному з етапів створення, що значно полегшить її подальший функціонал та допоможе покращити показники її якості.

Саме тестування і є найважливішим аспектом перевірки якості програмного забезпечення. Суть тестування полягає в перевірці вихідних результатів при заданих масивах вхідних даних, зрівняння уже з прорахованими результатами для підведення висновків влаштування поставлених задач. Тестування і є одним з найбільш значущих етапів створення програмного забезпечення, який вносить чимало коректив в його функціональність.

Для перевірки результатів та проходження тестів якості програмного забезпечення використовується автоматизоване тестування за допомогою програмних засобів. Під час проходження методів гнучкого програмування Test Driven Development (TDD) спочатку пишуться тести, а потім вже

починає писатись код програми, сама програма вважається готовою після проходження усіх тестів. В таких підходах до тестування автоматизація закладена в самих методах. За її допомогою затрати часу на тестування стають набагато меншими. На жаль, автоматизоване тестування не зможете повністю замінити ручне, проте воно являється надзвичайно ефективним в регресійному, інсталяційному, модульному та інших видах тестування.

Вперше свій побачив автоматизацію тестування ще за часів систем CPM та DOS. Тоді її суть полягала в редагуванні командного рядка та аналізі вихідних даних. Дещо пізніше з'явилися варіанти роботи через мережу за допомогою віддалених запитів через API. Перші спогади автоматизованого тестування беруть свої корні ще з книги Фредеріка Брукса, під назвою "Людина – місяць", в ній мовиться про майбутнє модульного тестування. Хоча автоматизоване тестування почало набирати свої обороти вже під кінець 1980 років.

Всього є 2 варіанти розвитку подій в автоматизації тестування: GUI-тестування та тести на рівні коду. До GUI-тестування відноситься дублювання усіх дій користувача, що піддержуються тестовими фреймворками. До тестування на рівню коду відноситься модульне тестування.

Частіше всього використовується графічний користувальницький інтерфейс, що і являється одним із видів автоматизації програмного забезпечення. Цей вид тестування набрав популярності через можливість тесту додатку без доступу до його вихідного коду, також за допомогою такого інтерфейсу додаток може тестуватись тим же способом, яким буде працювати з ним людина.

На кожному етапі тестування необхідно приймати важливе рішення на рахунок ведення автоматизації. Можемо взяти за приклад графічний інтерфейс користувача, де потрібно редагувати тести на кожному етапі зміни

інтерфейсу. Надалі може з'ясуватись, що затрати на такий вид тестування стануть на багато більшими, ніж на тестування вручну.

Також потрібно розуміти, що етап тестування додатку не є головним аспектом оцінки його якості, хоча і є його невід'ємною частиною його життєвого циклу. Перш за все, потрібно скласти план ще на початку підготовки вимог тестування. Усі професіонали дотримуються ідеї, що чим раніше знайдена помилка, тим менше затрат піде на її виправлення.

У даній роботі розглянуто усі головні аспекти автоматизації програмного продукту, знайдені найважливіші варіанти рішення головних проблем тестування, пройдена повна робота над аналізом переваг та недоліків, дані рекомендації та визначена ефективність по впровадженню автоматизації в тестування програмного забезпечення.

Мета дипломного проекту полягає в розширенні автоматизації тестування програмного продукту, шляхом розробки нових бібліотек додатку Jmeter.

РОЗДІЛ 1
ВИВЧЕННЯ ОСОБЛИВОСТЕЙ ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

1.1. Якість програмного забезпечення, основні принципи та тверження

Якість програмного забезпечення - це поєднання особливостей програмного забезпечення, які потрібні для вирішування його потреб.

Забезпечення якості - це поєднання способів, які включають етапи створення, випуску та використовування програмного забезпечення інформаційних систем, які потрібні на етапі життєвого циклу програмного забезпечення, для збільшення якості продукту, який використовується.

Контроль якості - це поєднання процесів, що виконуються над об'єктом для тестування, для одержання інформації про його стан в розділах:

- Готовність до випуску;
- Відповідність усім вимогам;
- Відповідність необхідному рівню якості.

Верифікація - це оцінювання системи або її складових для того, щоб визначити чи влаштовують створення умов даного етапу, сформованих на початку. Тобто дізнатися більше інформації про виконання наших термінів, цілей, завдань, визначених на початку чинної фази.

Щоденно у нашій роботі ми маємо діло з дуже абстрактними поняттями 《якості програмного забезпечення》 і якщо запитати тестувальника чи програміста - “що таке якість програмного забезпечення?”, кожен з них знайде унікальне тлумачення цього терміну, розглянемо його очима міжнародних норм та стандартів:

- якість ПЗ - це ступінь, в якому програмне забезпечення має необхідну сукупність властивостей [iso 840 г Управління та забезпечення якості].

- якість ПЗ - це комбінація властивостей програмного забезпечення, яка потрібна для вирішення поставлених, заздалегідь передбачуваних потреб.

Особливості якості програмного забезпечення:

- функціональність - відзначається можливістю програмного забезпечення виконувати задачі, що відповідають поставленим потребам, при заздалегідь відомих умовах використання програмного забезпечення. Тобто функціональність несе відповідальність за справність ПЗ і функціональну сумісність, захищеність та відповідність усім стандартам у галузі.

- надійність - можливості ПЗ вирішувати поставлені задачі в необхідних умовах протягом вказаного інтервалу часу або за вказану кількість операцій. Головні атрибути цієї характеристики - цілісність і завершеність системи, її можливість коректно, самостійно працювати після найменших помилок в роботі.

- зручність супроводу - це зручність, за допомогою якої програмне забезпечення може тестуватися, аналізуватися, змінюватись для виправлення помилок, для полегшення подальшої експлуатації та реалізації вимог користувача.

- зручність у використанні - це змога доступного розуміння, використання та вивчення програмного забезпечення для користувача.

- портативність - описує програмне забезпечення через легкість перенесення між оточеннями.

- ефективність - це можливості ПЗ забезпечувати задану продуктивність з даними ресурсами, часом, тощо.

Рис. 1.1. Модель якості програмного забезпечення

Зараз найчастіше використовуються багаторівнева модель якості програмного забезпечення. Вона має вигляд набору стандартів ISO 9126. На найвищому рівні є шість основних особливостей якості програмного забезпечення, кожний з яких визначається відповідним набором атрибутів. (див. рис. 1.1)

1.2. Тестування та типи тестувань програмного забезпечення.

Тестування ПЗ - це випробування відповідності між дійсною та очікуваною поведінкою програмного забезпечення, яка виконується на комплексі тестів що обирається спеціальним чином [IEEE GuideToSoftwareEngineeringBodyKnowledge 2004].

Іншими словами тестування - це техніка контролю якості, яка включає планування дій, виконання, проєктування тестів та аналіз одержаних результатів.

Верифікація - це етап оцінювання системи або її складових, для того, щоб визначити чи дотримуються результати даного етапу розробки усіх умов, сформованих на його початку [IEEE]. Перевірити чи виконуються поставлені терміни цілі та завдання, які були визначені на початку поставленої фази.

Валідація - це процес визначення відповідності розробки програмного забезпечення за потребами та очікуваннями користувача, згідно з усіма вимогам системи [BS7925-1].

План тестування - це загальний документ, який містити весь обсяг робіт тестування, що починається зі стратегії, розкладів, опису об'єкта, критеріїв початку та завершення тестування, закінчуючи необхідними в роботі спеціальними знаннями, обладнанням, а також необхідною оцінкою усіх ризиків та варіанти їх рішень.

Тест дизайн - це один з етапів тестування програмного забезпечення, на ньому створюються та проєктуватися тест кейси, згідно з цілями тестування та їх критеріями.

Баг репорт - це загальний документ, який містити послідовність дій, що призвели до неправильної роботи об'єкти з зазначеними очікуваннями результату.

Деталізація тест кейсів - це деталізація необхідного результату та опису тестових кроків для його досягнення, за допомогою якої забезпечується відношення тестового покриття до часу.

Тестовий випадок - це артефакт, який описує конкретні умови і параметри, необхідні для реалізації та перевірки тестових функції або їх частин.

Тестове покриття - це одна з систем оцінювання якості тестування, яка являється щільністю тестів виконуваного коду або його вимог.

Час проходження тестових випадків - це проміжок часу від початку етапу проходження кроків тестового випадку і до отримання результатів тесту.

Усі типи тестування програмного забезпечення залежать від їх цілей, ми можемо розділити на три групи:

- пов'язані зі змінами;
- функціональні;
- нефункціональні.

Далі ми будемо намагатися більш інформативна розповісти про кожний тип тестування, його використання та призначення тестування програмного забезпечення.

Функціональні тести будуються на особливостях та функціях взаємодії з системою, вони представлені на всіх етапах тестування: модульному або компонентному, системному, інтеграційному та приймальному.

Функціональні типи оцінюють поведінку системи. Найбільш поширені типи функціональних тестів представлені нижче:

- тестування безпеки;
- функціональне тестування;
- тестування взаємодії.

Нефункціональне тестування показує тести, які потрібні для пояснення особливостей програмного забезпечення, вони вимірюються в різних величинах. Найбільш поширені типи функціональних тестів представлені нижче.

Тестування продуктивності:

- стресове тестування;
- об'ємне тестування;
- тестування навантаження;
- тестування надійності;
- тестування зручності для користування;
- конфігураційне тестування;
- тестування установки;
- тестування на відновлення.

Після всіх змін, виправлення багів, програмне забезпечення має бути протестована для узгодження факту, що помилка була вирішена. Найбільш поширені типи тестувань призначені для узгодження правильності виправлення багу та працездатність додатку представлені нижче:

- регресійне тестування;
- тестування збірки;
- димове тестування;
- санітарне тестування.

1.3. Автоматизоване тестування програмного забезпечення.

Автоматизоване тестування ПЗ - це етап верифікації ПЗ, в якому головні функції тесту, такі як ініціалізація, запуск, аналіз, виконання та видача результату, реалізовуються автоматично, використовуючи інструменти для автоматизації тестування.

Експерт з автоматизації тестування ПЗ - це кваліфікований розробник ПЗ, в обов'язки якого входить створення, підтримка та налагодження стану тестових скриптів та комплектів інструментів для тестування.

Тест скрипт - це комплекс вправ для автоматизованої перевірки деякої частини ПЗ.

Інструмент для автоматизованого тестування - це ПЗ, з його допомогою експерт з тестування може здійснювати створення, підтримку, аналіз та налагодження результатів скриптів.

Тест набір - це набір тест скриптів, для випробування деякої частини ПЗ, поєднаною загальними цілями, необхідними для запуску набору.

Тест для запуску - це поєднання тестових наборів для спільного запуску (паралельного чи послідовного).

1.4. Навантажувальне тестування програмного забезпечення.

Зараз програмне забезпечення повинно без зупинки працювати під великим навантаженням, будь-яка проблема, яка пов'язана з несправною продуктивністю, автоматично може стати причиною відмови користувача від використання продукту, саме тому тестування навантаження стає необхідністю для стабільної роботи додатків.

Тестування продуктивності - автоматизоване тестування, що виконує роботу для певної кількості користувачів.

На початку роботи з навантажувального тестування, потрібно розуміти, це не прогін чи запис скриптів, а набагато складніший процес:

- тестування навантаження - це глобальна аналітична робота.
- тестування навантаження - це тестування, при якому необхідно мати набір навичок програмування, знання протоколів мережі та баз даних.

Різні типи тестування ставлять перед собою різну мету. Для того, щоб розповісти про підходи такого виду тестування та які проблеми за допомогою нього вирішуються, потрібно розібратися в термінології.

Навантажувальне тестування - це тестування, що виконує роботу для певної кількості користувачів або на загальнодоступному ресурсі. За приклад можемо взяти банківських співробітників, де вони працюють лише через один додаток, який знаходиться на серверах банку. Також, як приклад, можемо взяти веб-магазин, в якому навантажують сервер користувачі інтернету.

За допомогою спеціальних технік та продуктів може відбуватись моделювання навантаження. Зараз необхідно розібратися з термінологією:

- ітерація - це одноразовий повтор операції в циклі.
- навантаження - спільні операції на загальному ресурсі.
- масштабованість додатків - зростання продуктивності.
- віртуальний користувач - процес програми, що виконує модельовані операції.
- інтенсивність операції - частота здійснення операції.
- продуктивність - загальна кількість операції за певний час.
- навантажувальна точка - задана кількість користувачів, що виконують операції.
- профіль навантаження - це комплекс операцій із вказаною частотою, який одержуються за допомогою статистичних даних.

Розглянемо, що спільного мають ці сутності. Якщо ми виставимо інтенсивність через час між ітераціями, то ми побачимо її ріст за скорочення інтервалу. Зростання інтенсивності пропорційне навантаженню. Якщо ми збільшимо інтенсивність, то зростає продуктивність, також зростає ступінь завантаженості ресурсу. В певний проміжок часу збільшення продуктивності зупиняється та відбувається деградація системи. Зміна навантаження на ресурси може підкорятися чи якомусь закону, чи залишатися рівномірною.

1.5. Автоматизоване функціональне тестування

Функціональне тестування програмного забезпечення - це етап верифікації характеристик додатка за допомогою інструментів для тестування.

Багато невірних уявлень пов'язано з автоматизованим тестуванням, проте це не дивно для вузько спрямованих IT-дисциплін. Для того, щоб мінімізувати неправильне застосування автоматизації, необхідно максимально використати її переваги та уникати її недоліки.

На даному етапі нам потрібно дати опис основних нюансів та відповісти на головне питання, коли її потрібно використовувати.

Головні переваги автоматизації тестування:

- швидкість виконання - нема потреби постійно перевіряти інструкції та документацію, це значно зменшує затрати часу.

- звіти - зберігаються автоматично, розсилають результати тестування.

- повторюваність - всі тести будуть виконані автоматично, буде відсутня можливість людської похибки, не буде ніяких проблем у результатах.

- витрати на підтримку - витрати часу автоматичного скрипта стануть значно меншими для аналізу та подальшої підтримки, ніж для проведення цих операцій вручну.

- виконання автоматичного тестування буде відбуватися без втручання інженера-тестувальника, який в цей час може займатися іншою роботою.

Головні недоліки автоматизації тестування:

- повторюваність - це також є і недоліком, тести виконуються одноманітно. Виконуючи тест вручну тестувальник зможе звернути увагу на баг та відразу його виправити, скрипт немає такої можливості.

- витрати часу на розробку - це дуже складний та тривалий процес, йде повноцінна розробка програмного забезпечення, яка тестує додаток користувача, а

це потребує бібліотек, фреймворків та іншого. Безумовно це все потрібно також налагоджувати та тестувати, а це займає чимало часу.

- витрати на підтримку - усі додатки мають властивість змінюватись та оновлюватись, це означає що витрати на їх підтримку будуть рости, хоча вони і залишається значно меншими, ніж при ручному тестуванні.

- пропуск дрібних помилок - скрипт може не помітити помилку на пошуки якої він не запрограмований, це такі помилки, як помилка контролю форм, помилка в позиціювання вікон чи помилка в написах.

- вартість інструменту - якщо розглядати користування ліцензійним програмним забезпеченням, то вартість такого може бути досить дорога, безкоштовні інструменти зазвичай відрізняються меншою зручністю роботи з ними та меншим функціоналом.

Перш за все потрібно переглянути усі плюси та мінуси автоматизованого тестування, порівняти їх, лиш тоді вирішувати чи звертатись за допомогою до скрипту, чи віддати перевагу ручному тестуванню, проте потрібно пам'ятати, що тестування вручну має також чимало недоліків.

1.6. Рівні автоматизації тестування

Додаток можна розділити на три рівні:

– UnitTestsLayer;
– FunctionalTestsLayer (Non-UI);
– GUI TestsLayer.

Для того, щоб продукт був найбільш якісним, потрібно автоматизувати усі його три рівні:

– UnitTestLayer(рівень модульного тестування);

На даному рівні автоматизованими тестами називають компонентні та модульні тести розробників. Якщо знання та кваліфікація експерта дозволяє йому писати такі автоматизовані тести, то він може це зробити на ранніх стадіях проекту, це допоможе уникнути багатьох проблем.

– FunctionalTestLayer(рівень функціонального тестування);

Не завжди внутрішню логіку додатка можна перевірити через шар Gui. Тому командою тестувальників буде розроблений доступ одразу до функціонального шару, уникаючи інтерфейс користувача.

– GuiTestLayer(рівень тестування через користувацький інтерфейс).

Саме на цьому етапі виконуються операції захвату бізнес-логіки додатку, не кажучи про інтерфейс користувача та функціональність.

РОЗДІЛ 2
ДІЮЧІ СПОСОБИ АВТОМАТИЗАЦІЇ ТЕСТУВАННЯ

2.1. Тестування додатків на рівні коду

Тестування на рівні коду застосовується в регресійному та модульному тестуванні. Основні його характеристики:

- Полегшення консолідації. Такий вид тестування допомагає ліквідувати вагання на рахунок певних модулів, тоді використовується варіант тестування «знизу – вверх»: спершу йде тестування окремих частин програми, лиш потім тестується програма в цілому. Цей варіант значно облегшує інтеграційне тестування. Потрібно розуміти, що інтеграційне тестування абсолютно не схожа навіть на ретельно побудовану ієрархію модульних тестів. Це вимагає роботи відразу декількох людей та не підлягає повній автоматизації.

- Стимуляція змін. Якщо програміст впевнюється у працездатності модулю за допомогою регресійного тестування, модульне тестування дозволяє йому зробити рефакторинг. Це призводить до зацікавленості розробника в зміні коду, оскільки код працює і після змін та набагато легше зробити перевірку.

- Роз'єднання реалізації та інтерфейсу. Оскільки ієрархія має класи, що містять в собі інші класи, тестування одного з них поширюється на інші, пов'язані з ним. Можемо розглянути приклад, де клас використовує базу даних; під час написання коду програміст розуміє, що тест взаємодіє з базою даних. Це являється дефектом, тому що тест не має виходити за рубіж класу. Після цього програміст віддаляється від роботи з базою даних та переходить до реалізації інтерфейсу через свій mock-об’єкт. Таким чином зменшуються залежності в системі за допомогою менш зв’язаних складових коду.

- Документування. Для класу, що проходить тестування, модульне тестування можна розглядати як повноцінний документ. Деякі клієнти, що вперше працюють з даним тестом, використовують модульний тест в ролі прикладу.

Звичайно, такий тип тестування також має свої недоліки:

- Дуже великий розмір написаного коду. Перевірка програмного продукту це дуже об’ємна та клопітлива праця. Можемо взяти за приклад булеву зміну, яка має два значення: перше – True, друге – False. В зв’язку з цим на первірку кожного рядку вихідного коду, потрібно буде написати до 5 тестових рядків.

- Немає доступу до вправлення усіх дефектів. Причиною цього стала неможливість вистежування усіх варіантів роботи програмного забезпечення, якщо не рахувати елементарних випадків. Окрім цього, паралельно відбувається тестування усіх модулів окремо. Отже, помилки функцій, інтеграцій, системного рівня, не будуть знайдені. Також така технологія даремна для тестування продуктивності програмного забезпечення. Саме тому, в поєднанні з іншими методами тестування, модульне тестування стає більш ефективним.

- Вимогливе слідування методам тестування. Автоматизоване тестування вимагає зберігати записи не тільки про усі проведені тести, але і про переміни кінцевого коду в кожному модулі програми. Необхідно дотримуватись організовування контролю версій програмного забезпечення. Тому, якщо кінцева

версія програмного забезпечення не буде в змозі пройти тестування, хоча бездоганно проходила його раніше, ми зможемо перевірити вихідний код та ліквідувати дефект. Також потрібно перевірити в стабільному вистежуванні невдалих тестів та їх аналізу. Нехтування такими умовами неминуче призведе до постійного збільшення негативних результатів тесту.

- Тестувальний інструментарій створений під велику кількість мов програмування. Деякі з них, наприклад Cobra та D, мають інтеграцію модульного тестування в граматику самої мови, без використання сторонніх бібліотек.

Бібліотеки для модульного тестування:

- Для C++:
    - ✓ CPPUnit;
    - ✓ Google C++ Testing Framework;
    - ✓ BoostTest;
    - ✓ Symbian - Фреймворк для Symbian OS всіх версій;
    - ✓ OC.DUnit — для Delphi;
    - ✓ API SanityAutotest - для динамічних C / C++ бібліотек в Unix-подібних.
- Для Python:
    - ✓ PyTest;
    - ✓ PyUnit;
    - ✓ Nose.
- Для Perl:
    - ✓ Test::Unit;
    - ✓ Test::Simple;
    - ✓ Test::Unit::Lite.
- utPLSQL — PL/SQL;
- vbUnit — VisualBasic;
- Для T-SQL:

    - ✓ SPUnit
    - ✓ TSQLUnit
- Для Java:
    - ✓ TestNG testNG.org
    - ✓ JUnit JUnit.org
    - ✓ JavaTESKUniTESK.ru
- NUnit — для мов платформи .NET: C#, VisualBasic .NET та ін;
- Для C:
    - ✓ cfixcfix
    - ✓ CTESK UniTESK.ru
    - ✓ CUnitcunit
- Для ActionScript 2.0 - мова сценаріїв, що застосовується віртуальною машиною AdobeFlashPlayer версії 7 і 8:
    - ✓ AS2Unit
- Для ActionScript 3.0 - скриптова мова, що застосувається віртуальною машиною AdobeFlashPlayer версії 9:
    - ✓ AsUnit
    - ✓ FlexUnit
- JsUnit — для JavaScript;
- Test:Unit — для Ruby.

2.2. Тестування за допомогою графічного інтерфейсу

Цей вид тестування набув популярності через можливість тесту додатку без доступу до його вихідного коду, також за допомогою такого інтерфейсу додаток може тестуватись тим же способом, яким буде працювати з ним людина.

GUI-автоматизація розвивалася протягом 4 етапів технік та інструментів:

- Сценарії (Scripting) - вид програмування на мовах, що були спеціально розроблені для автоматизованого тестування програмного забезпечення – вирішує чимало проблем playbacktools/capture. Проте створенням займаються розробники найвищого рівня, що виконують свою роботу окремо від тестувальників, на пряму займаються запуском тестів. Крім цього скрипти не можуть бути впровадженими в систему або взагалі будь-яким чином бути об'єднаними з нею, хоч вони і найкраще підходять для тестування GUI. Також, зміни в програмному продукті вимагають об’ємних змін у відповідних скриптах, і мають отримувати підтримку бібліотекою тест-скриптів, що стає в кінці неможливою задачею.
- Утиліти відтворення та запису (playbacktools/capture) записують послідовність дій тестувальника, який займається ручним тестуванням. Таким чином вони мають можливість здійснювати тестування без допомоги людини протягом усього часу виконання тестів, таким чином це значно збільшує продуктивність і усуває одноманітне повторення одних й тих самих дій, як це буває в ручному тестуванні. Паралельно, усі зміни в тестованому програмному продукту призведуть до перезапису та зміни ручних тестів. Це і являється причиною того, що це покоління не ефективне, а тому і не масштабоване.
- Data-driventesting - це сукупність декількох взаємодіючих тестових скриптів і їх джерел даних під Фреймворк, використовуваний в методології. В такому фреймворку змінні потрібні і для вхідних, і для вихідних значень: в тестовому скрипті звичайно закодовані ведення логів тестування, читання джерел даних та навігація по додатку. Таким чином, логіка також залежить від даних, що використовуються в скрипті. Data-driventesting – метод, потрібний для автоматизації тестування. Характерною функцією є те, що тестові скрипти реалізують верифікацію на основі даних, які зберігаються в базі даних або їх центральному сховищі. Функції баз даних можуть виконувати csv, xls файли або ODBC-ресурси і т.д.

- Keyword-based виконує поділ процесу розробки тестових випадків на два етапи: планування та реалізації.
- Відомі програмні продукти для такого виду тестування бувають:

З відкритим вихідним кодом:

- Jmeter;
- Selenium.

Комерційні:

- AutomatedQATestComplete;
- IBM RationalFunctionalTester, IBM RationalPerformanceTester, IBM RationalTestStudio;
- HP LoadRunner, HP QuickTest Professional, HP QualityCenter.

Також не можемо пройти повз тестування навантаження, яке також автоматизовується. Тестування продуктивності (PerformanceTesting) або навантажувальне тестування (LoadTesting) - це автоматизоване тестування, яке повторює роботу деякої кількості бізнес користувачів на певному ресурсі.

Основними напрямками навантажувального тестування є:

1. Характеристика продуктивності і працездатності додатки на етапі випуску патч-сетів та нових релізів.

2. Характеристика працездатності і продуктивності додатки на етапі розробки і передачі в експлуатацію.

3. Підбір відповідної для даного використання конфігурації сервера і програмної платформи.

4. Оптимізація продуктивності додатки, включаючи оптимізацію коду і налаштування серверів.

Звернему увагу, що в рамках однієї цілі можуть застосовуватись різні типи тестів продуктивності і навантаження, для прикладу, для перших трьох цілей необхідно виконувати як тестування стабільності, так і тестування продуктивності. Але при плануванні навантажувального тестування логічніше все ж

відштовхуватися від технічних цілей, які отримуються в результаті тестування і класифікувати тести за ними:

1. Якщо розглянемо дослідження продуктивності додатка, саме часи відгуку для операцій на різних навантаженнях в досить об'ємних діапазонах, включаючи стресові навантаження, то отримаємо тестування продуктивності.

2. Якщо ціллю є перевірка стійкості додатку (в режимі тривалого використання), то проводиться тривалий процес навантажувального тестування - це тест стабільності. При цьому аналіз часу відгуку може бути, але не головним за пріоритетом.

3. Стрес тестування ставить перед собою ціль перевірити, чи повертається до роботи система після надмірного навантаження до нормального режиму, а також метою стресового тестування можуть стати перевірка системи у тих випадках, коли один із серверів програмного продукту в блоці перестає працювати або критично змінилася апаратна конфігурація бази даних і т.д. Потрібно підмітити, під час стрес тестування перевіряється здатність системи до регенерації після навантаження, а ніяк не її продуктивність.

Комерційні інструменти для автоматизованого тестування навантаження:

Таблиця 2.1.

Комерційні продукти для навантажувального тестування

Hewlett-Packard (MercuryInteractive)	HP PerformanceCenter (включає HP LoadRunner)
AutomatedQACorp	TestComplete
IBM Rational	SilkPerformer
Borland (Segue)	RationalPerformanceTester
Microsoft	MS WebApplicationStressTool

В цілому можна скласти зведену таблицю найбільш популярних засобів автоматизації.

Таблиця 2.2.

Засоби для автоматизації тестування

Виробник	Функціональне тестування	Навантажувальне тестування	Якість коду	Керування тестами
Jmeter	+	+	-	+
AutomatedQA	+	+	-	+
HP	+	+	+	+
IBM	+	+	+	+
Open-source	Abbot, Selenium, Watir	Grinder, OpenSTA	GCT, NCover, Cobertura	FitNesse, TestLink

У зв’язку з тим, що Jmeter являється безкоштовним у використанні, йому і була надана перевага.

2.3. Знайомство з додатком Jmeter

ApacheJMeter необхідний додаток з доступним вихідним кодом, розроблений з підтримкою мови програмування Java і призначений для функціонального та навантажувального тестування програмного забезпечення та оцінки його

ефективності. Перш за все був розроблений для тестування веб-додатків, проте зараз розповсюджений на інші види програмного забезпечення.

ApacheJMeter може бути використаний для тестування продуктивності динамічних і статичних ресурсів. Цей додаток може бути використаний для моделювання масштабного навантаження на об’єкти, мережі чи сервера, для перевірки його працездатності та аналізу загальної продуктивності при абсолютно різних типах навантаження. Його можливо використовувати для графічного аналізу продуктивності, з метою перевірки скрипту/сервера/поведінки об'єкта при тяжкому навантаженні.

Можливості і переваги ApacheJMeter:

- Навантажувальне тестування та тестування продуктивності для різних типів серверів :
  - ✓ SOAP;
  - ✓ Web - HTTP, HTTPS;
  - ✓ JMS;
  - ✓ LDAP;
  - ✓ База даних через JDBC;
  - ✓ Рідні скрипти або команди;
  - ✓ Пошта - POP3 (S), SMTP (S) і IMAP (S).
- Багатопоточність дозволяє одночасно проводити процес відбору зразків та проб різних функцій в окремих групах.
- Повна мобільність і 100% Java.
- Кешування та автономне відтворення / аналіз результатів тестів.
- Розробка графічного інтерфейсу дозволяє проводити більш точні виміри та значно прискорити тестування.
- Можливість розширення додатку:
  - ✓ Кількість витягів статистик навантаження обирається за допомогою підключених таймерів.

- ✓ Делімітовані тестові можливості.
- ✓ Функції можуть бути використані для отримання можливості керування даними або ж для динамічного введення тестування.
- ✓ Візуалізації та аналіз даних, плагіни представляють персоналізацію, а також велику розширюваність.
- ✓ Підключення скрипкових плагінів.

2.3.1. Створення плану тестування з використання Jmeter

Найголовніше з чого потрібно почати роботу з JMeter - план тестування. План тестування додатку це опис послідовності кроків, які буде виконувати JMeter. План тестування складається з:

- Логічні контролери (Logicconrollers);
- Групи потоків (ThreadGroups);
- Типові контролери (Samplegeneratingcontrollers);
- Відповідності (Assertions);
- Слухачі (Listeners);
- Конфігураційні елементи (Configurationelements);
- Таймери (Timers).

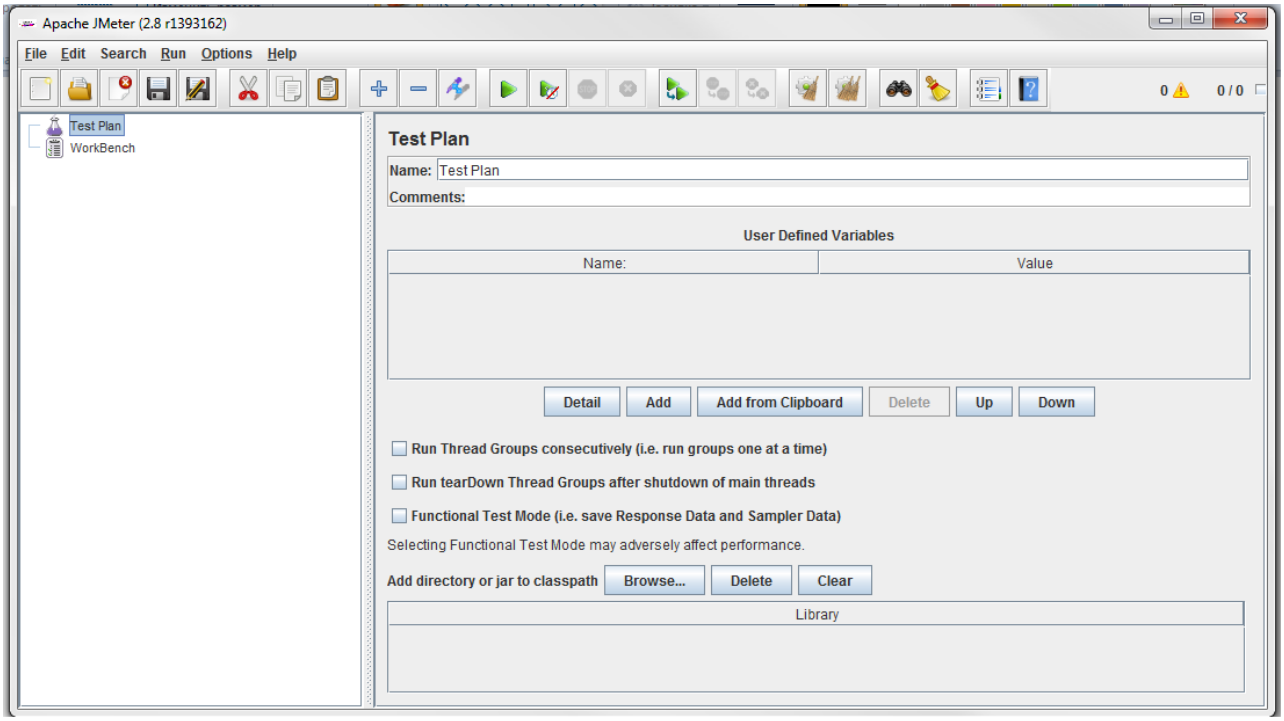

Рис. 2.1. Головна сторінка ApacheJmeter

Необхідно звернути увагу на те, що на рисунку 2.1, дерево елементів спочатку складається з двох пунктів: WorkBench і TestPlan.

WorkBench являє собою тимчасову папку. Суть інтерфейсу Jmeter полягає в переміщенні елементів дерева сценарію з місця на місце. Спочатку конструюється дерево плану, після чого його частини переміщаються на "верстак" та назад. Вміст Workbench не зберігається на диску. Виконується в процесі тестування лише вміст TestPlan.

Для наповнення TestPlan елементами сценарію використовуйте або головне меню додатка "Edit ->Add -> Елемент сценарію", або контекстне меню на елементі "TestPlan".

2.3.2. Створення групи потоків

ThreadGroup дає можливість встановлювати параметри навантаження на додаток. Головними його параметрами є:

– Ramp-upperiod - інтервал часу, через який виконується запуск наступного етапу процесу;

– Numberofthreads - кількість імітованих користувачів, що одночасно працюють з сайтом;

– Forever – цей параметр вказує на те, що сценарій буде виконуватися завжди, доки не буде перервано втручанням;

– Loopcount - кількість разів, яке буде виконуватися сценарій всередині ThreadGroup;

– Actiontobetakenafter a SampleError - процес, що виконується після того, як запит буде видавати помилку;

– Scheduler – планувальник для часу роботи сценарію.

Щоб додати ThreadGroup з контекстного меню елемента «TestPlan» виберіть «Add» -> «ThreadGroup» (див. рисунок 2.2).

Тільки всередину елемента «ThreadGroup» можна додавати елементи запитів - «FTP Request» , «WebServiceRequest», «HTTP Request» і т.д.

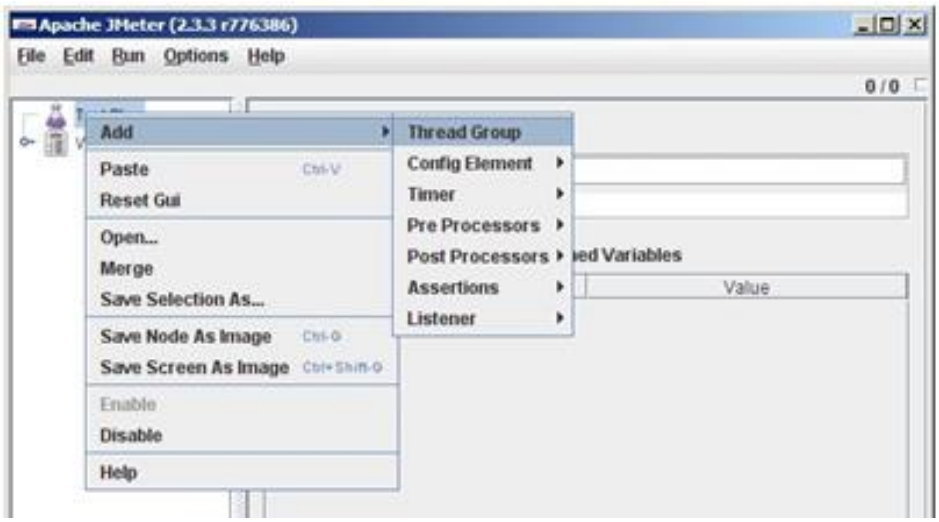

Рис. 2.2. Основні функції Jmeter

2.3.3. Запис HTTP трафіку

За допомогою елемента HTTP Proxy Server відбувається запис тестових сценаріїв користувача у JMeter.

Для роботи з цим елементом потрібно:

1. З елемента "WorkBench" вибрати «Add» -> «Non-Test Elements» -> «HTTP Proxy Server» (див. малюнок 2.3)

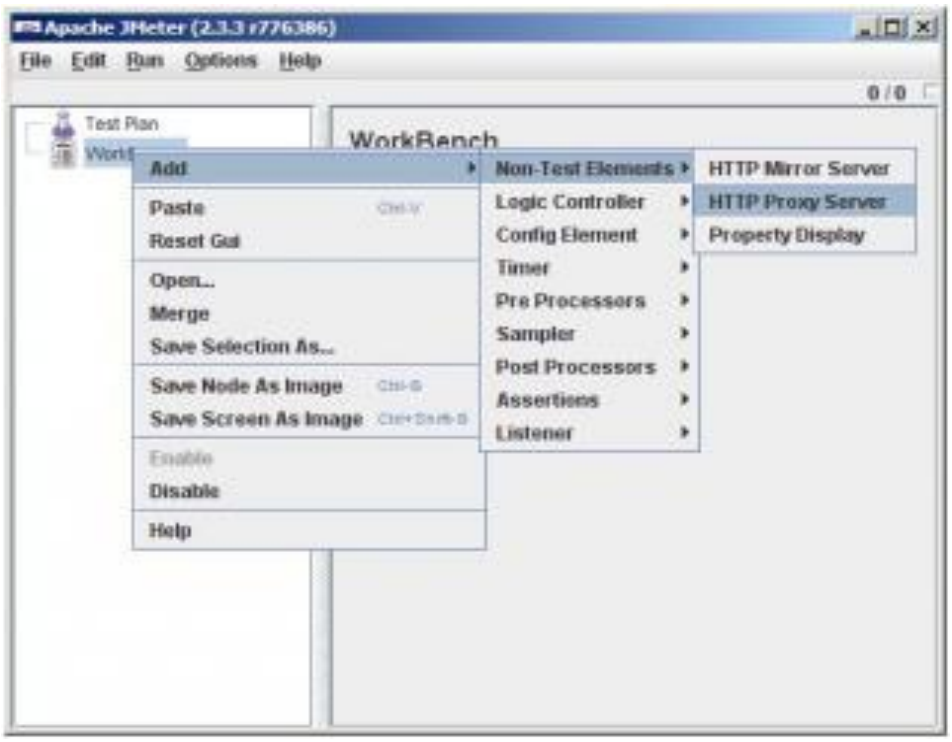

Рис. 2.3. Структура “Non-Test Elements” в Jmeter

2. В полі «Port» вказати порт проксі-сервера

3. За допомогою регулярних виразів (RegEx), встановити фільтр на записувані запити в наступні секції: URL PatternstoExclude – ті запити, що не включаються в сценарій; URL PatternstoInclude - що включаються в нього. Наприклад: фільтр «. * \. Gif», встановлений у секції URL PatternstoExclude, усуне зі сценарію запити з картинками в форматі gif (див. рисунок 2.4)

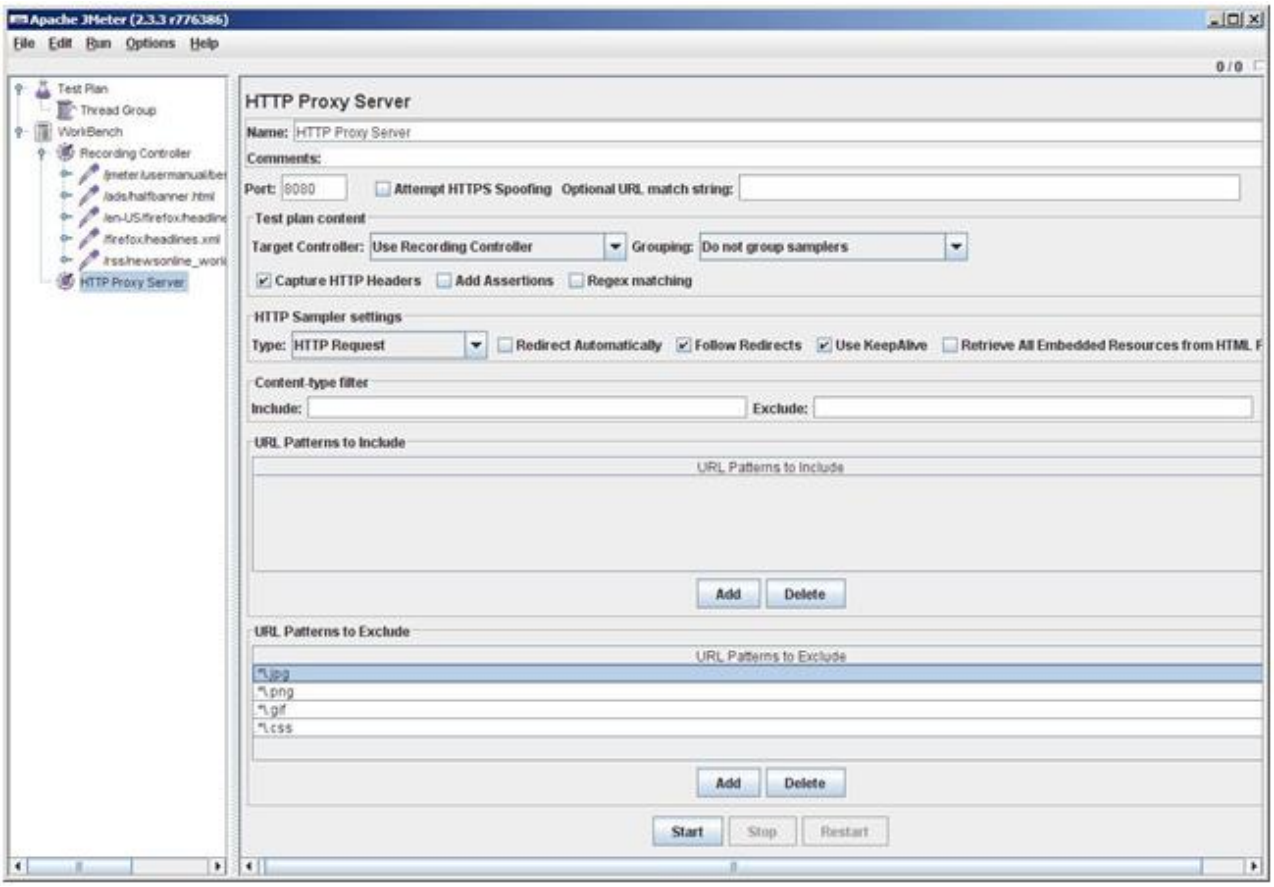

Рис 2.4. Структура “HTTP Proxy Server”

4. Натиснути кнопку «Start»;

5. В браузері встановити налаштування проксі-сервера, як і для HTTP Proxy Server; Наприклад, вибрати Tools -> Internet Options ->Connections ->Lansettings в Internet Explorer (див. рисунок 5)

Рис. 2.5. Налаштування браузеру

6. Пройти тест
7. Повернутися в JMeter і натиснути кнопку «Stop»

2.3.4. Створення HTTP запитів

Елемент «HTTP Request» потрібен для роботи з HTTP трафіком в JMeter. Для створення цього елемента потрібно з контекстного меню елемента «ThreadGroup» вибрати «Add» -> «Sampler ->«HTTP Request» (див. рисунок 2.6). У налаштуваннях запиту необхідно вказати наступні основні параметри:

- Name – Ім’я запиту;
- Method - Метод передачі даних;
- Server nameor IP - Адреса веб-сервера (IP-адресу або URL);
- Path - Шлях для запуску файла на сервері;

- Parameters – Значення переданих параметрів;
- Portnumber - Порт веб-сервера (стандарт 80);
- Protocol (defaulthttp) - Протокол (стандартHTTP).

Щоб здійснити відправку файлу в запиті потрібно вказати шлях до файлу в секції «Sendfileswithrequest» та його ім'я (див. рисунок 2.7).

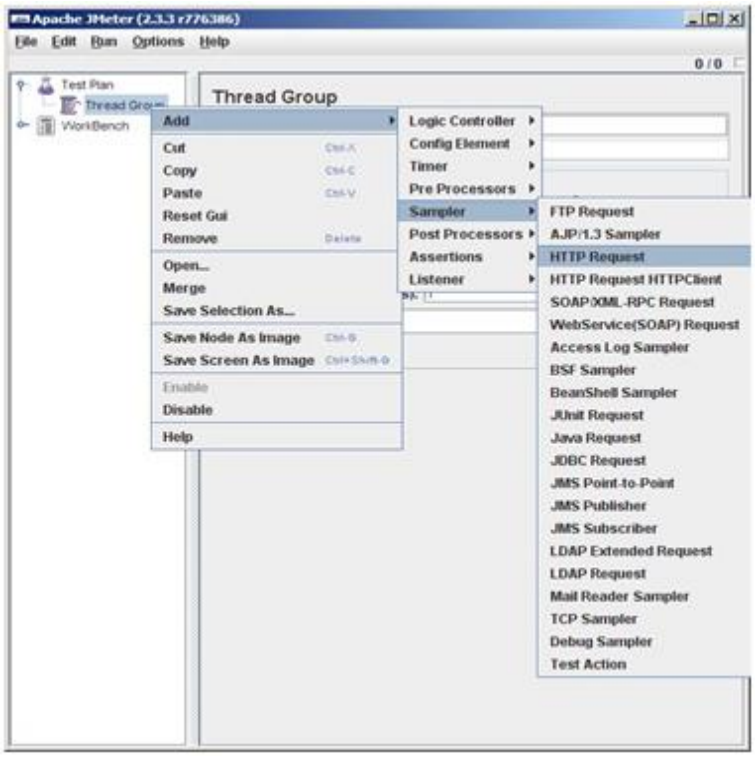

Рис. 2.6. Структура “Sampler”

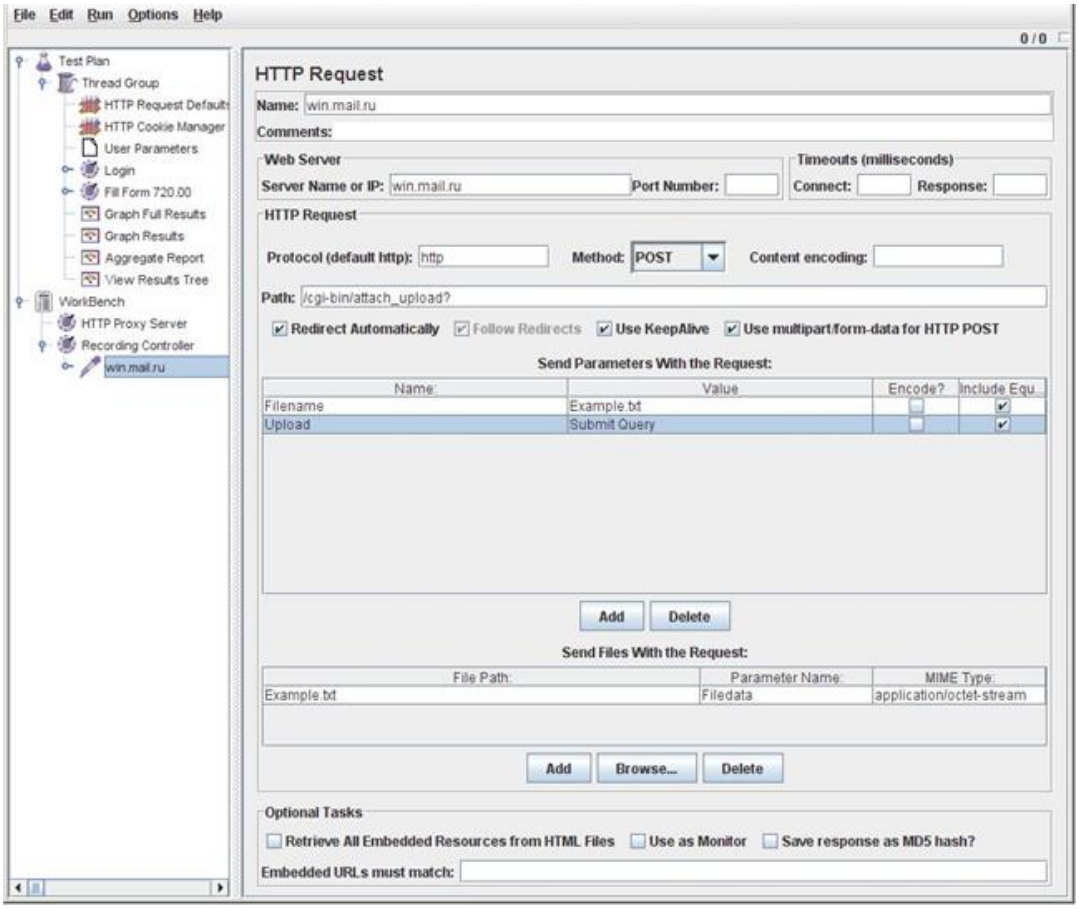

Рис. 2.7. Структура HTTP Request

2.3.5. DataDriven запитів

JMeter має елемент «UserDefinedVariables», який виконує функцію за допомогою якої існує можливість вказати параметри запитів через змінні, які визначаються самим користувачем.

Для роботи з ним необхідно:

1. З контекстного меню елемента «ThreadGroup» вибрати «Add» -> «ConfigElement» -> «UserDefinedVaiables» (див. рисунок 2.8)

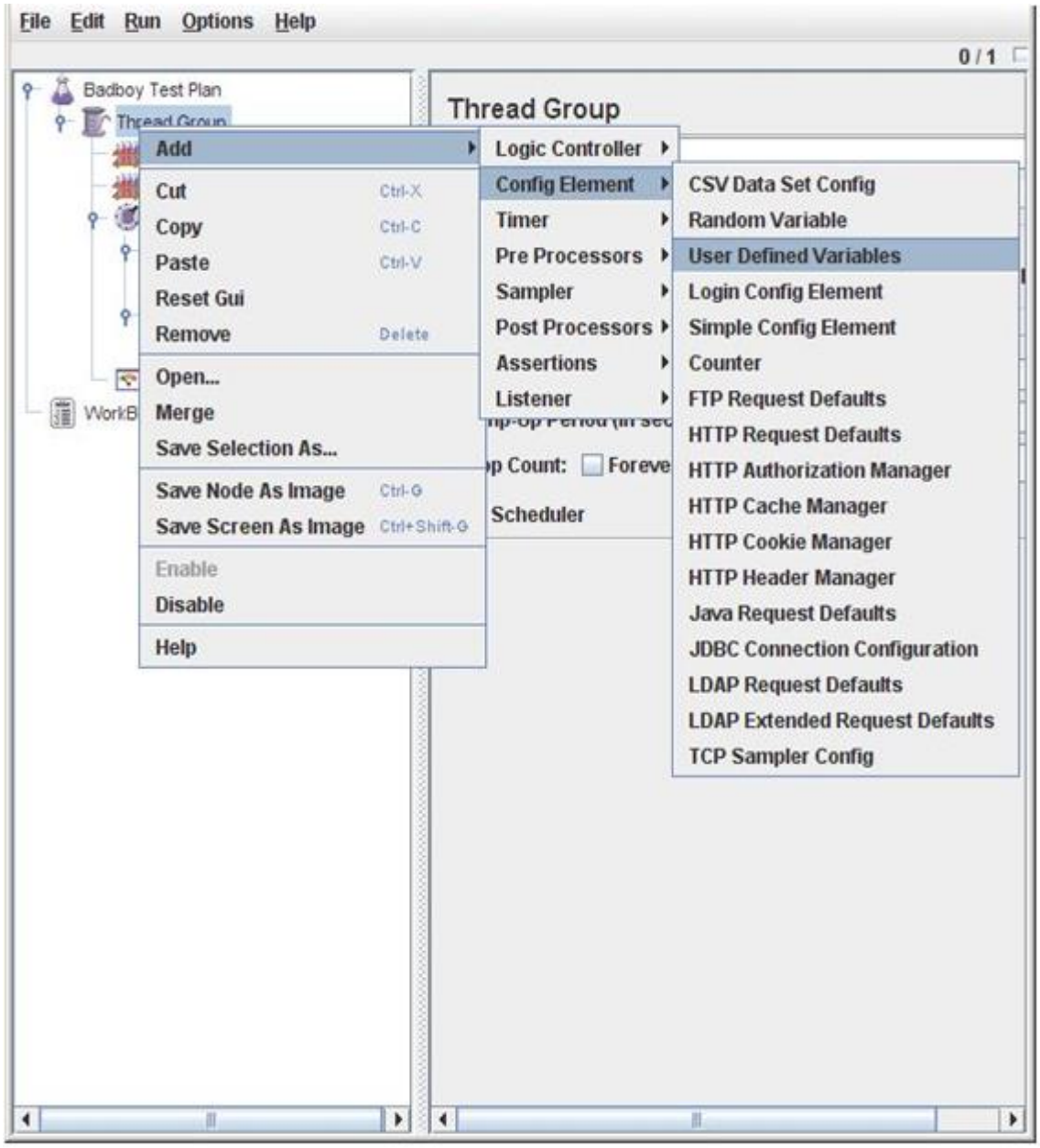

Рис. 2.8. Структура ConfigElement

2. У формі натиснути кнопку «Add»
3. Вказати значення та назву змінної в полях «Value» та «Name» (див. рисунок 2.9)

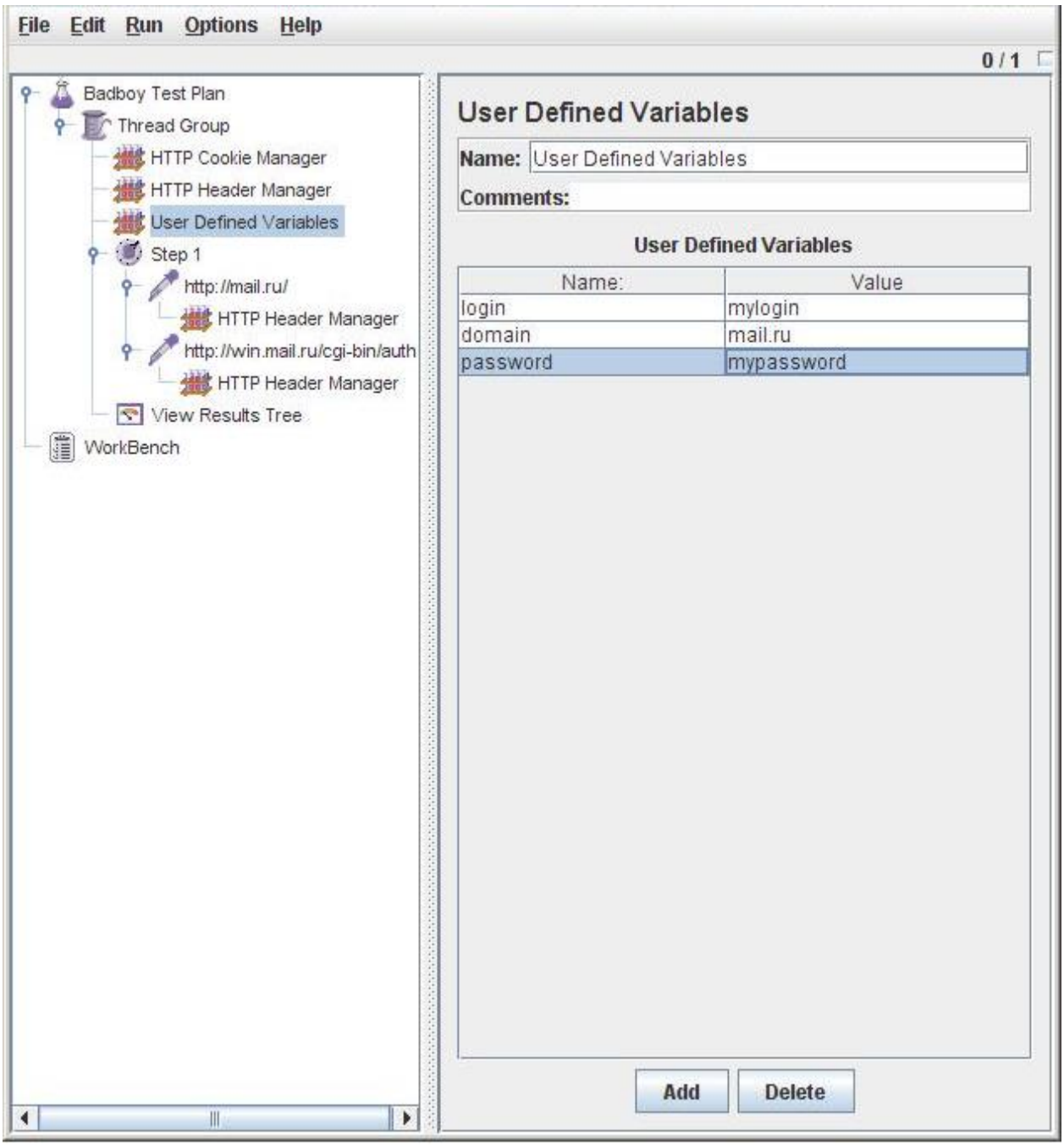

Рис. 2.9. Вигляд UserDefinedVariables

4. Створити елемент «HTTP Request» і призначити потрібні налаштування

5. Призначити в значенні параметра запиту змінну. Змінні призначаються у форматі $ {VariableName} (див. рисунок 2.10)

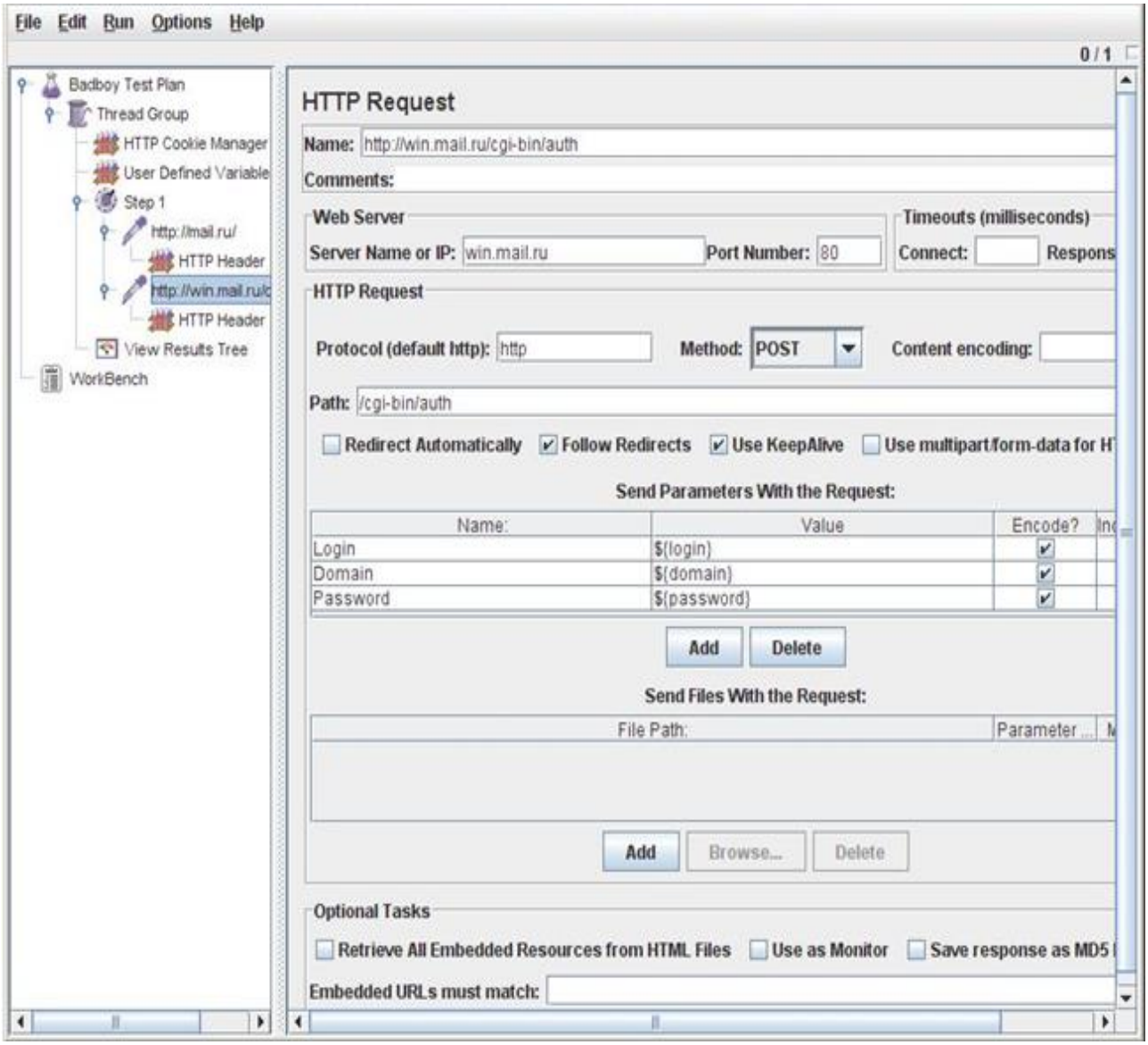

Рис 2.10. Приклад HTTP запиту

2.3.6. Передача значення змінної з файлу

Для зчитування та зберігання змінної з файлу найбільш зручно використовувати елемент «CSV DataSetConfig». Цей елемент використовується в редагуванні рядків з файлу, для читання та відділення їх на змінні.

Щоб почати роботу з «CSV DataSetConfig» необхідно виконати наступні кроки:

1. Потрібно створити файл (приклад, c: \ csvdata.csv) та вказати в цьому файлі значення змінних. (див. рисунок 2.11)

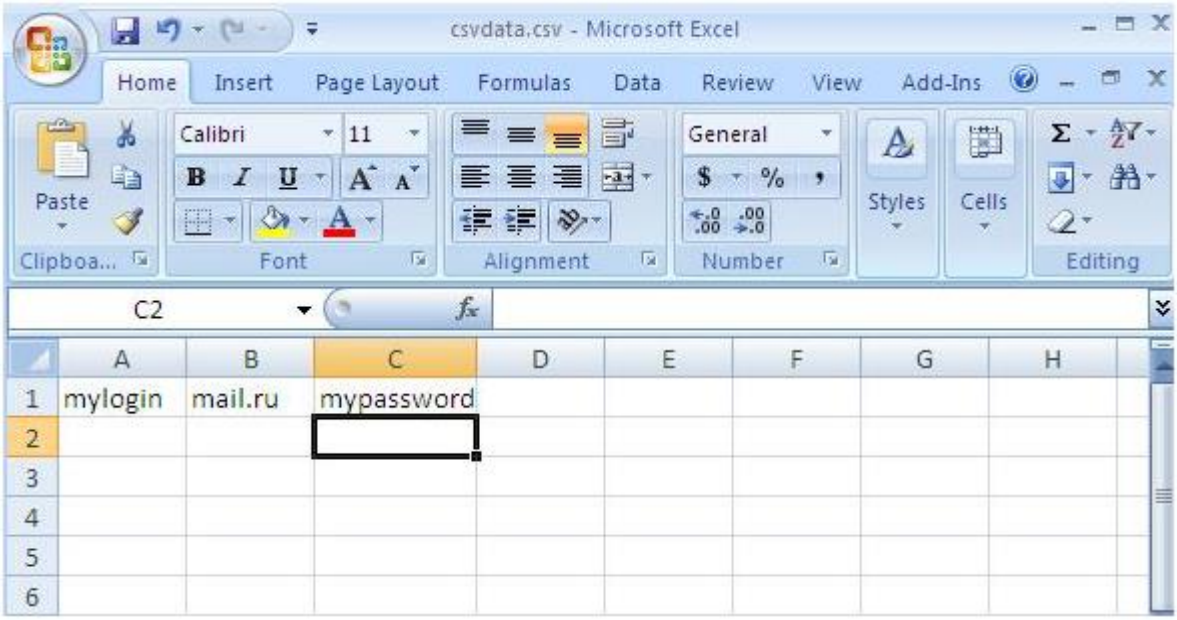

Рис. 2.11. Формат заповнення Microsoft Excel

2. Далі потрібно відкрити JMeter та обрати «Add» -> «ConfigElement» -> «CSV DataSetConfig» з контекстного меню елемента «ThreadGroup» (див. рисунок 2.12).

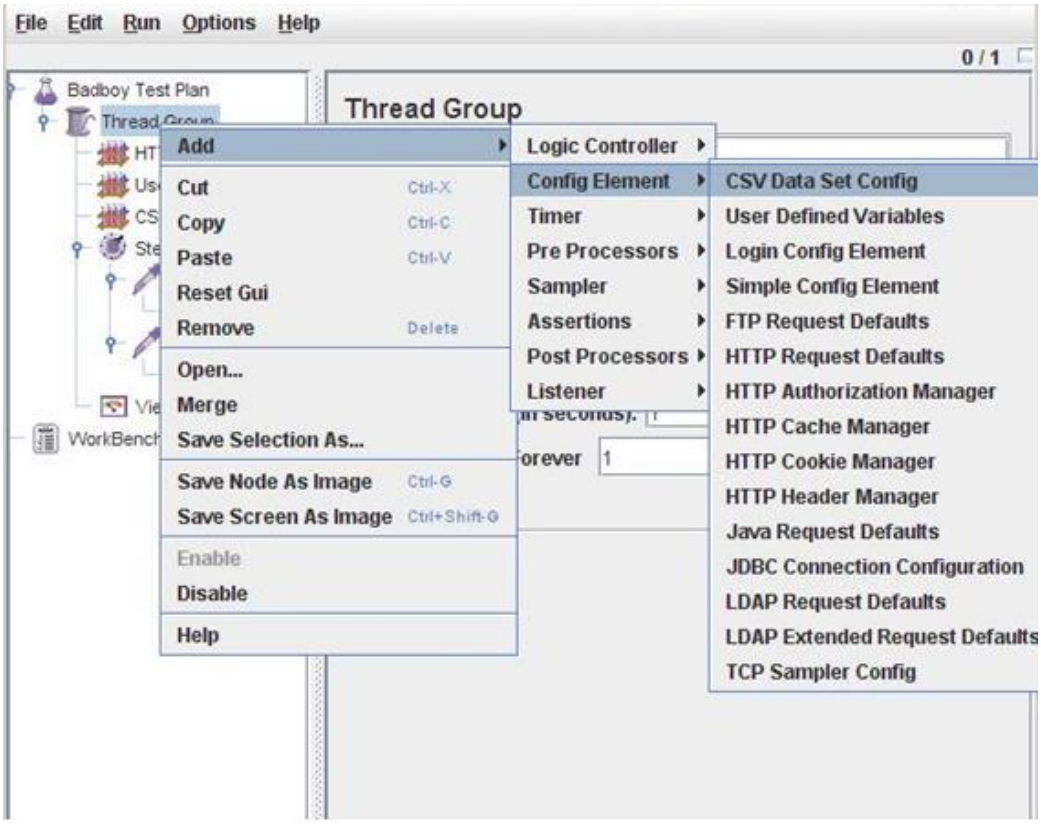

Рис. 2.12. Розташування елемента CSV DataSetConfig

3. У формі, що з'явилася, потрібно знайти елемент «CSV DataSetConfig». На даному етапі необхідно заповнити поля, як малюнку 13: «FileName» - ім'я файла (c: \ csvdata.csv); «Delimiter (use't'fortab)»- символ роздільника (у нашому випадку це символ «,»); «VariableNames (comma-delimited)» - назви змінних, розділених символом «, ».

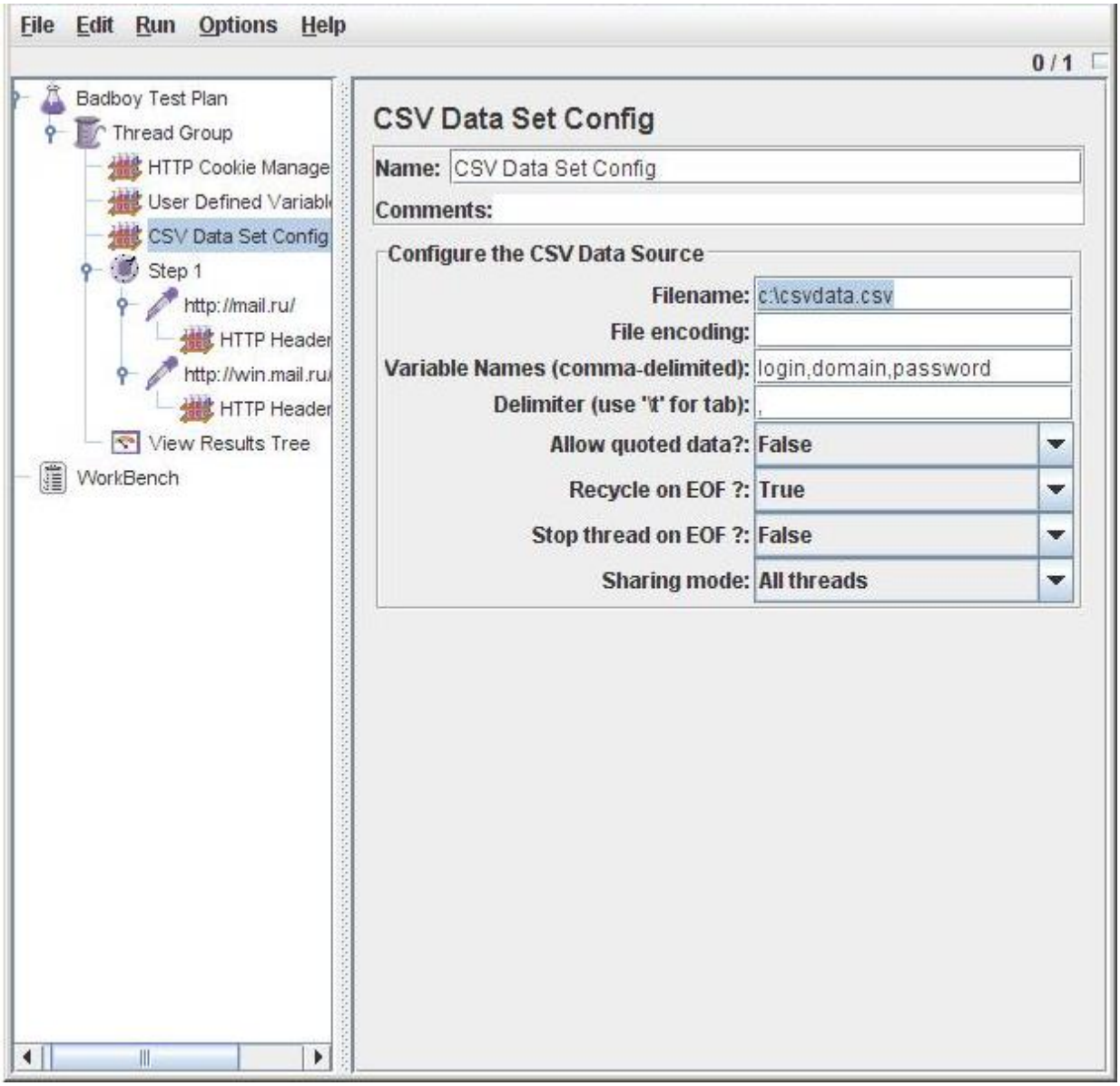

Рис. 2.13. Структура CSV DataSetConfig

4. Потрібно створити елемент «HTTP Request», вказати потрібні значення.

5. Позначити змінні в значенні параметрів запиту. Змінні вказуються у форматі $ {VariableName} (див. малюнок 14).

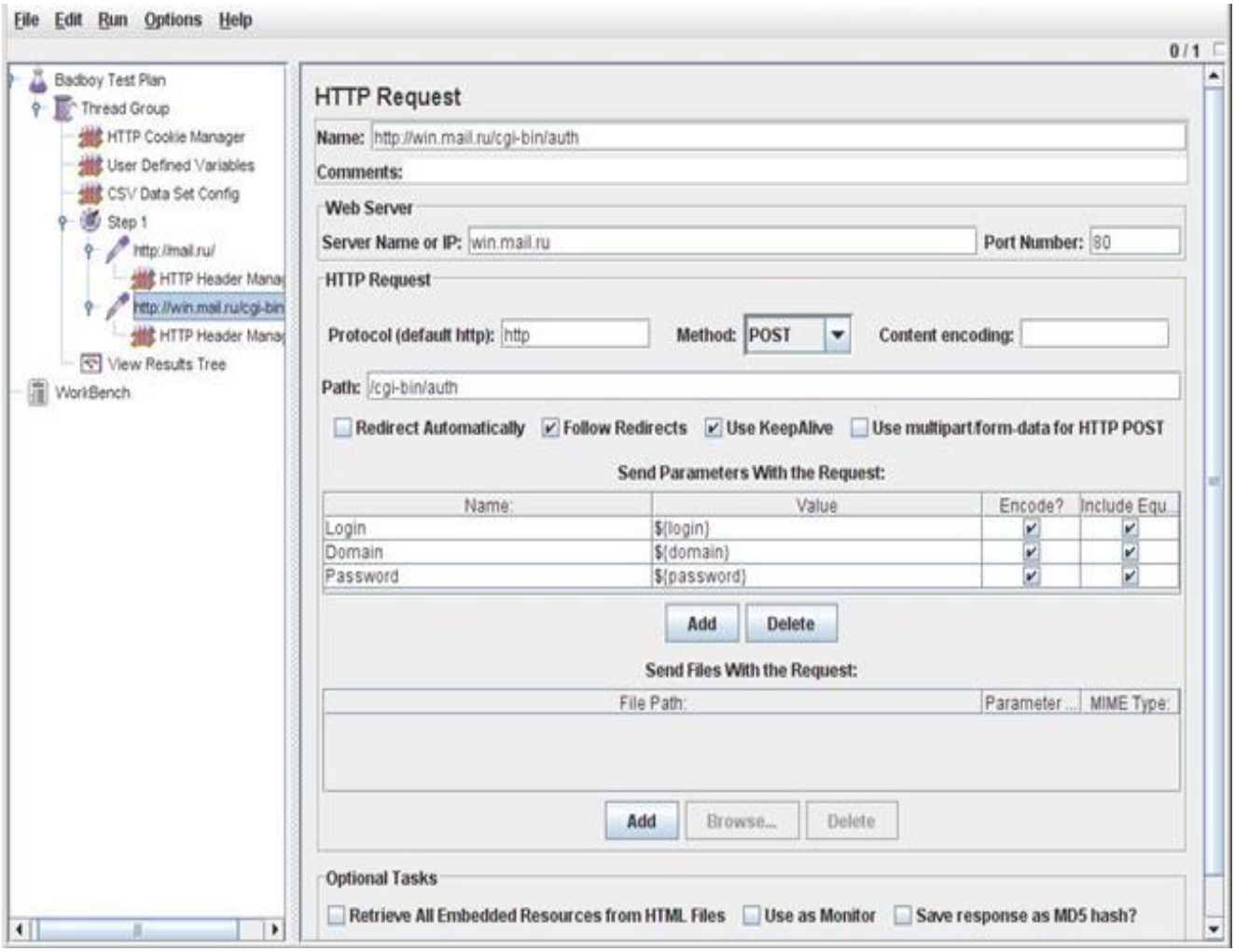

Рис. 2.14. Форма HTTP Request

2.3.7. Створення налаштувань "Default" для HTTP запитів

Для підвищення ефективності в введенні дублювання даних (приклад, якщо з’являється необхідність тестування декількох сервісів, що знаходяться на одному сайті) маємо можливість вказати налаштування "Default". Для цього необхідно додати елемент HTTP RequestDefaults в групу потоків «ThreadGroup» і з контекстного меню вибрати «Add» -> «ConfigElement» -> «HTTP RequestDefaults» (див. рисунок 2.15).

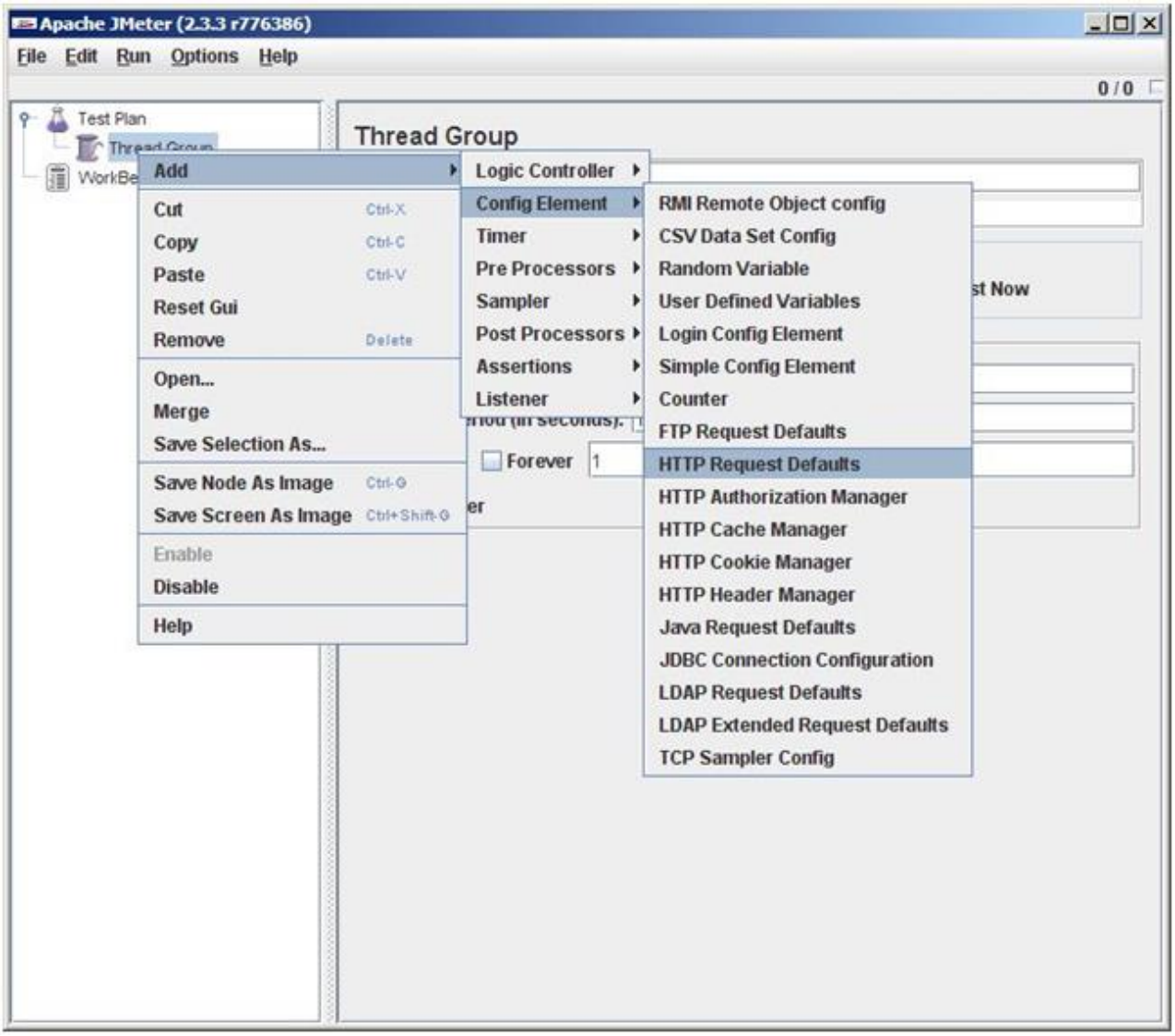

Рис. 2.15. Розташування HTTP RequestDefaults

«HTTP RequestDefaults» представлений на рисунку 2.16. У параметрі "Default" можна вказати:

- Path - Шлях до запуску файлу на сервері;
- PortNumber Порт веб-сервера (стандарт 80);
- Protocol (defaulthttp) - Протокол (стандарт HTTP);
- Server Nameor IP - Адреса веб-сервера (IP-адресу або URL);

– SendParameterswithrequest - Передані значення параметрів.

Усі наступні елементи «HTTP Request» отримають налаштування «Default», тому потрібність вказувати для кожного наступного запиту ці налаштування зникає.

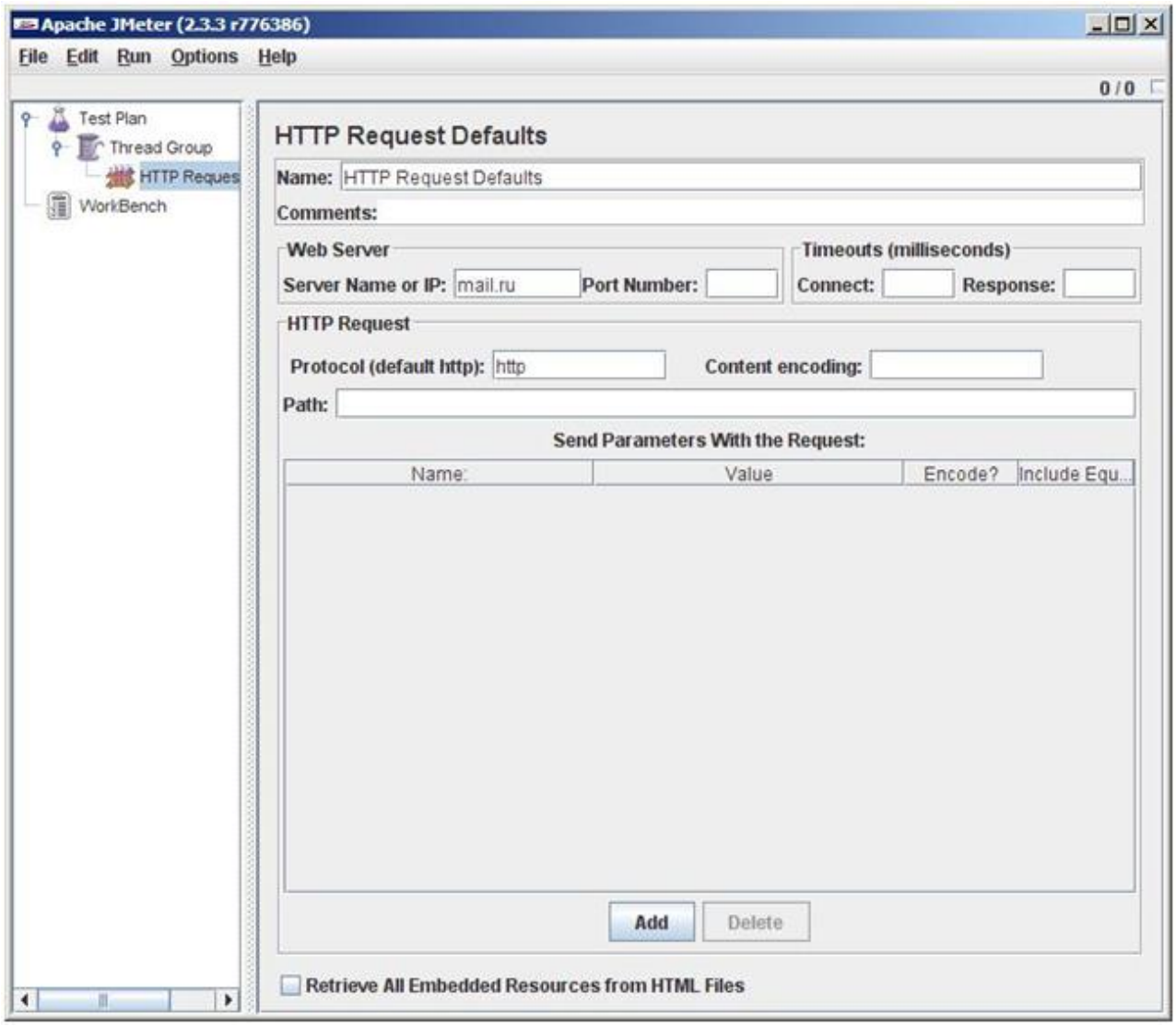

Рис. 2.16. Структура HTTP RequestDefaults

2.3.8. Створення HTTPS запитів

Проксі сервер JMeter не має можливості обробити трафік через HTTPS протокол. Для цього є можливість використати сторонній інструмент, наприклад Badboy. Для цього потрібно вказати сценарій в BadBoy і використати його функцією «File»> «ExporttoJMeter ...» (див. рисунок 2.17).

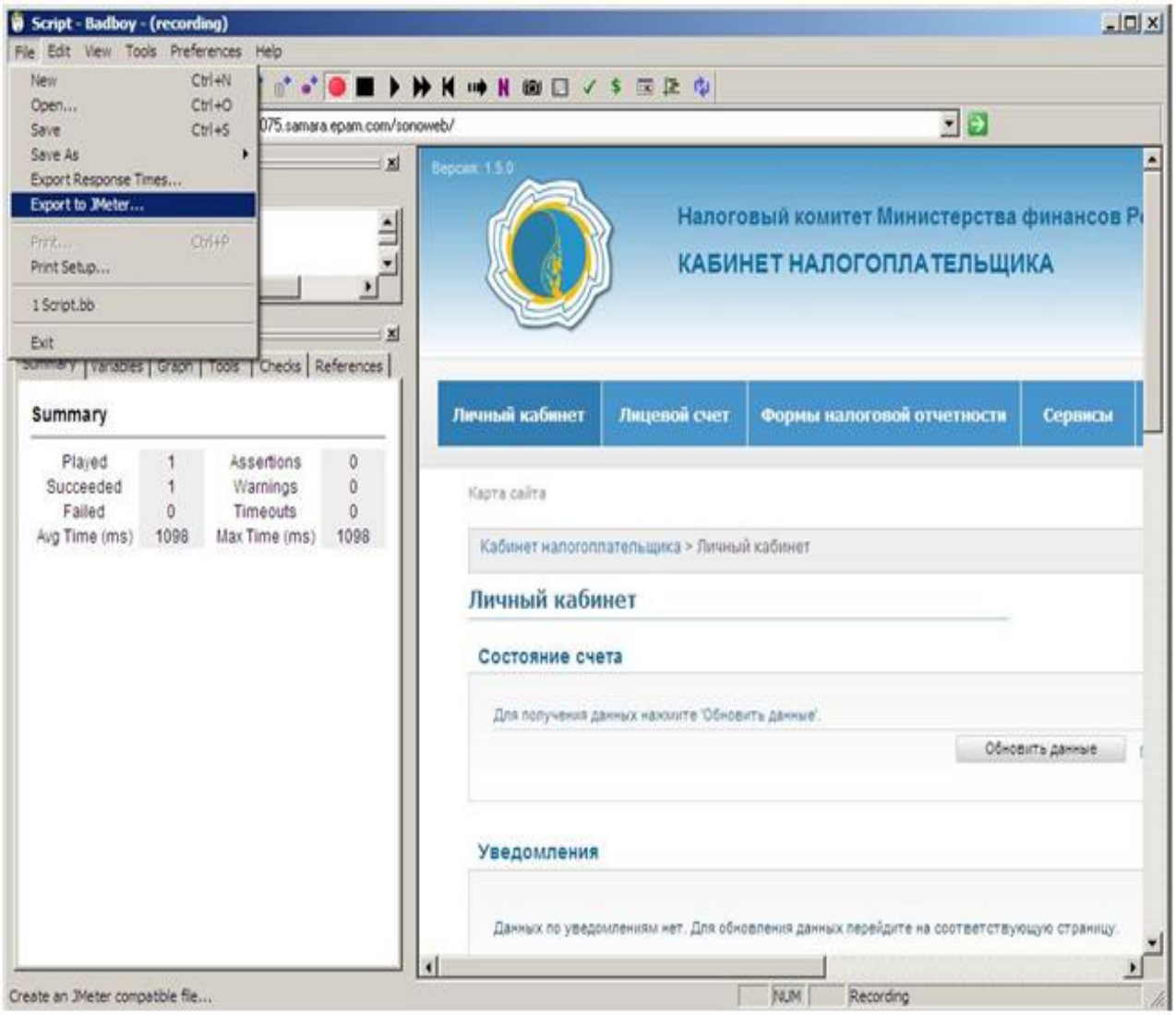

Рис. 2.17. Контекстне меню ExporttoJmeter

Але важливо запам'ятати, існує чимала кількість складових сценарію Badboy, що не можуть переноситись в JMeter. Спочатку, це процеси, які вказані в режимі Navigate, пізніше робота з розкладом, робота з завантаження даних з excel-файлів і т.д. Ці процеси потрібно виконувати в Jmeter поступово.

Після цього експортований проект потрібно відкрити в JMeter за допомогою головного меню «File» -> «Open» (див. рисунок 2.18).

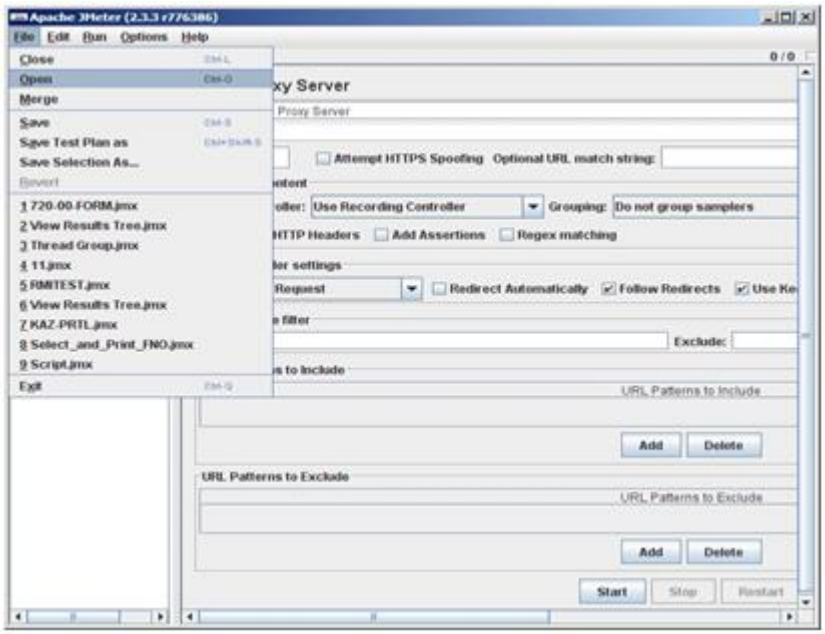

Рис. 2.18. Структура File

2.3.9. Робота з сертифікатами безпеки

Можемо скористатися функцією SSL Manager, вона потрібна для роботи з сертифікатами безпеки в JMeter:

1. Необхідно обрати пункт головного меню «Options»> «SSL Manager" (див. рисунок 2.19)

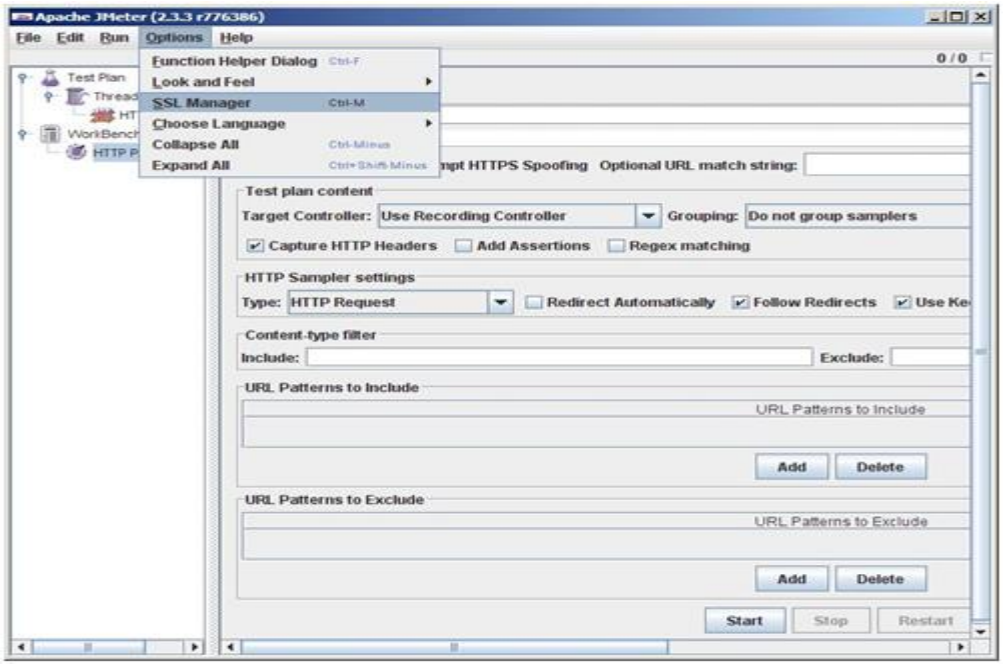

Рис. 2.19. Контекстне меню SSL Manager

2. Далі потрібно обрати файл сертифіката в форматі p12 (див. малюнок 2.20)

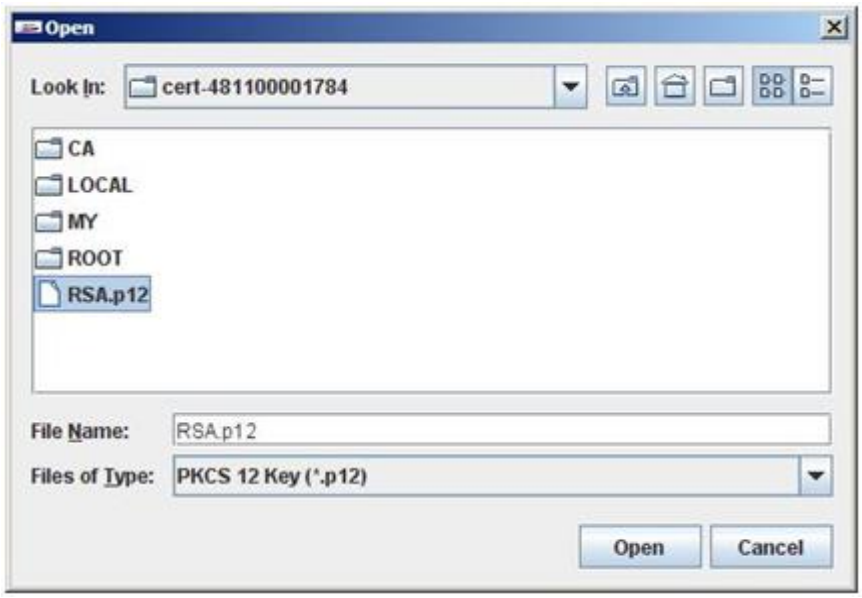

Рис. 2.20. Контекстне меню Openfile

Далі можемо один раз вказати пароль і шлях до сертифіката в файлі конфігурації system.properties, щоб оминути необхідність обирати файл сертифіката при кожному запуску JMeter.

Потрібно зробити наступні дії:

1. В блоці bin відкрити файл system.propeties.

2. Змінити коментар з рядків javax.net.ssl.keyStorePassword і javax.net.ssl.keyStore, видаливши символ «#» на початку рядка.

3. Вказати в рядку javax.net.ssl.keyStorePassword пароль до сертифіката, а в рядку javax.net.ssl.keyStore його шлях .

Наприклад:

 # Locationofthekeystore

javax.net.ssl.keyStore = C: \ \ certificates \ \ cert-481100001784 \ \ RSA.p12

 # Thepasswordtoyourkeystore

javax.net.ssl.keyStorePassword = 123456

2.3.10. Розподілене тестування

Однією з важливих функцій JMeter є засіб для розподіленого управління - коли управління копіями JMeter виконується з одного центрального комп'ютера, але Jmeter генерує трафік на декількох машинах.

Для налаштування розподіленого тестування потрібно пройти наступні кроки.

Для клієнта:

– У файлі <jmeter-home> / bin / jmeter.properties вказати хости в рядку remote_hosts.

– Перезапустити JMeter.

– Обрати пункт меню «Run» -> «RemoteStart» або «Run» -> «RemoteStartAll» (див. рисунок 2.21). «RemoteStartAll» - запускає усіх зазначених агентів відразу; «RemoteStart» - запускає одного вибраного агента .

Для серверів:

– Запустити файл <java-home> / bin / rmiregistry.exe.

– В файлі <jmeter-home> / bin / jmeter.properties змінити коментар, видаливши символ «#», з рядка server.rmi.create = false.

– Запустити <jmeter-home> / bin / jmeter-server.bat.

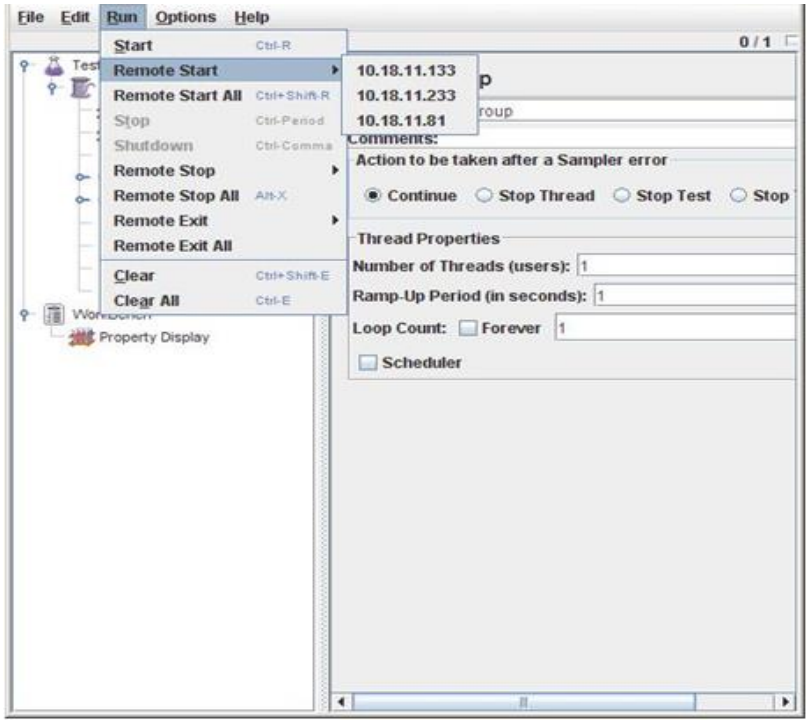

Рис 2.21. Контекстне меню RemoteStart

РОЗДІЛ 3
СТВОРЕННЯ СЦЕНАРІЇВ ТЕСТУВАННЯ

3.1. Тестування на основі типів вимог

Види моделі вимог (тобто інтерфейси, класи та перерахування) описують відносини й поняття, які застосовуються користувачами для надання відомостей про свій бізнес та його аналіз. Типи, що мають відношення тільки до внутрішньої будови системи не входять сюди.

Створювати тести необхідно з використанням цих видів вимог. Такий підхід допомагає зв’язати зміни в тестах зі змінами у вимогах. За допомогою такого підходу є можливість обговорювати очікувані результати та усі тести відразу з зацікавленою особою або користувачем. Це дає нам змогу вирішувати потреби користувачів не під час робочого процесу; крім того, так можна уникнути випадкових вад у розробці.

Для проведення таких тестів вручну необхідно використовувати словник моделі вимог в скриптах тесту. При проведенні автоматичних тестів такий підхід припускає створення функцій методів доступу і засобів оновлення для зв'язку моделі вимог з кодом та використання схем класів вимог до якості фундаменту тестового коду.

Наприклад, модель вимог буде включати типи "пункт меню", "меню", "замовлення" та зв'язки між ними. Ця модель представляє відомості, які обробляються і зберігаються системою, але не показує усі складнощі її реалізації.

В системі існує декілька різних реалізацій кожного типу - в API, базах даних та інтерфейсах користувача. У розподіленій системі може бути декілька видів кожного примірника, який буде одночасно зберігатись в різних складових системи.

Через звичайний API додаток ми можемо отримати доступ до багатьох методів та засобів поновлення. Проте потрібно буде провести деякі операції перед початком проведення тестів. Ці додаткові варіанти поновлення системи та методи доступу до них називають інструментарієм тесту. Відповідальність за надання цих засобів та методів лежить на розробниках системи, тому що вони залежать від внутрішньої будови системи, , а тест-інженери використовують модель вимог та з її допомогою створюють код тестів.

При складанні автоматичних тестів можемо використовувати звичайні тести для сортування засобів обновлення та методів доступу.

3.2. Сценарії тестування

З архітектурою клієнт-сервер для автоматизованого тестування додатків були визначені основні методи:

– AddParameter – метод, що додає параметр до файлу. Зараз, цей метод став дуже актуальним, оскільки більше додатків, написаних через архітектуру клієнт-сервер, використовують так звані конфіг файли.

– CompareID – порівняння на рівність значень конфіг файлу та тих, що зберігаються в самій базі даних.

– ExecuteQueryFromFile – виконання SQL скрипту із файлу.

– ExecuteCommand – дає дозвіл на виконання консольних команд через рядок в додатку Jmeter. Оскільки всі наші тести будуть розміщені в додатку Jmeter то це розширення нам просто необхідне. Наприклад зможемо запустити або зупинити роботу сервера.

– FileClear – очищає вказаний файл.

– FileExist – перевірка на присутність файлу.

– LogContain – перевірка файлу на присутність зазначеного тексту.

– LogHasData – перевірка файлу на знаходження у ньому даних.

– SelectAndCompareValue – метод за допомогою якого ми можемо порівняти (дістати) значення з бази даних і з вказаним значенням.

– SetUpDMSFile – змінює значення в файлі.

– TestDBWithXML – друкування результатів в CSV файл, порівняння файлу XML та бази даних.

3.3. Основні принципи мови реалізації та інтеграція додатків в Jmeter

Програмний продукт - це послідовність директив, написаних мовою програмування. Потрібно розуміти, що людська мова і мова програмування, це дві різні речі. Мови програмування мають дуже багато правил, яких необхідно дотримуватись. Цей набір правил, називають синтаксисом мови. Регламент синтаксису визначає словник мови, як конструюється програма з використанням таких структур, як, наприклад, функції, цикли, розгалуження. Тільки синтаксично вірна програма може бути запущена. Програмні продукти з дефектами будуть усунені. Програміст повинен створити

програму, яка не матиме жодного синтаксичного дефекту, що виконати усю роботу вірно, йому необхідно детально знати усі правила синтаксису. Програма повинна бути осмислена, тобто виконувати логічну послідовність дій, а не тільки дотримуватись синтаксичної коректності. Семантика програми – це її сенс. Семантично вірна програма вважає саме тоді, коли вона працює таким чином, як це бачив програміст на усіх етапах її створення.

Створимо найлегшу програму. Сам процес складається з трьох основних частин:

– написання тексту мовою програмування програмістом;
– компіляція введеного коду;
– виконання скомпільованого коду.

Найлегший код програми буде виглядати так:

public class HelloWorld {

// A program to display the message // "Hello, World!" on standard output

public static void main (String [] args) {System.out.println ("Hello, World!");

}

}

Результат цього повідомлення здійснюється за допомогою команди

System.out.println ("Hello, World!");

Ця програма виводить повідомлення "Hello, World!"

Це приклад виклику функції. Тут викликана функція System.out. Println. Функція – це комплект інструкцій, які об’єднуються і утворюють одне ціле під спільним ім'ям. В цьому прикладі ми використовуємо вбудовану функцію. Вбудована функція – це функція, яка є заздалегідь визначеною і являється невід’ємною частиною програмної мови. Програмний продукт має

роз'яснення(коментарі). При компіляції програми коментарі ігноруються. Рядки коментарів починаються з символів // і закінчуються в кінці рядка. Все, крім коментарів, підпорядковане правилам синтаксису мови. Багаторядкові коментарі укладаються між символами / * і * /. Для початку вказуємо клас з ім'ям HelloWorld у першому рядку нашої програми. Потрібно, що не всякий клас є програмою. У класі необхідно додати функцію main для того, щоб клас став самостійною програмою вона визначається таким чином:

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

Інтерпретатор звертається до функції main, це необхідно для звернення до інтерпретатора Java запуску програми. Слово public означає, що функція викликається не з самої програми, а ззовні. Оскільки функція викликається інтерпретатором, то на це необхідно звертати увагу. Також між дужками розташовується комплект інструкцій, які будуть викликані і виконані при зверненні до функції.

Аплети створюються по-іншому:

```
public class InterestIConsole extends ConsoleApplet {
protected String getTitleO {
return "Sample program V'lnterestlV";
}
protected void program () {
double principal; / / вкладена сума
double rate; / / річний відсоток
double interest; / / відсоток за рік
```

*/ * Обчислення * / principal = 17000; rate = 0.07;*

*interest = principal * rate; / / відсоток principal = principal + interest; / * виведення результату * / console.put ("The interest earned is $"); console.putln (interest);*

console.put ("The value of the investment after one year is $"); console.putln (principal);

}

Також можемо використовувати наступну програму, що міститься у файлі :

ConsoleApplet.java:

*impo гt j ava.awt. *;*

*impo rt java.awt.event. *;*

public class ConsoleApplet extends java.applet.Applet

implements Runnable, ActionListener {protected String title = "Java Console I / O"; protected String getTitleO {return title;

}

protected ConsolePanel consoles-protected void program () {

console.putln ("Hello, World!");

}

private Button runButton;

private Thread programThread = null;

/ / Потік run () private boolean programRunning = falser-private boolean firstTime = true; / / false - якщо програма вважає перший раз

public void run () {/ / запуск programRunning = true;

program ();

programRunning = false; stopProgram ();

}

synchronized private void startProgram () {runButton.setLabel ("Abort Program"); if (! firstTime) {console.clear (); try {Thread.sleep (300);} / / затримка перед перезапуском програми

catch (InterruptedException e) {}

}

firstTime = false; programThread = new Thread (this); programThread.start ();

}

synchronized private void stopProgram () {if (programRunning) {programThread.stop (); try {programThread.join (1000);} catch (InterruptedException e) {}

}

console.clearBuffers (); programThread = null; programRunning = false; runButton.setLabel ("Run Again"); runButton.requestFocus ();

}

public void init () {

setBackground (Color.black);

setLayout (new BorderLayout (2,2)); console = new ConsolePanel (); add ("Center", console);

Panel temp = new Panel ();

temp.setBackground (Color.white); Label lab = new Label (getTitle ()); temp.add (lab);

lab.setForeground (new Color (180,0,0)); add ("North", temp);

runButton = new Button ("Run the Program");

temp = new Panel ();

temp.setBackground (Color.white);

temp.add (runButton);

runButton.addActionListener (this);

add ("South", temp);}

public Insets getlnsets () {return new Insets (2,2,2,2);

}

public void stop () {

if (programRunning) {stopProgram (); console.putln ();

*console.putln ("*** PROGRAM HALTED");*

}

}

synchronized public void actionPerformed (ActionEvent evt) {if (programThread! = null) {stopProgram (); console.putln ();

*console.putln ("*** PROGRAM ABORTED BY USER");*

}

else

startProgram ();

}

}

Примітивні і змінні типи:

Усі імена мають основну роль у програмуванні. З їх допомогою називаються різні речі. При використанні імен необхідно знати семантику та синтаксис мови. Ім'я - це послідовність символів. Ім'я має складатися з цифр, букв, символів підкреслення (_) і починатися з якоїсь літери. Наприклад:

– n;

– n;

– rate;

– x15;

– HelloWorld;

– quite_a_long_name.

Малі та заголовні літери в імені вважаються різними. Імена HelloWorld, helloworld, heiioworid, hEiioWorLD - різні імена. Деякі імена є установленими і виконують спеціальні функції, вони не можуть бути використані програмістом для інших цілей, наприклад:

– public;
– static;
– class;
– else;
– while;
– if.

та інші імена.

Значення присвоюється змінної за допомогою оператора присвоювання у вигляді:

variable = expression;

Наприклад:

rate = 0.08;

interest = rate * principal;

Змінна служить для зберігання тільки одного типу даних. Існують стандратні типи, вбудовані в Java:

– short;
– byte;
– long;
– int;
– double;
– boolean;
– char;
– float.

Тип short відповідає двом байтам, проміжок від -32 768 до 32 767. Тип long відповідає 64 бітам, проміжок від -9 223 372 036 854 775 808 до 9 223 372 036 854 775 807. Тип int відповідає 32 бітам, проміжок від -2147483648 до 2147483647. Тип char займає два байти пам'яті. Тип float займає 4 байти пам'яті, максимальне значення приблизно 1038. Ім'я константи називається літералом. Для логічного літерала існує два значення true і false.

Приклад програми, в якій використовуються змінні і оператори присвоювання:

public class Interest {

*public static void main (String [] args) {/ * оголошення змінних * / double principal; / / вкладена сума*

double rate; / / річний відсоток

double interest; / / нарахування за рік

*/ * Обчислення * / principal = 17000; rate = 0.07;*

*interest = principal * rate; principal = principal + interest;*

*/ * Виведення результату * / System.out.print ("The interest earned is $"); System.out.println (interest);*

System.out.print ("The value of the investment after one year is $");

System.out.println (principal);}

}

Об'єкти, рядки, функції:

Річ у тому, що string - це об'єкт, тому існує принципова відмінність між примітивними типами та типом string.. У нашому прикладі буде використаний клас Textio.

Класи в Java виконують декілька важливий функцій. Перша полягає в тому, що класи об'єднують в собі функції та змінні, які знаходяться в цьому

класі. Функції і змінні являються статичними членами класу, наприклад, функція main - статична. Слово static використовується для визначення таких функцій. Також класи потрібні для опису об’єктів. Клас - це тип, а об'єкт - це його значення, string - це назва класу. Наприклад, "Hello, World" - написаний в форматі типу string.

Ми знаємо математичні функції, наприклад, квадратний корінь. У мові Java існує аналогічна функція, Math.sqrt. Ця функція - статичний член класу Math. Якщо x - це числове значення, то Math.sqrt(x) - це теж значення, рівне квадратному кореню від x. Щоб вивести це значення на екран, ми прописуємо наступну команду: System.out.print (Math.sqrt (x));

Можемо використати оператор присвоєння для ініціалізації змінної:

lengthOfSide = Math.sqrt (x);

Ще приклад математичної функції:

Math.abs (x)

Ця функція обчислює модуль (абсолютне значення). Серед математичних функцій є й інші, наприклад, Math, sin(x), Math.cos(x), Math.tan(x), Math.asin(x), Math.acos(x), Math.atan(x) , Math.exp(x), Math, log(x), Math.pow(x, y), Math, floor(x), Math, random().

Ось приклад програми, яка використовує ці функції:

public class TimedComputation {

public static void main (String [] args) {

long startTime; // стартовий час в мілісекундах

long endTime; // час при завершенні обчислень, в мілісекундах double time; // Різниця часів, в секундах. startTime = System.currentTimeMillis ();

double width, height, hypotenuse; // сторони трикутника width = 42.0; height = 17.0;

*hypotenuse = Math.sqrt (width * width + height * height);*

System.out.print ("A triangle with sides 42 and 17 has hypotenuse");

System.out.printIn (hypotenuse);

*System.out.printIn ("\ nMathematically, sin (x) * sin (x) +" + "cos (x) * cos (x) - 1 should be 0."); System.out.println ("Let's check this for x = 1: ");*

*System.out.print ("sin (l) * sin (l) + cos (1) * cos (1) - 1 is"); System.out.printIn (Math.s in (1) * Math.s in (1)*

*+ Math.cos (1) * Math.cos (1) - 1); System.out.println ("(There can be round-off errors when" + "computing with real numbers!)");*

System.out.print ("\ nHere is a random number:");

System.out.printIn (Math.random ());

endTime = System.currentTimeMillis ();

time = (endTime - startTime) / 1000.0;

System.out.print ("\ nRun time in seconds was:");

System.out.printIn (time);

}

}

Величина string - це об'єкт. Він містить послідовність символів, які складають рядок. У ньому є також функції. Наприклад, для обчислення довжини рядка існує функція length. Рядок можна створити за допомогою наступної команди:

String str;

str = "Seize the day!";

Ще приклад:

System.out.print ("The number of characters in"); System.out.println ("the string V'Hello WorldV is"); System.out.println ("Hello World". Length ());

Об'єкт string містить безліч функцій. Ось деякі з них:

– si.equals (s2)- повертає логічне значення, true - якщо рядок si в точності така ж, як рядок s2.

– si.equalslgnoreCase (s2)- те ж саме, що і sl.equalsO, але великі і малі літери вважаються однаковими.

– si.charAt(N) - значення типу char, символ, розташований на позиції з номером N в рядку, починаючи з нульової позиції.

– si. length () - ціле значення, рівне кількості символів в рядку.

– si.substring(n,m) - тип string, рядок з символами в позиціях N.

– N + 1, ..., м-1.

– si.compareTo(s2) - ціле, якщо рядки рівні, то 0.

– si.indexof(s2) - ціле, якщо s2 є фрагментом si, то повертається номер позиції, з якого починається s2 в si.

– si.touppercase() - рядок, записана великими літерами.

– si.trim - рядок з віддаленими недрукованих символів, такими як прогалини, табуляції і т. п.

Рядки можна складати:

– System.out.println("Hello," + name + ".Pleased to meet you!");

– Примітивні типи можна додавати до рядку, тоді вони приводяться до рядковому типу:

– System.out.print("After"); System.out.print (years);

– System.out.print("years, thevalueis"); System.out.print (principal);

Це можна записати у вигляді:

– System.out.print ("After" + years + "years, the value is" + principal);

Вирази:

- Функції, літерали, змінні - це прості вирази. З простих виразів та операторів складаються більш складні вирази, наприклад A + B * C, B * C.
- Арифметичні оператори:

Арифметичні оператори позначаються знаками +, -, *, /.

- Оператори зменшення та збільшення на одиницю:

Такі оператори вимагають тільки одного операнда, позначаються + + і - .

counter = counter + 1; goalsScored = goalsScored + 1;

Вирази можна записати за допомогою операторів збільшення та зменшення:

counter + +; goa1s S зі red + +;

Приклад:

y = x + +; y = + + x;

Text10.putIn (-x); z = (+ + x) * (y-);

Якщо ми візьмемо x, що дорівнює 8, то після виконання функції y = x + + y буде дорівнює 8, а після виконання інструкції y = + + x y буде дорівнює 9.

- Оператори порівняння:

Оператори порівняння використовуються при роботі з логічними величинами, значення їх булеве, тобто true або false:

- a! = в - A "не дорівнює" B;

- a == в - A "дорівнює" B;

- a> в - A "більше ніж" B;

- a <в - A "менше ніж" B;

- a> = в - A "більше або дорівнює" B;

- a <= в - A "менше або дорівнює" B.

Ми можемо їх використовувати в якості операндів при роботі з числовими значеннями.

Приклад:

boolean sameSign;

sameSign = ((x> 0) == (y> 0));

- Логічні оператори:

З логічними значеннями працюють логічні оператори. Це оператори or (АБО) (i), and (I) (&&),not (НЕ) (!). Наприклад:

(X! = 0) && (y / x> 1) test =! test;

- Умовні оператори

Загальний вигляд простого умовного оператора:

boolean-expression? expression-1: expression-2

Коли булевий вираз має значення true, то виконується expression-i, інакше виконується expression-2. Наприклад:

next = (N ce 2 == 0)? (N / 2): (3 * N +1);

- Оператори приведення та присвоювання типу

Наприклад:

int A; double X; short B; A = 17;

X = A; // A приводиться до типу double

B = A; // не можна, тип не наводиться автоматично

- Корекція:

int A;

short B;

A = 17;

B = (short) A; / / тип наводиться в явному вигляді

Тип short утворюється шляхом взяття 4 бітів від вихідного значення, частина інформації губиться. При приведенні таких типів можуть відбутися смислові зміни значень.

Приклад використання операторів приведення типу: (Int)(6*Math.Random o) - ціле випадкове число з набору 0, 1,2, 3, 4, 5.

Можна наводити типи з цілих до char, при цьому береться значення символу відповідно до кодуванням Unicode. Наприклад, (char) 97 - це 'a', а (int) '+' дорівнює 43.

Приклади:

x - = y; / / x = x - y;

x * = y; / / x = x * y;

x / = y; / / x = x / y;

x ce = y; / / x = x% y; (для цілих x і y)

q && = p; / / q = q && p; (для логічних booleans q і p)

- Комбіновані оператори присвоювання можна використовувати і з рядками:

str + = X str = str + X

- Ієрархія операторів:

Відомо, що між операторами існує ієрархія. У послідовності операторів вони виконуються відповідно до правил ієрархії операторів, а не один за одним. Приведемо приклад списку операторів за їх розташуванням в самій ієрархії. Якщо оператор знаходиться високо у списку, то це означає, що він володіє високим пріоритетом. З 2 операторів, перший виконується той, який знаходиться в списку вище за пріоритетом в зрівнянні з іншим.

Якщо для виконання оператора необхідна одна операція, то такий оператор називається унарним. В бінарних операторах повинно бути хоча б 2 операнда, в тернарних – мінімум 3.

Список операторів відповідно до ієрархії:

1. Унарні оператори: -,!, + +, Унарні + і -;

2. Ділення та множення: /,*,%;

3. Віднімання і додавання: -, +;

4. Оператори відносин: <,>, <=,> =;

5. Оператори рівності та нерівності: ==,! =;

6. Логічне додавання "І": &&;

7. Логічне "АБО": | |;

8. Умовний оператор:?;

9. Оператори присвоювання: =, + =, - =, * =, / =,% =.

Ті оператори, що знаходяться в одному рядку в списку, мають однаковий пріоритет. З яким порядком оператори виконуються в тексті програми, за таким же пріоритетом вони і розташовуються. Завжди потрібно пам’ятати, що для операторів присвоювання та унарних операторів цей порядок такий, що оператори виконуються справа наліво. Інші - зліва направо. Їх порядок необхідно змінювати. Саме з цією метою використовуються круглі дужки. Спочатку виконуються оператори в дужках, після чого уже всі інші.

Перед інтегруванням додатків, нам потрібно їх розробити, для цього ми використовуємо Eclipse IDE (див. рис. 3.1), для написання програмного коду на Java.

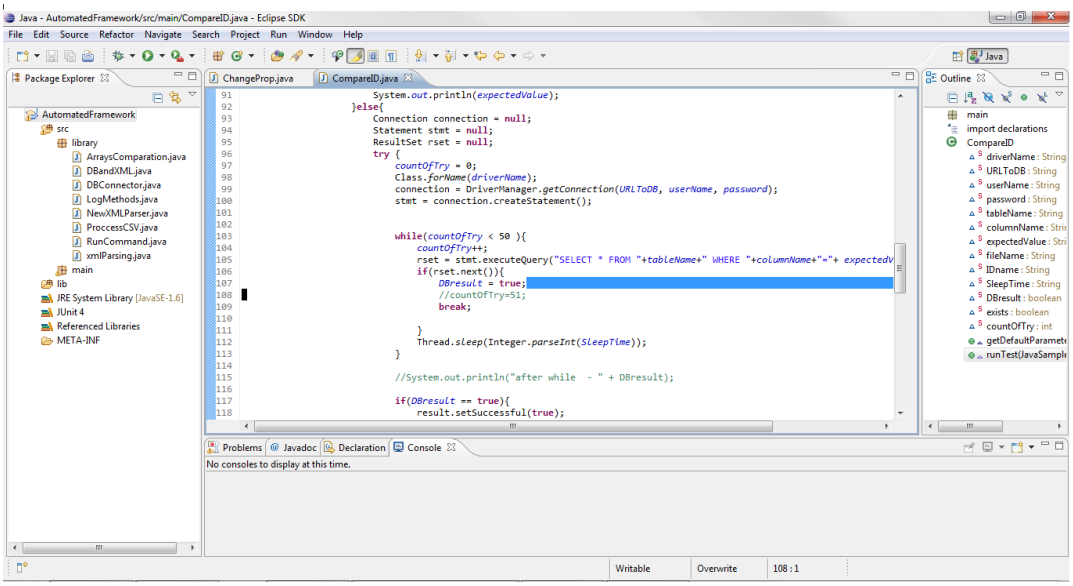

Рис. 3.1. Середовище розробки – Eclipse

Переобумовлюємо два методи:

1. getDefaultParameters();

Цей метод використовується для відображення вхідних параметрів. Наприклад:

```
@Override
    public Arguments getDefaultParameters() {
        Arguments args = new Arguments();
        args.addArgument("fileName", "");
        args.addArgument("parameter", "");
        args.addArgument("value", "");
        return args;
        }
```

Як бачимо на рисунку 3.2 відображаються три вхідні параметри, відповідно до написаного кода.

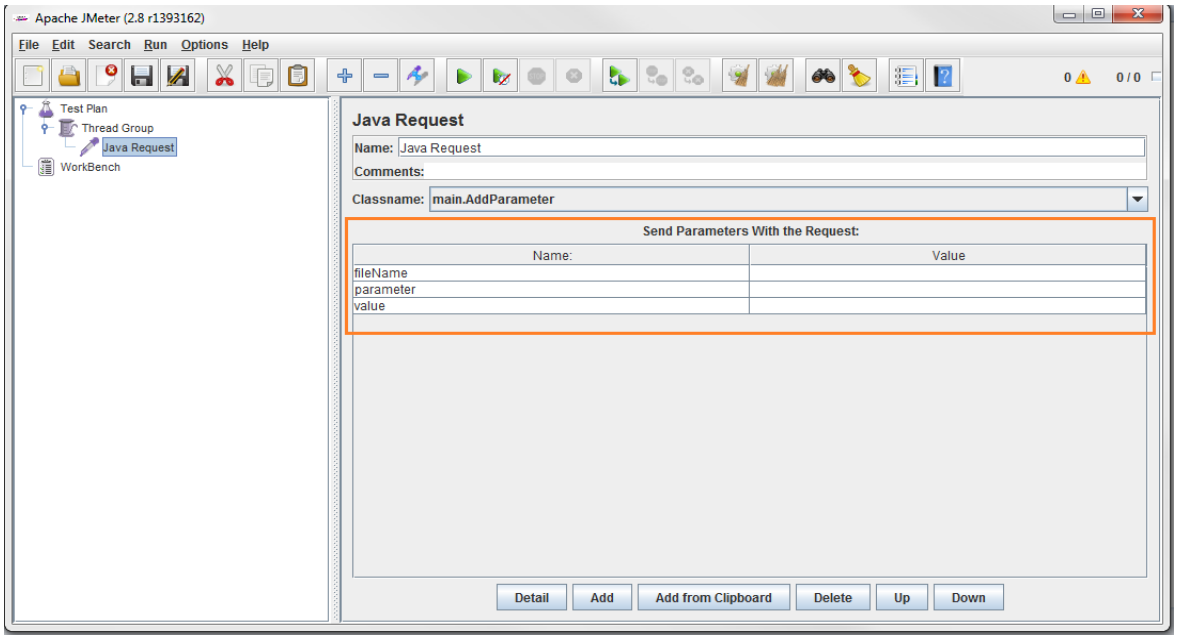

Рис. 3.2. Вхідні параметри Java запиту

2. Метод - runTest();

В даному методі описується сама структура потрібного класу або методу.

```
@Override
    public SampleResult runTest(JavaSamplerContext contex) {

        SampleResult result = new SampleResult();
        result.setSampleLabel("Start of test");
        result.setDataType(SampleResult.TEXT);
        result.sampleStart();

//Some code . . .;
```

```
result.sampleEnd();
        return result;
    }
```

2. Після написання коду створюємо .jar файл за допомогою Eclipse.

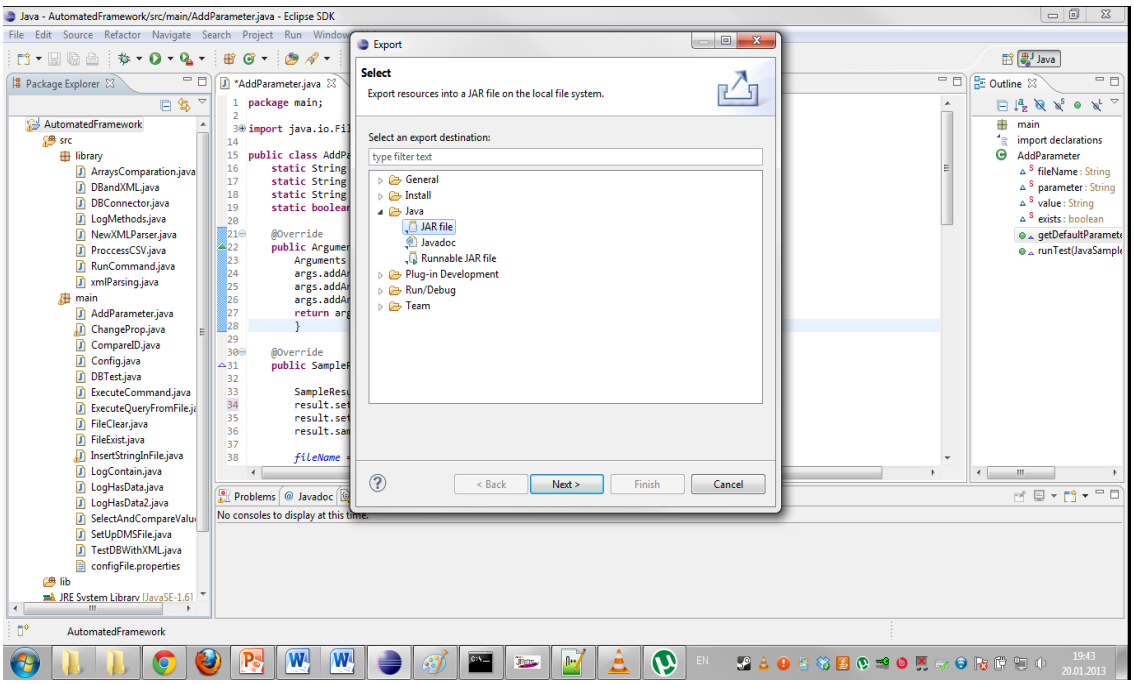

Рис. 3.3. Експорт .jar файлу

Всі класи які були імпортовані в Jmeter можна переглянути під час додавання Java request-у (див. рис. 3.4).

Під час експорту .jar файлу, потрібно вказати директорію – lib/ext (домашня папка Jmeter).

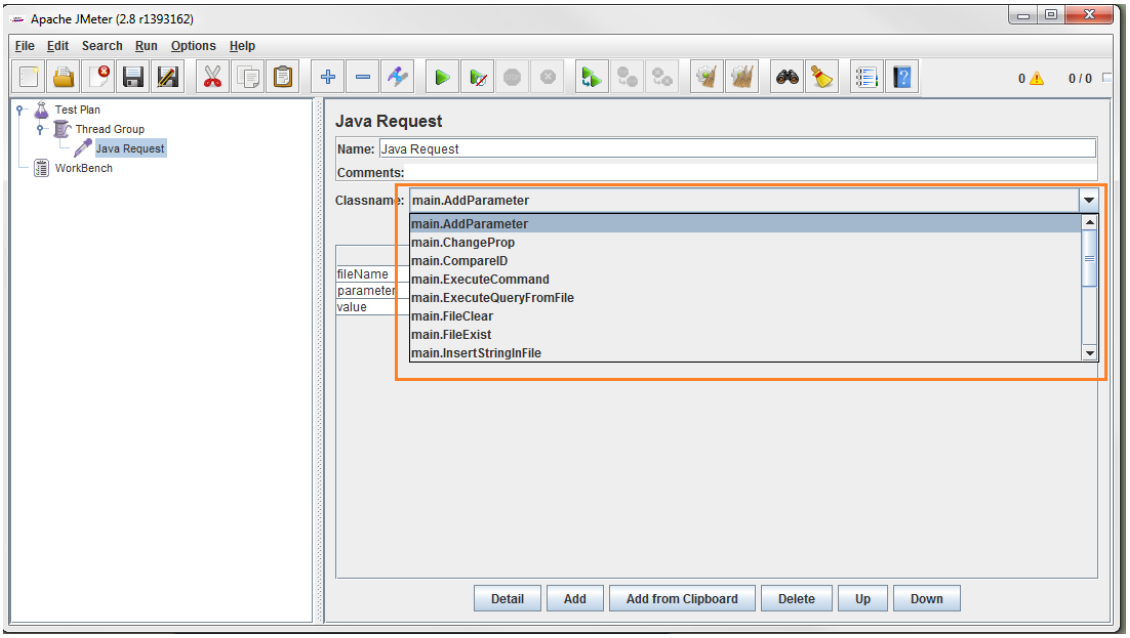

Рис. 3.4. Список імпортованих класів в Jmeter

3.4. Реалізація сценаріїв тестування в коді

Для реалізації усіх сценарії нам потрібні були наступні класи та бібліотеки (див. рис. 3.5):

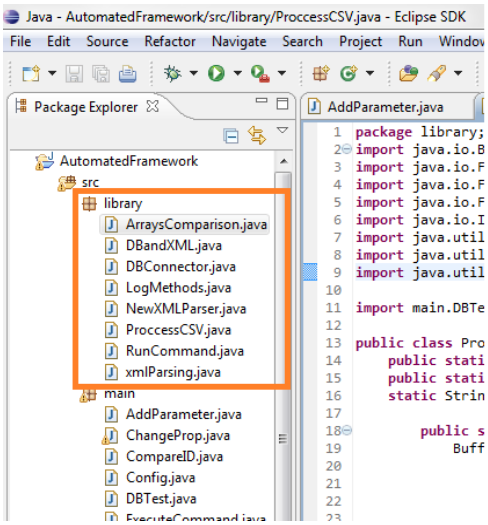

Рис. 3.5. Список розроблених бібліотек

1. Клас – ArraysComparison.java:

Даний клас використовується для порівняння двох масивів з типом String.

Вигляд початку метода:

public String compareArrays(String XMLarray, String DB){//…

Метод приймає два об’єкти типу String. І за допомогою циклу foreach ці два об’єкти перебираються по одному значені і порівнюються:

for (String item : arrXMLResult) {//…

2. DBandXML.java.

Оскільки тут використовується великий обсяг функціоналу та найбільша кількість бібліотек для виконання коду, то це один з основних класів який використовує розроблений додаток.

Цей клас має два методи:

– compareXMLandDB - метод порівнює значення з бази даних та значення в XML файлі, може приймати для вхідних параметрів:

- String driverName – назва драйверу який буде використовуватись (в залежності від типу БД він змінюється, наприклад для IBM DB використовується db2jcc.jar);
- String fileName – назва XML файлу;
- String userName – ім’я користувача;
- String password – пароль;
- String URLToDB – шлях до бази даних.

Після завершення програми метод повертає тип Boolean. True – значення рівні, False – значення не рівні.

– executeQueryFromFile – виконання SQL команди з файлу, може приймати для вхідних параметрів:

  - String fileName – шлях до файлу з якого буде виконуватись SQL команда.

Наступні параметри аналогічні з методом, compareXMLandDB. Якщо виникає незвичайна ситуація та програма видає помилку, то ми можемо обробити її за допомогою коду:

```
}catch(Exception e){
    StringWriter sw = new StringWriter();
    PrintWriter pw = new PrintWriter(sw);
    e.printStackTrace(pw);
    result = result + sw.toString();
//...
```

3. DBConnector.java – виконання SQL команди з консолі Jmeter.

Головним параметром для початки роботи являється запит на виконання. Якщо цей запит буде виконано успішно, то метод повертає набір назад з бази даних. Обробку дефектів бере на себе java.lang.Throwable, що виглядає наступним чином:

```
catch (ClassNotFoundException e) {
    // Could not find the database driver
    System.err.println ("Unable to load database driver");
    DBTest.errorResult =  DBTest.errorResult + ". error: Unable to load database driver: "  + e.getMessage();
```

```
            System.err.println ("Details : " + e);
            System.exit(0);
        } catch (SQLException e) {
            // Could not connect to the database
            System.out.println("Could not connect to the database");
            System.err.println ("Details : " + e);
            DBTest.errorResult = DBTest.errorResult + ". SQL error: " +
e.getMessage();
        }finally{
            if(rset!=null){
            rset.close();
        }
            if(stmt!=null){
            stmt.close();
            }
            if(connection!=null){
            connection.close();
            }
        }
```

Можемо побачити, що оброблюються дві не стандартні ситуації, якщо виникає помилка в запиті або драйвер до бази даних не знайдений. Після завершення роботи, метод виконує структуру – finally. В ній відбувається закривання зв’язку між базою та нашим методом, а також знищується пам'ять яка була виділена на інші об’єкти.

4. LogMethods.java – один із найбільш об’ємних класів. Містить в собі наступні методи:

– public boolean isEmpty(String fileName) – перевірка на вміст даних в файлі, true – файл пустий, false – файл містить дані.

```
fileInputStream = new FileInputStream(new File(fileName));
        int empty = fileInputStream.read();
        if (empty == -1)
        {
        //System.out.println("!File " + fileName + " empty !");
        result = true;
        }
```

Якщо значення empty == -1, то це свідчить про те, що файл пустий і значенню result присвоюється true.

– public boolean contains(String fileName, String value) – цей метод дозволяє перевірити вказане значення в файлі, true – файл містить вказане значення, false – файл не містить потрібних даних.

```
bufferedReader = new BufferedReader(new FileReader(fileName));
                String line = null;
                while ((line = bufferedReader.readLine()) != null) {
                    //Process the data, here we just print it out
                    //System.out.println(line);
                    if(line.contains(value)){
                     result = true;
                    }
                 }
```

За допомогою класу - BufferedReader весь набір даних в файлі розбивається на рядки. Після цього ми проходимось по кожному за допомогою циклу while ((line = bufferedReader.readLine()) != null). Перевірка на вміст даних в рядку виконується методом contains(value).

public boolean cleanAllLogsFiles(String folderName)- видалення всіх файлів в заданій теці. В якості вхідного параметра приймає назву теки. Якщо видалення відбулось успішно – true, інакше – false.

```
File[] files = directory.listFiles();
        for (File file : files)
        {
                // Delete each file
                if (!file.delete())
                {
                        // Failed to delete file
                        System.out.println("Failed to delete "+file);
                }
        }
```

– static public String convertToDate(String time)- змінює дату в формат "yyyy-MM-dd HH:mm:ss".

```
try {
        DateFormat df = new SimpleDateFormat("yyyy-MM-dd HH:mm:ss");
        Date date = df.parse(time);
        result = date.toString();
        System.out.println(result + " - result");
        } catch (ParseException e) {
```

```
            e.printStackTrace();
            result = e.getMessage();
        }
        return result;
```

Формат дати можна задавати за допомогою класу SimpleDateFormat();

5. NewXMLParser.java – включає в себе наступні методи:

– public static boolean ReplaceValue(String xpathvalue, String fileName, String newValue)- змінює значення тегу в XML файлі, знаходить тег за допомогою Xpath. Для прикладу розглянемо наступний XML файл:

```
<?xml version="1.0"?>
<note>
   <to>Tove</to>
   <from>Jani</from>
   <heading>Reminder</heading>
   <body>Don't forget me this weekend!</body>
</note>
```

Для того, що б змінити значення тегу <to>Tove</to>, потрібно визвати метод з наступними параметрами - ReplaceValue(“/note/to”, “fileLocation”,”Mandziuk”);

Після виконання даного методу XML набуде наступного вигляду:

```
<?xml version="1.0"?>
<note>
   <to>Mandziuk</to>
   <from>Jani</from>
```

```
    <heading>Reminder</heading>
    <body>Don't forget me this weekend!</body>
</note>
```

– public static boolean AddString (String inFile, String linenom, String lineToBeInserted) – дозволяє вставити рядок в файл, номер рядку також вказується у вхідних параметрах.

```
FileOutputStream fos = new FileOutputStream(outFile);
        PrintWriter out = new PrintWriter(fos);
  String thisLine = "";
        int i =1;
        while ((thisLine = in.readLine()) != null) {
          if(thisLine.equals(lineToBeInserted)){
          out.println(thisLine);
          i++;
        }else{ if (i == lineno) out.println(lineToBeInserted);
     out.println(thisLine);
           i++;
           //return true;
                }
    }
```

Файл перебирається по лінії за допомогою циклу while, якщо лінія вже містить вказаний рядок if(thisLine.equals(lineToBeInserted)),

цикл проходить дальше, інакше вставляється потрібний рядок операції –

else{ if (i == lineno) out.println(lineToBeInserted).

6.ProccessCSV.java – клас для роботи з CSV(comma separated values) файлом.

– public static Map<Integer,String>

– writeDataToMap(String fileName) – в якості вхідного параметру приймає повну назву CSV файлу і записує його в колекцію Map<Integer,String>. В тип Integer попадає номер лінії, а в тип String – значення рядку.

```
 br= new BufferedReader( new FileReader(fileName));
int lineNumber = 0;
String strLine = "";
//read comma separated file line by line
while( (strLine = br.readLine()) != null)
{
   lineNumber++;
 strLine = strLine.replaceAll("\"", ""); // delete quotes in string
 //System.out.println("Value of string - " + strLine);
    allFile.put(lineNumber, strLine);
 }
```

Як ми бачимо, цикл while проходиться по файлу і дістає значення рядку та його порядковий номер, після цього записує ці дані в колекцію allFile.put(lineNumber, strLine);

– public static void writeResulToCSV(String

– rowWithData, String resultOfTest,String fileName) – метод записує результат перевірки в CSV файл.

7. RunCommand.java – запуск консольної команди за допомогою java.

Включає тільки один метод:

– public static String execute(String command) – в параметр передається команда до виконання.

```
Process p1 = null;
        String result = "";
        try {
            p1 = Runtime.getRuntime().exec(command);
            BufferedReader reader = new BufferedReader(new InputStreamReader(p1.getInputStream()));
            String line = null;

            while((line = reader.readLine()) != null)
            {
                result += line + "\n";
            }
        } catch (IOException e) {
            result += e.getMessage();
        }
        return result;
```

Головний метод - Runtime.getRuntime().exec(command). Цей метод необхідний для виконання команди, яка вказується з допомогою циклу і виводиться в форматі параметру String.

8. xmlParsing.java – обробка XML файлу.

ВИСНОВКИ

Ми дослідили варіанти автоматизованого тестування та дізналися, що вони нам допомагають вирішувати одразу декілька проблем:

1. Значна економія часу та людських ресурсів. Це повністю автоматизована система, втручання в яку необхідно лише для поповнення бібліотеки сценаріїв, підтримки деяких тестів в актуальному стані та вивчення звітів роботи.

2. Надійність. Як правило система завжди оновлюється, саме тоді і виникають помилки, які важко вловити навіть досвідченому спеціалісту. Таким чином система знаходиться під постійним контролем скрипту і не може порушуватися при зміні компонентів, що являється стабілізацією її надійності.

Звернувши увагу на матеріал зверху, можемо підвести список усіх рекомендацій для автоматизації тестування:

– Необхідно використовувати цикл розробки самої системи паралельно з методами життєвого циклу автоматизованого тестування.

– Необхідно суворе контролювання процесу безпосередньо програмного забезпечення. Автоматизоване тестування потребує чітко визначеного процесу для забезпечення ефективності та виконання роботи в установлені інтервали часу.

– Не завищувати очікування від автоматизації тестування.

– Правильно прорахувати кількість тестів, які потребують автоматизації. Деякі тести дуже складно автоматизувати або ж вони взагалі не піддаються їй, можете затратити чимало зусиль на неї.

– Прорахувати потрібність впровадження автоматизації в тестування.

– Можемо використовувати спеціальні типи тестування, для того, щоб скоротити їх терміни, використовується, якщо нам недоцільно проводити повністю весь цикл тестування.

– Виконувати автоматизацію тільки в неробочий час генерації звітів.

Необхідно також відповідально підійти до вибору засобів для автоматизації тестування.

В деяких компаніях, вартість засобів для рішень автоматизації дуже висока, тому їх використання в маленьких роботах стає недоцільною. Проте, якщо компанія планує довгострокову роботу з програмним забезпеченням, то такий варіант вкладу буде повністю виправданим. Та потрібно завжди пам’ятати, що ці продукти використовують виробники з чудовою репутацією, що не може не гарантувати нам якість її роботи. До того ж вони мають централізовану підтримку та бездоганно взаємодіють один з одним.

При теперішніх тенденціях розвитку програмного забезпечення та переходу на вільне програмування, ми можемо розглядати варіант переходу на вільне автоматизоване тестування. Відразу потрібно підмітити відсутність централізованої підтримки такого тестування, відсутність гарантій та різнорідність програмних інструментів. З іншої сторони не має потреби робити покупки коштовних ліцензій та доступність до великої кількості уже чинних засобів, можливість налаштувань у додатках з закритим кодом.

У даному проекті були представлені усі основи якості автоматизованого тестування програмного забезпечення, було проведено робота з уже діючими засобами автоматизації. Реалізовані нові бібліотеки Jmeter, які допомогли розширити його можливості в тестуванні та покращили його функціонал. Поставлені задачі – виконані.

СПИСОК БІБЛІОГРАФІЧНИХ ПОСИЛАНЬ ВИКОРИСТАНИХ ДЖЕРЕЛ

1. SWEBOK - 2004. IEEE Guide to Software Engineering Body of Knowledge – Введ. 2015: Видав. стандартів – 2004.
2. IEEE 610.12 - 1990. Standard Glossary of Software Engineering Terminology - Введ. 2012.10: Видав. стандартів – 1990.02
3. Блэк Р. Ключевые процессы тестирования. Планирование, подготовка, проведение, совершенствование/ Р. Блэк; Москва: Лори, 2006 - 537 с.
4. Котляров В. Основы тестирования программного обеспечения/ В. Котляров, Т. Коликова; Москва: Бином. Лаборатория знаний, 2006 – 248 с.
5. Винниченко І. Автоматизация процессов тестирования/ І. Винниченко; Санкт Петербург: видавницький дом “Питер”, 2005 -203 с.
6. Бейзер Б. Тестирование черного ящика. Технологии функционального тестирования программного обеспечения и систем/ Б. Бейзер; Санкт Петербург: видавницький дом “Питер”, 2004 – 320 с.
7. Полевой В. Как автоматизировать тестирование ПО? / В. Полевой; Москва: Cnews, 2007 -219 с.
8. Тестирование и качество ПО. [Електронний ресурс] - Режим доступу: http://software-testing.ru/ (дата звернення 23.05.21р) – Назва з екрана.

ДОДАТКИ

Додаток А

Створення аплетів:

public class InterestIConsole extends ConsoleApplet {

protected String getTitleO {

return "Sample program V'lnterestlV";

}

protected void program () {

double principal; / / вкладена сума

double rate; / / річний відсоток

double interest; / / відсоток за рік

*/ * Обчислення * / principal = 17000; rate = 0.07;*

*interest = principal * rate; / / відсоток principal = principal + interest; / * виведення результату * / console.put ("The interest earned is $"); console.putln (interest);*

console.put ("The value of the investment after one year is $"); console.putln (principal);

}

Також можему використовувати наступну програму, що міститься у файлі

ConsoleApplet.java:

*impo гt j ava.awt. *;*

*impo rt java.awt.event. *;*

public class ConsoleApplet extends java.applet.Applet

implements Runnable, ActionListener {protected String title = "Java Console I / O"; protected String getTitleO {return title;

}

protected ConsolePanel consoles-protected void program () {

console.putln ("Hello, World!");

}

private Button runButton;

private Thread programThread = null;

/ / Потік run () private boolean programRunning = falser-private boolean firstTime = true; / / false - якщо програма вважає перший раз

public void run () {/ / запуск programRunning = true;

program ();

programRunning = false; stopProgram ();

}

synchronized private void startProgram () {runButton.setLabel ("Abort Program"); if (! firstTime) {console.clear (); try {Thread.sleep (300);} / / затримка перед перезапуском програми

catch (InterruptedException e) {}

}

firstTime = false; programThread = new Thread (this); programThread.start ();

}

synchronized private void stopProgram () {if (programRunning) {programThread.stop (); try {programThread.join (1000);} catch (InterruptedException e) {}

}

```
console.clearBuffers (); programThread = null; programRunning = false; runButton.setLabel ("Run Again"); runButton.requestFocus ();
}
public void init () {
setBackground (Color.black);
setLayout (new BorderLayout (2,2)); console = new ConsolePanel (); add ("Center", console);
Panel temp = new Panel ();
temp.setBackground (Color.white); Label lab = new Label (getTitle ()); temp.add (lab);
lab.setForeground (new Color (180,0,0)); add ("North", temp);
runButton = new Button ("Run the Program");
temp = new Panel ();
temp.setBackground (Color.white);
temp.add (runButton);
runButton.addActionListener (this);
add ("South", temp);}
public Insets getlnsets () {return new Insets (2,2,2,2);
}
public void stop () {
if (programRunning) {stopProgram (); console.putln ();
console.putln ("*** PROGRAM HALTED");
}
}
synchronized public void actionPerformed (ActionEvent evt) {if (programThread! = null) {stopProgram (); console.putln ();
console.putln ("*** PROGRAM ABORTED BY USER");
}
```

else

startProgram ();

}

}

Створення змінних та операторів присвоння:

public class Interest {

*public static void main (String [] args) {/ * оголошення змінних * / double principal; / / вкладена сума*

double rate; / / річний відсоток

double interest; / / нарахування за рік

*/ * Обчислення * / principal = 17000; rate = 0.07;*

*interest = principal * rate; principal = principal + interest;*

*/ * Виведення результату * / System.out.print ("The interest earned is $"); System.out.println (interest);*

System.out.print ("The value of the investment after one year is $");

System.out.println (principal);}

}

Математичні функції:

public class TimedComputation {

public static void main (String [] args) {

long startTime; // стартовий час в мілісекундах

long endTime; // час при завершенні обчислень, в мілісекундах double time; // Різниця часів, в секундах. startTime = System.currentTimeMillis ();

```
double width, height, hypotenuse; // сторони трикутника width = 42.0; height = 17.0;
hypotenuse = Math.sqrt (width * width + height * height);
System.out.print ("A triangle with sides 42 and 17 has hypotenuse");
System.out.printIn (hypotenuse);
System.out.printIn ("\ nMathematically, sin (x) * sin (x) +" + "cos (x) * cos (x) - 1 should be 0."); System.out.println ("Let's check this for x = 1: ");
System.out.print ("sin (l) * sin (l) + cos (1) * cos (1) - 1 is"); System.out.printIn (Math.s in (1) * Math.s in (1)
+ Math.cos (1) * Math.cos (1) - 1); System.out.println ("(There can be round-off errors when" + "computing with real numbers!)");
System.out.print ("\ nHere is a random number:");
System.out.printIn (Math.random ());
endTime = System.currentTimeMillis ();
time = (endTime - startTime) / 1000.0;
System.out.print ("\ nRun time in seconds was:");
System.out.printIn (time);
}
}
```

Додаток В

Виконання SQL команди в Jmeter:

```
catch (ClassNotFoundException e) {
    // Could not find the database driver
    System.err.println ("Unable to load database driver");
    DBTest.errorResult =  DBTest.errorResult + ". error: Unable to load database driver: "  + e.getMessage();
    System.err.println ("Details : " + e);
    System.exit(0);
} catch (SQLException e) {
    // Could not connect to the database
    System.out.println("Could not connect to the database");
    System.err.println ("Details : " + e);
    DBTest.errorResult =  DBTest.errorResult + ". SQL error: "  + e.getMessage();
}finally{
    if(rset!=null){
    rset.close();
}
    if(stmt!=null){
    stmt.close();
    }
    if(connection!=null){
    connection.close();
    }
}
```

Метод static public String convertToDate:

```
try {
        DateFormat df = new SimpleDateFormat("yyyy-MM-dd HH:mm:ss");
        Date date = df.parse(time);
        result = date.toString();
        System.out.println(result + " - result");
        } catch (ParseException e) {
                e.printStackTrace();
                result = e.getMessage();
        }
        return result;
```

Метод public static boolean AddString:

```
FileOutputStream fos = new FileOutputStream(outFile);
        PrintWriter out = new PrintWriter(fos);
  String thisLine = "";
        int i =1;
        while ((thisLine = in.readLine()) != null) {
          if(thisLine.equals(lineToBeInserted)){
          out.println(thisLine);
          i++;
        }else{ if (i == lineno) out.println(lineToBeInserted);
     out.println(thisLine);
           i++;
           //return true;
```

```
                }
            }
```

Метод DBConnector.java:

```
        catch (ClassNotFoundException e) {
            // Could not find the database driver
            System.err.println ("Unable to load database driver");
            DBTest.errorResult =  DBTest.errorResult + ". error: Unable to load database driver: "  + e.getMessage();
            System.err.println ("Details : " + e);
            System.exit(0);
        } catch (SQLException e) {
            // Could not connect to the database
            System.out.println("Could not connect to the database");
            System.err.println ("Details : " + e);
            DBTest.errorResult =  DBTest.errorResult + ". SQL error: "  + e.getMessage();
        }finally{
            if(rset!=null){
            rset.close();
        }
            if(stmt!=null){
            stmt.close();
            }
            if(connection!=null){
            connection.close();
            }
```

```
}
```

Метод public boolean cleanAllLogsFiles:

```
File[] files = directory.listFiles();
        for (File file : files)
        {
                // Delete each file
                if (!file.delete())
                {
                        // Failed to delete file
                        System.out.println("Failed to delete "+file);
                }
        }
```

Метод static public String convertToDate:

```
try {
        DateFormat df = new SimpleDateFormat("yyyy-MM-dd HH:mm:ss");
        Date date = df.parse(time);
        result = date.toString();
        System.out.println(result + " - result");
        } catch (ParseException e) {
                e.printStackTrace();
                result = e.getMessage();
        }
        return result;
```

Метод public static boolean ReplaceValue:

```
<?xml version="1.0"?>
<note>
   <to>Tove</to>
   <from>Jani</from>
   <heading>Reminder</heading>
   <body>Don't forget me this weekend!</body>
</note>
<?xml version="1.0"?>
<note>
   <to>Mandziuk</to>
   <from>Jani</from>
   <heading>Reminder</heading>
   <body>Don't forget me this weekend!</body>
</note>
```

Метод public static boolean AddString:

```
FileOutputStream fos = new FileOutputStream(outFile);
        PrintWriter out = new PrintWriter(fos);
  String thisLine = "";
        int i =1;
        while ((thisLine = in.readLine()) != null) {
          if(thisLine.equals(lineToBeInserted)){
          out.println(thisLine);
          i++;
```

```
    }else{ if (i == lineno) out.println(lineToBeInserted);
  out.println(thisLine);
        i++;
        //return true;
            }
}
```

Метод writeDataToMap:

```
     br= new BufferedReader( new FileReader(fileName));
    int lineNumber = 0;
    String strLine = "";
    //read comma separated file line by line
    while( (strLine = br.readLine()) != null)
    {
       lineNumber++;
     strLine = strLine.replaceAll("\"", ""); // delete quotes in string
     //System.out.println("Value of string - " + strLine);
       allFile.put(lineNumber, strLine);
     }
```

Метод public static String execute

```
Process p1 = null;
            String result = "";
            try {
                  p1 = Runtime.getRuntime().exec(command);
```

```
                    BufferedReader reader = new BufferedReader(new
InputStreamReader(p1.getInputStream()));
                    String line = null;

                    while((line = reader.readLine()) != null)
                    {
                        result += line + "\n";
                    }
                } catch (IOException e) {
                    result += e.getMessage();
                }
                return result;
```