

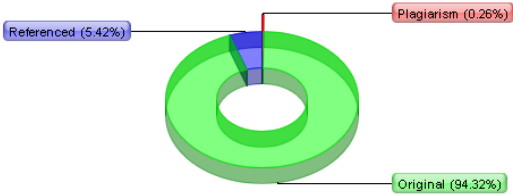
Детектор Плагиата v. 2215 - Отчёт оригинальности: 16.06.2024 18:41:45

Проанализированный документ: Нєлєпа_ПЗ.docx Лицензия: ВОЛОДИМИР МАТІЄВСЬКИЙ_License2

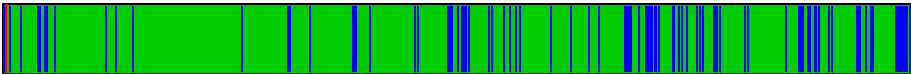
- Тип поиска: Поиск переписанного
- Язык: Uk
- Тип проверки: Интернет
- ТЕЕ и кодировка: DocX n/a

Детальный анализ тела документа:

Диаграмма соотношения частей:



Граф распределения зон:



Источники плагиата: 9

| | | |
|------|----|--|
| 0.5% | 28 | 1. https://it.nmu.org.ua/ua/scientific_method_materials/lecture_notes/Конспект_лекцій_БД_частина1_2021.pdf |
| 0.3% | 10 | 2. https://uk.wikipedia.org/wiki/Інтерфейс_користувача |
| 0.3% | 8 | 3. https://git-scm.com/book/uk/v2/Додаток-A:-Git-в-інших-середовищах-Графічні-інтерфейси |

Детали обработанных ресурсов: 175 - ОК / 8 - Ошибок

Важные замечания:

| | | | |
|-----------------|-----------------|------------------------|----------------------|
| Википедия: | Google Книги: | Сервисы платных работ: | Античит: |
| | | | |
| [не обнаружено] | [не обнаружено] | [не обнаружено] | Обнаружено сокрытие! |

Античит-отчет UACE:

| |
|--|
| 1. Статус: Анализатор Включен Нормализатор Включен сходство символов установлено на 100% |
| 2. Обнаруженный процент загрязнения UniCode: 27% с лимитом: 4% |
| 3. Процент нераспознанных символов после нормализации: 14,9% |
| 4. Все подозрительные символы будут отмечены фиолетовым цветом: Abcd... |
| 5. Найдены невидимые символы: 0 |
| Рекомендации по оценке: Особое внимание следует уделить анализу этого отчета! Предполагается, что этот документ содержит значительное количество символов, чуждых языку документа. Это прямое указание на то, что автор документа использовал специальное программное обеспечение\онлайн-веб-сервис, чтобы эффективно скрыть текст в попытке избежать обнаружения потенциального плагиата. Настоятельно рекомендуется передать это дело на более высокий уровень! В случае сомнений обращайтесь: в службу поддержки Детектора плагиата! |
| Алфавитная статистика и анализ символов: |

Активные ссылки (URL-адреса, извлеченные из документа):

URL не найдены

Исключённые ресурсы:

URL не найдены

Включённые ресурсы:

URL не найдены

Детальний аналіз документа:

| | |
|--|-------|
| Міністерство освіти і науки України Державний заклад | |
| Цитування: 0,08% | id: 1 |
| «Луганський НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ІМЕНІ Тараса Шевченка» | |
| Обнаружен Плагиат: 0,19% https://it.nmu.org.ua/ua/scientific_me... | id: 2 |
| Факультет (інститут) Навчально-науковий інститут математики та інформаційних технологій (повна назва) Кафедра інформаційних технологій та | |
| систем (повна назва) Пояснювальна записка до кваліфікаційної роботи за першим (бакалаврським) рівнем освіти Розробка клієнт-серверного java -додатку для автоматизації спілкування із клієнтами Виконав: студент 4 курсу напряму підготовки (спеціальності) 121 | |
| Цитування: 0,04% | id: 3 |
| «Інженерія програмного забезпечення» | |
| (шифр і назва напряму підготовки, спеціальності) Нелепа Р.О. (прізвище та ініціали) Керівник __Переяславська С.О. (прізвище та ініціали) Рецензент __Козуб Ю.Г. (прізвище та ініціали) Полтава – 2024 року ЗМІСТ ВСТУП4 Розділ 1. Теоретичні основи РОзробки клієнт-серверних додатків6 1.1.Клієнт-серверна архітектура6 1.2.Моделі клієнт-серверних систем7 1.3.Протоколи взаємодії клієнта і сервера11 1.4.Особливості розробки клієнт-серверних додатків і їх реалізація на мові програмування java 13 Висновки до розділу 116 РОЗДІЛ 2. РОЗРОБКА ПРОГРАМНОГО ДОДАТКА | |
| Цитування: 0,03% | id: 4 |
| “КОРПОРАТИВНИЙ ЧАТ” | |
| НА ПЛАТФОРМИ JAVA 17 2.1. Короткий огляд структури додатку17 2.2. Розробка серверного додатку18 2.3. Розробка клієнтського додатку22 2.4. Розробка бота24 2.5. Приклад роботи додатка27 Висновки до розділу 230 ВИСНОВКИ31 СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ32 Додаток А34 ДОДАТОК В44 ПЕРЕЛІК УМОВНИХ СКОРОЧЕНЬ OOP - Object-Oriented Programming MVC - Model View Controller TCP - Transmission Control Protocol IP - Internet Protocol JVM - Java Virtual Machine JRE - Java Runtime Environment JDK - Java Development Kit GUI - Graphical User Interface ВСТУП Актуальність роботи. Кожен день люди користуються сучасними інформаційними технологіями не замислюючись про те як вони влаштовані і як працюють. На даний момент більшість користувачів не уявляють свого існування без доступу до всесвітньої мережі і при цьому навіть не цікавляться тим на якому принципі і за допомогою яких інструментів здійснюється обмін інформацією. Основним бажанням більшості користувачів є можливість споживати інформацію, а також легко отримувати прості відповіді на конкретно поставлені запитання. Як показує практика в плані взаємодії, одним з найефективніших способів є підхід | |
| Цитування: 0,01% | id: 5 |
| "клієнт-сервер" | |
| - це концепція під якою розуміють дві сторони комунікуючих між собою. Так само варто звернути увагу що в основі взаємодії по типу | |
| Цитування: 0,01% | id: 6 |
| "клієнт-сервер" | |
| лежить принцип того, що роботу завжди починає тільки клієнт, а серверна частина тільки відгукується на його запит. Мова java є основою практично для всіляких типів мережевих додатків. У світі налічуються мільйони фахівців, які розробляють додатки на java , яка дозволяє ефективно розробляти та тестувати їх. Метою роботи є розробка клієнт-серверного java -додатка | |
| Цитування: 0,03% | id: 7 |
| "Корпоративний чат". | |
| Об'єкт дослідження: клієнт-серверний додаток | |
| Цитування: 0,03% | id: 8 |
| "Корпоративний чат" | |
| на java з використанням сокетів. Предмет дослідження: технологія створення клієнт-серверного додатка за допомогою об'єктно орієнтованої мови програмування java . Відповідно до предмета дослідження і мети, були виділені основні завдання дослідження: - узагальнити поняття, види і складові при реалізації клієнт-серверного додатка на java . - розглянути підходи і принципи створення клієнт-серверного додатка. - дослідити можливості java для розробки клієнт-серверного додатку, зокрема технологію сокетів; - розробити клієнт-серверний додаток | |
| Цитування: 0,03% | id: 9 |
| "Корпоративний чат" | |
| на мові java і протестувати його. Для вирішення завдань дослідження використано такі методи дослідження: теоретичні: аналіз наукової літератури, узагальнення та систематизація теоретичних положень про створення клієнт-серверних додатків на мові програмування java з використанням сокетів; емпіричні: порівняння моделей клієнт-серверних додатків; експериментальні: тестування розробленої програми. До складу роботи входять два розділи, які висвітлюють питання розробки клієнт-серверного додатку на мові програмування java . Практичне значення розробки – розроблений клієнт-серверний додаток на мові програмування java , який може застосовуватися у навчальних цілях під час підготовки відповідних напрямків спеціалістів та використовуватися у розважальних цілях. Розділ 1. Теоретичні основи РОзробки клієнт-серверних додатків Клієнт-серверна архітектура У світі технологій, інтернет приніс докорінні зміни в процес обміну інформацією, який об'єднав всю земну кулю. Інтернет це мережа, і ця мережа має архітектуру і структуру для комунікації. Модель клієнт-сервер - це розподілена структура додатку, яка розділяє завдання або робочі навантаження між постачальниками ресурсу або служби, званими серверами, і ініціаторами запитів служб, званими клієнтами. Часто клієнти і сервери обмінюються даними по комп'ютерній мережі на окремому обладнанні, і клієнт, і сервер можуть перебувати в одній системі. Хост сервера запускає одну або кілька серверних програм, які діляться своїми ресурсами з клієнтами. Клієнт зазвичай не ділиться | |

своїми ресурсами, але запитує контент або послугу з сервера. Таким чином, клієнти ініціюють сеанси зв'язку з серверами, які очікували вхідних запитів. Прикладами комп'ютерних додатків, що використовують модель клієнт-сервер, є електронна пошта, інтернет. Клієнт-серверні обчислення - це найбільш ефективне джерело інструментів, які наділяють співробітників повноваженнями і відповідальністю. Клієнт-серверні обчислення - це нездоланний рух, який змінює спосіб використання комп'ютерів. Хоча ці обчислення дуже молоді, вони вже використовуються в повну силу і не залишають недоторканими жодну область і куточок комп'ютерної індустрії. Розробка клієнт-серверних додатків вимагає гібридних навичок, які включають обробку транзакцій, проектування бази даних, дизайн графічного призначеного для користувача інтерфейсу і вміння працювати з Інтернетом. Компоненти клієнт - серверної архітектури. Архітектура клієнт-сервер працює, коли клієнтський комп'ютер відправляє запит ресурсу або процесу на сервер з мережевого з'єднання, який потім обробляється і доставляється клієнту. Серверний комп'ютер може керувати декількома клієнтами одночасно, в той час як один клієнт може бути підключений до декількох серверів одночасно, кожен з яких надає свій набір послуг. Дана архітектура складається з трьох компонентів, а саме клієнт, сервер і мережу. Клієнт - це розрахована на одного користувача робоча станція, яка надає послуги презентації і відповідні обчислення, можливості підключення, послуги бази даних і інтерфейси, що відповідають потребам бізнесу. Мережа - з'єднання яке забезпечує обмін інформацією між клієнтом і сервером. Сервер - це один або кілька багатокористувацьких процесорів із загальною пам'яттю, що забезпечують обчислення, зв'язок і служби баз даних, а також інтерфейси, що відповідають потребам бізнесу. Моделі клієнт-серверних систем Розрізняють три рівні архітектур

Цитирования: **0,05%**

id: **10**

"клієнт - сервер":

Архітектура [Peer-to-peer](#) - однорангова, децентралізована або пірінгова мережа. Заснована на рівноправ'ї учасників, тобто в такій мережі відсутні виділені сервери і кожен вузол є клієнтом і одночасно виконує функції сервера. У мережі присутня деяка кількість комп'ютерів, де кожен може зв'язатися з будь-яким іншим. У подібній архітектурі кожен комп'ютер повинен мати здатність обробляти запити від інших комп'ютерів в мережі. Рис.1.1. Однорангова або пірінгова архітектура Дворівнева архітектура - термін

Цитирования: **0,01%**

id: **11**

«дворівневий»

описує спосіб поділу обробки в додатку клієнт / сервер. Дворівневий додаток ідеально забезпечує кілька робочих станцій з єдиним рівнем уявлення, який взаємодіє з рівнем централізованого зберігання даних. Рівень представлення зазвичай є клієнтом, а рівень зберігання даних - сервер. Фактично, більшість архітектур клієнт / сервер є дворівневими. У подібній архітектурі, в якій мережеве навантаження розподілено між сервером і клієнтами - сервер очікує від клієнтських програм запити і надає їм свої ресурси у вигляді даних (наприклад, завантаження файлів або робота з базами даних) або ресурси у вигляді сервісних функцій (наприклад, робота з електронною поштою, спілкування за допомогою миттєвого обміну повідомленнями і даними або перегляд веб сторінок в інтернеті). Ви можете уникнути проблем з дворівневим клієнт-сервером, розширивши два рівня до трьох. Трирівнева архітектура додає до дворівневої моделі новий рівень, який ізолює обробку даних в центральному місці і максимізує повторне використання об'єктів. Рис. 1.2. Дворівнева архітектура Трирівнева або багаторівнева архітектура - дана модель клієнт-серверної системи, містить у своїй архітектурі

Цитирования: **0,03%**

id: **12**

"третього учасника"

- сховище даних. При використанні цього шаблону, три рівня прийнято називати шарами: Клієнтський шар - програмне забезпечення з призначенням для користувача інтерфейсом, який відправляє запит на сервер і отримує відповідь. Шар так званої логіки - сервер, на якому відбувається обробка запитів і відповідей. Так само його ще називають серверним шаром. Тут відбуваються всі логічні операції: операції з даними, звернення до інших сервісів або сховищ даних. Шар даних - сервер баз даних, до нього звертається сервер. У цьому шарі зберігається вся необхідна інформація, якою користується додаток при роботі. Таким чином, сервер приймає на себе обов'язок по зверненню до даних, не даючи можливості користувачеві звернутися до них безпосередньо. Трирівнева архітектура використовується для великих і складних проектів з недоступними для користувача базами даних, для збереження і захисту даних. Використовуючи таку архітектуру, отримуємо чимало плюсів, основні з них: можливість побудувати захист, при атаці зловмисника на сховище даних; можливість регулювати доступ до призначених для користувача даних; можливість редагувати і модифікувати дані перед відправкою їх клієнту, так само завдяки подібній системі ми отримуємо можливість скоротити обсяг інформації, який відправимо клієнту і тим самим знизимо вимоги до якості з'єднання; можливість розширити програму на кілька серверів, при тому що вони все ще залишатимуться з можливістю використовувати ту ж саму базу даних. Рис. 1.3. Трирівнева архітектура Описані вище моделі відображають сучасні підходи до архітектурної побудови клієнт-серверних додатків. Протоколи взаємодії клієнта і сервера Протокол визначає загальний набір правил і сигналізує, що комп'ютери в мережі використовують для зв'язку. Існує безліч протоколів, кожен з яких має переваги перед іншими. Найпопулярніші з них: [TCP](#) / [IP](#) Протокол управління передачею ([TCP](#)), Інтернет-протокол ([IP](#)). Спочатку він був розроблений Міністерством оборони США, щоб дозволити різним комп'ютерам спілкуватися. Сьогодні, як багато хто з нас знають, цей протокол використовується в якості основи для Інтернету. Оскільки [TCP](#) / [IP](#) повинен охоплювати такі великі відстані і перетинати кілька невеликих мереж, він є маршрутизацією протоколом, що означає, що він може відправляти дані через маршрутизатор по дорозі до місця призначення. В кінцевому підсумку це трохи уповільнює роботу, але ця можливість робить її дуже гнучкою для великих мереж. [IPX](#) / [SPX](#) Розроблено компанією [Novell](#). [IPX](#) схожий на оптимізований [TCP](#) / [IP](#). Це теж маршрутизація протокол, що робить його зручним для великих мереж, але він забезпечує більш швидкий доступ по мережі, ніж [TCP](#) / [IP](#). Недоліком є те, що він погано працює по аналогових телефонних лініях. [IPX](#) постійно перевіряє статус передачі, щоб бути впевненим, що всі дані надходять. Це вимагає додаткової смуги пропускання, коли у аналогових телефонних ліній для початку не так багато. Це призводить до повільного доступу. Звичайно, з [IPX](#) дані більш

надійні. [NetBEUI](#) Цей протокол, розроблений [Microsoft](#), розроблений для невеликих локальних мереж і працює досить швидко. У ньому відсутні накладні витрати на адресацію [TCP / IP](#) і [IPX](#), що означає, що його можна використовувати тільки в локальних мережах. Ви не можете отримати доступ до мереж через маршрутизатор. Архітектура [TCP / IP](#) Набір протоколів [TCP / IP](#) складається з наступних компонентів: мережевий протокол ([IP](#)) і його логіка маршрутизації, три транспортних протоколу ([TCP](#), [UDP](#) і [ICMP](#)), а також ряд служб сеансів, презентацій і додатків. [IP](#) представляє мережевий рівень. Кожній системі призначається унікальний мережевий адрес, незалежно від того, чи підключена вона до локальної або глобальної мережі. [TCP](#) надає транспортні послуги по [IP](#). Він орієнтований на з'єднання, тобто вимагає, щоб між двома сторонами був встановлений сеанс для надання своїх послуг. Він забезпечує наскрізну передачу даних, усунення помилок, упорядкування даних і управління потоком. [TCP](#) забезпечує той вид зв'язку, який користувачі та програми очікують в локально підключених сеансах. Цікаво, що навколишнє середовище взаємозалежної локальної мережі демонструє багато з тих же характеристик, що і середовище, для якої був розроблений [TCP / IP](#). Зокрема: Маршрутизація: міжмережам потрібна підтримка маршрутизації; маршрутизація дуже ефективна в середовищах [TCP / IP](#). З'єднання в порівнянні з режимом без встановлення з'єднання: активність [LAN](#) включає обидва; то набір протоколів [TCP / IP](#) ефективно підтримує обидва в рамках інтегрованої структури. Чутливість до адміністративного навантаження: адміністративна підтримка [LAN](#) зазвичай обмежена. Середовища [TCP / IP](#) містять величезну кількість динамічних можливостей, в яких пристрої та мережі виявляються динамічно, а таблиці маршрутизації автоматично підтримуються і синхронізуються. Мережі мереж: [TCP / IP](#) забезпечує виняткову гнучкість в якості адміністративного підходу до управління об'єднаннями мереж. Використовуючи переваги свого динамічного характеру, він забезпечує дуже незалежне управління частинами мережі (при необхідності). Особливості розробки клієнт-серверних додатків і їх реалізація на мові програмування [Java](#) [Java](#) - це мова програмування, винайденна [Sun Microsystems](#). Мета [Sun](#) полягала в тому, щоб дозволити програмістам створити одну копію програми, яку користувачі могли б запускати практично на будь-якому комп'ютері і операційній системі. Ця можливість була розроблена, щоб зробити [Java](#) життєво важливим компонентом програмування в Інтернеті. Програми, написані на [Java](#), переводяться в свою власну архітектуру в форматі, званому байт-кодами. Щоб запустити програму [Java](#), користувачеві потрібна інша програма, яка може інтерпретувати програму [Java](#) і надати їй необхідне середовище і послуги. Програмний рівень, віртуальна машина [Java](#) ([JVM](#)), робить практично будь-яку апаратну і програмну платформу схожою на програму [Java](#). Фактично, [JVM](#) - це драйвер пристрою для програм [Java](#). Запуск [Java](#)-програми з [JVM](#) в даний час повільніше, ніж запуск еквівалентної [C](#)-програми, але технологія [JVM](#) поліпшується швидкими темпами. Архітектура клієнт-сервер має різні характеристики, які спрощують комунікацію. Модель клієнт-сервер є однією з центральних ідей мережевих обчислень. Велика кількість додатків, використовують модель клієнт-сервер, включаючи основну програму інтернету, [TCP / IP](#). Деякі особливості архітектури клієнт-сервер: обмін повідомленнями: клієнт - сервер це система, яка слабо пов'язана і взаємодіє через механізм передачі повідомлень. Повідомлення працює як механізм доставки запитів і відповідей; масштабованість: можливість модифікувати дані перед відправкою їх клієнту; аутентифікація; надійність: сервер не повинен очікувати, що клієнтський код буде вільним від помилок; загальні ресурси: сервер має можливість обслуговувати клієнтів одночасно і регулювати їх загальний доступ до загальних ресурсів. Система клієнт-сервер стала архітектурою обчислень і додатків по всьому світу. Тобто система клієнт - сервер наближає обробку додатків, саме, до користувача, а так само допомагає поліпшити продуктивність. Зв'язок між клієнтом і сервером з використанням сокетів. Серед понять і термінів, пов'язаних з роботою в мережі, в [Java](#) є таке важливе поняття як

” Цитування: **0,01%**

id: **13**

«сокет».

Воно позначає точку, через яку відбувається з'єднання. Використання сокетів дозволяє створити підключення двох комп'ютерів. Простіше кажучи, сокет з'єднує в мережі дві програми. Також, сокети дозволяють програмісту розглядати мережеве з'єднання як ще один потік, в який можна записувати або читати байти. Історично сокети є розширенням однієї з ідей [UNIX](#): усе введення-виведення має виглядати для програмістів як файловий ввід-вивід, незалежно від того, чи працюють вони з клавіатурою, графічним дисплеєм, звичайним файлом або підключення до мережі. Сокети захищають програміста від низькорівневих деталей мережі, таких як типи носіїв, розміри пакетів, повторна передача пакетів, мережеві адреси і багато іншого. Сокет може виконувати наступні основні операції: підключити до віддаленої машини; забезпечувати обмін даними; закривати з'єднання; зв'язуватися з портом; очікувати сигнал входу даних; приймати з'єднання з віддалених машин на прив'язаному порті. Припускаючи, що ми використовуємо [TCP](#) для реалізації, орієнтованої на з'єднання, [Java](#) має два пов'язаних класу, які необхідні, тобто клас [Socket](#) і клас [ServerSocket](#). Клас сокета використовується як клієнтом, так і сервером і має методи, відповідні першим чотирьом з цих операцій. Останні три необхідні тільки сервера і реалізуються класом [ServerSocket](#). Клас [Socket](#) реалізує ідею сокета. Через його канали, введення і виведення, будуть спілкуватися клієнт з сервером, по суті це так звані "розетки". Оголошується цей клас на стороні клієнта, а сервер відтворює його, отримуючи сигнал на підключення. Так відбувається спілкування в мережі. Тому, найпростішим варіантом реалізації клієнт - серверного додатка на [Java](#) буде на основі сокетів. У цьому методі і сервер, і клієнт є додатком [Java](#). Клієнтський додаток повинен бути завантажено на клієнтський комп'ютер. Сервер може працювати одночасно, тобто обслуговувати більше одного клієнта одночасно. Сервер і база даних знаходяться на сервері. Основним недоліком цього методу, як уже говорилося раніше, є те, що програмне забезпечення для клієнта має бути фізично присутнім на клієнтській машині, а клієнтська машина повинна мати інтерпретатор [Java](#) для його запуску. Якщо в клієнтську програму вносяться певні зміни, його необхідно поширити серед клієнтів. Висновки до розділу 1 У даному розділі була розглянута теоретичні основи розробки клієнт-серверних додатків. Під час дослідження було розглянуто клієнт-серверну архітектуру, визначено ролі компонентів (клієнта, сервера) в ній, розглянуто основні моделі (однорангова, дворангова, трьохрангова). Також проаналізовано основні протоколи, які можуть застосовуватися у клієнт-серверних архітектурах ([TCP / IP](#), [IPX](#) / [SPX](#), [NetBEUI](#)). Крім того, розглянуто основні

можливості мови програмування [Java](#) в розробці клієнт – серверного додатку та визначено способи реалізації на мові програмування [Java](#) майбутнього додатку. Таким чином, визначено, що додаток

Цитування: 0,03%

id: 14

«Корпоративний чат»

буде мати клієнт-серверну архітектуру, із застосуванням мережевого протоколу [TCP](#) та технологією сокетів як інтерфейсу взаємодії клієнта та сервера для можливості обміну текстовими повідомленнями. РОЗДІЛ 2. РОЗРОБКА ПРОГРАМНОГО ДОДАТКА

Цитування: 0,03%

id: 15

“КОРПОРАТИВНИЙ ЧАТ”

НА ПЛАТФОРМІ [JAVA](#) 2.1. Короткий огляд структури додатку В ході розробки був структурований існуючий матеріал і на його основі реалізовано клієнт-серверний додаток. На початку розробки необхідно вибрати між дворівневою і багаторівневою архітектурою. Для створення клієнт-серверного додатка була обрана дворівнева архітектура і об'єктно-орієнтована мова програмування - [Java](#), для написання всієї програми. Вся обробка даних відбувається на сервері. Дворівневу архітектуру легко можна реалізувати. Структурна модель будувалася таким чином що б на будь-якому етапі була можливість змінювати і додавати новий функціонал. Рис.2.1. Структурна модель додатка "Корпоративний чат". А саме ця модель складається з двох модулів [client](#) і [server](#). До клієнтської частини відносяться такі класи як: [class ClientGuiController](#), [class ClientGuiModel](#), [ClientGuiView](#), [BotClient](#). До серверної частини відносяться класи: [class Server](#), [enum MessageType](#), [class Message](#), [class ConsoleHelper](#), [class Connection](#). Подібна структурна модель демонструє поля, члени класів і методи для роботи з ними в клієнтській і серверній частинах. 2.2. Розробка серверного додатку Найбільш значущим у сервері є код його основного класу "[Server](#)". Розглянемо основні поля, методи класу

Цитування: 0,01%

id: 16

"[Server](#)":

Сервер повинен підтримувати безліч з'єднань з різними клієнтами одночасно. Це можна реалізувати за допомогою наступного алгоритму: - Сервер створює серверне сокетне з'єднання. - У циклі очікує, коли якийсь клієнт підключиться до сокету. - Створює новий потік обробник [Handler](#), в якому буде відбуватися обмін повідомленнями з клієнтом. - Чекає наступне з'єднання. Оскільки сервер може одночасно працювати з декількома клієнтами, нам знадобиться метод для відправки повідомлення відразу всім. Додамо до класу [Server](#): 1. Статичне поле [Map String, Connection connectionMap](#), ключем буде ім'я клієнта, а значення - з'єднання з ним. [private static Map String, Connection connectionMap = new ConcurrentHashMap\(\);](#) 2. Статичний метод [void sendBroadcastMessage \(Message message\)](#), який повинен відправляти повідомлення [message](#) усім з'єднанням з [connectionMap](#). Якщо під час надсилання повідомлення трапиться виключення [IOException](#), потрібно відловити його і повідомити користувачеві що не змогли відправити повідомлення. [public static void sendBroadcastMessage\(Message message\) { for \(Connection connection : connectionMap.values\(\)\) { try{ connection.send\(message\); }catch \(IOException exception\){ ConsoleHelper.writeMessage\("Не змогли отправить сообщение " + connection.getRemoteSocketAddress\(\)\); } } } Не менш важливим методом є \[void notifyUsers \\(Connection connection, String userName\\) throws IOException\]\(#\) {}, він повинен відправляти через \[connection\]\(#\) повідомлення про те, що був доданий користувач з ім'ям \[name\]\(#\) для кожного імені з \[connectionMap\]\(#\). \[private void notifyUsers\\(Connection connection, String userName\\) throws IOException { for\\(String name : connectionMap.keySet\\(\\)\\){ if\\(name.equals\\(userName\\)\\){ continue; } connection.send\\(new Message\\(MessageType.USER_ADDED, name\\)\\); } } Також додамо метод який буде виконувати цикл обробки повідомлень сервером \\[void serverMainLoop\\\(Connection connection, String userName\\\) throws IOException, ClassNotFoundException\\]\\(#\\), де значення параметрів таке ж, як і у методі \\[notifyUsers\\\(\\\)\\]\\(#\\). Метод \\[serverMainLoop\\\(\\\)\\]\\(#\\) повинен в нескінченному циклі отримувати повідомлення від клієнта, якщо повідомлення, отримане методом \\[serverMainLoop\\\(\\\)\\]\\(#\\), має тип \\[MessageType.TEXT\\]\\(#\\), то повинно бути відправлено нове повідомлення всім учасникам чату використовуючи метод \\[sendBroadcastMessage\\\(\\\)\\]\\(#\\). Якщо повідомлення отримане предметом \\[serverMainLoop\\\(\\\)\\]\\(#\\), має тип відмінний від \\[MessageType.TEXT\\]\\(#\\), метод \\[sendBroadcastMessage\\\(\\\)\\]\\(#\\) не повинен бути викликаний, і має бути виведено повідомлення про помилку. \\[private void serverMainLoop\\\(Connection connection, String userName\\\) throws IOException, ClassNotFoundException { while\\\(true\\\){ Message message = connection.receive\\\(\\\); if\\\(message.getType\\\(\\\) == MessageType.TEXT\\\){ String data = message.getData\\\(\\\); sendBroadcastMessage\\\(new Message\\\(MessageType.TEXT, userName +\\]\\(#\\)\]\(#\)](#)

Цитування: 0,01%

id: 17

": "

+ [data](#)));

}else{

[ConsoleHelper.writeMessage](#)(

Цитування: 0,04%

id: 18

"Получено сообщение от "

+ [socket.getRemoteSocketAddress\(\)](#) +

| | |
|--|--------|
| Цитування: 0,06% | id: 19 |
| ". Тип сообщения не соответствует протоколу." | |
| <pre>); } } } Клас Handler реалізує протокол спілкування з клієнтом, за допомогою основного методу класу: run() - який буде викликати усі допоміжні методи. 1. Метод run() виводить на екран повідомлення про те що було встановлено з'єднання з віддаленою адресою. 2. створює нове з'єднання (connection) використовуючи в якості параметра поле socket. 3. Викликає метод serverHandshake() використовуючи в якості параметра щойно створене з'єднання. 4. Викликає метод sendBroadcastMessage() використовуючи в якості параметра нове повідомлення. 5. Викликає метод notifyUsers() використовуючи в якості параметрів connection і userName. 6. В кінці метод run() видаляє з connectionMap запис відповідну userName, і відправляє всім учасникам чату повідомлення про те, що поточний користувач був видалений. 7. Коректно обробляє виключення IOException і ClassNotFoundException. public void run() { ConsoleHelper.writeMessage(</pre> | |
| Цитування: 0,05% | id: 20 |
| "Установленно новое соединение с " | |
| <pre>+ socket.getRemoteSocketAddress()); String userName = null; try(Connection connection = new Connection(socket)){ userName = serverHandshake(connection); // повідомляємо всім учасникам що приєднався новий учасник sendBroadcastMessage(new Message(MessageType.USER_ADDED , userName)); // повідомляємо новому учаснику про існуючі учасників notifyUsers(connection, userName); // обробляємо повідомлення користувачів serverMainLoop(connection , userName); }catch (IOException ClassNotFoundException exception){ ConsoleHelper.writeMessage("Ошибка при обмене данными с " + socket.getRemoteSocketAddress()); } if(userName != null){ connectionMap.remove(userName); sendBroadcastMessage(new Message(MessageType.USER_REMOVED , userName)); } ConsoleHelper.writeMessage("Соединение с " + socket.getRemoteSocketAddress() + " закрыто."); } 2.3. Розробка клієнтського додатку Клієнт, повинен запросити у користувача адресу і прот сервера, під'єднатися зазначеною адресою, отримати запит імені від сервера, запитати ім'я у користувача, відправити ім'я користувача сервера, дочекатися прийняття імені сервером. Після цього клієнт може обмінюватися текстовими повідомленнями з сервером. Обмін повідомленнями відбувається в двох паралельно працюючих потоках. Один займається читанням з консолі і відправкою прочитаного сервера, а другий потік отримує дані від сервера і виводить їх в консоль. Так як клієнт реалізований з графічним інтерфейсом, то доречно було скористатися патерном MVC (Model-View-Controller). При розробці систем з призначенням для користувача інтерфейсом, слідуючи паттерну MVC потрібно розділяти систему на три складові частини. Їх, в свою чергу, можна називати модулями або компонентами. У кожній складовій компоненти буде своє призначення. Model - так звана модель. Вона містить всю логіку програми. View - вид, друга частина системи. Даний модуль відповідає за відображення даних користувачеві. Все, що бачить користувач, генерується видом. Controller - контролер. У ньому зберігається код, який відповідає за обробку дій користувача (будь-яка дія користувача в системі обробляється в контролері). Model (Модель) - найбільш незалежна частина системи. Модель не повинна нічого знати про модуль Вид і Контролер. Модель настільки незалежна, що її розробники можуть практично нічого не знати про Вид і Контролер. Основне призначення View (Виду) - надавати інформацію з Моделі в зручному для сприйняття користувача форматі. Основне обмеження Виду - він ніяк не повинен змінювати модель. Основне призначення Контролера - обробляти дії користувача. Саме через Контролер користувач вносить зміни в модель. Точніше в дані, які зберігаються в моделі. За допомогою MVC, можна спростити написання програм, підвищити читаність коду, зробити легше розширення і підтримку системи в майбутньому. З усього цього можна зробити цілком логічний висновок. Будь-яку систему потрібно розбивати на модулі. Найбільш значущім класом клієнта є</pre> | |
| Цитування: 0,01% | id: 21 |
| " ClientGuiController :" | |
| - цей клас відповідає за обробку дій користувача (будь-яка дія користувача в системі обробляється в контролері). Розглянемо основні методи клієнтського класу | |
| Цитування: 0,01% | id: 22 |
| " ClientGuiController :" | |
| <p>Вкладення клас GuiSocketThread в якому перевизначені важливі методи: 1. void processIncomingMessage(String message) - встановлює нове повідомлення моделі і викликає оновлення виведення повідомлень у представника. protected void processIncomingMessage(String message) {</p> <pre>// Виводимо текст повідомлення model.setNewMessage(message); view.refreshMessages(); } 2. void informAboutAddingNewUser(String userName) - додає нового користувача в модель і викликає оновлення виведення користувачів у відображення. protected void informAboutAddingNewUser(String userName) { //виводимо інформацію про додавання нового користувача model.addUser(userName);</pre> | |

```

view.refreshUsers());
} 3. void informAboutDeletingNewUser (String userName) - видалляє користувача з моделі і
викликає оновлення виведення користувачів у відображення. protected void
informAboutDeletingNewUser(String userName) {
//виводимо актуальний список користувачів
model.deleteUser(userName);
view.refreshUsers();
} 4. void notifyConnectionStatusChanged (Boolean clientConnected) - викликає аналогічний
метод у уявлення. protected void notifyConnectionStatusChanged(boolean clientConnected) {
view.notifyConnectionStatusChanged(clientConnected);
} 2.4. Розробка бота Бот є клієнтом, який автоматично відповідає на деякі команди (час,
дату). Тому реалізований бот відправляє поточний час і дату. Розглянемо основні методи
класу BotClient, а саме getUsername() і processIncomingMessage(String message): 1.
getUsername() - генерує нове ім'я бота на випадок, якщо до сервера підключиться кілька
ботів, наприклад: date_bot_x, де x - будь-яке число від 0 до 99. protected String
getUsername() {
return "date_bot_" + (int) (Math.random() * 100); } 2. processIncomingMessage (String message)
- обробляє вхідні повідомлення: а) Виводить в консоль текст отриманого повідомлення
(message). б) Отримує з message ім'я відправника і текст повідомлення. в) Відправляє
відповідь в залежності від тексту прийнятого повідомлення, Наприклад, якщо текст
повідомлення:

```

| | |
|---|--------|
| Цитирования: 0,01% | id: 23 |
| "Дата" | |
| - відправить повідомлення яке містить поточну дату. | |
| Цитирования: 0,01% | id: 24 |
| "День" | |
| - відправить дату (тільки поточний день). | |
| Цитирования: 0,01% | id: 25 |
| "Місяць" | |
| - відправить дату (тільки поточний місяць). | |
| Цитирования: 0,01% | id: 26 |
| "Рік" | |
| - відправить дату (тільки поточний рік). | |
| Цитирования: 0,01% | id: 27 |
| "Час" | |
| - відправить поточний час. | |
| Цитирования: 0,01% | id: 28 |
| "Час" | |
| - відправить поточну годину. | |
| Цитирования: 0,01% | id: 29 |
| "Хвилини" | |
| - відправить поточну хвилину. | |
| Цитирования: 0,01% | id: 30 |
| "Секунди" | |
| - відправить поточну секунду. Відповідь містить ім'я клієнта, який надіслав запит і чекає на відповідь. protected void processIncomingMessage(String message) { | |
| // Виводимо текст повідомлення в консоль | |
| ConsoleHelper.writeMessage(message); | |
| // Відокремлюємо відправника від тексту повідомлення | |
| String userNameDelimiter = | |
| Цитирования: 0,04% | id: 31 |
| ": "; | |
| String[] | |
| split = message.split(userNameDelimiter); | |
| if (split.length != 2) return; | |
| String messageWithoutUserName = split[1]; | |
| // Готуємо формат для відправки дати згідно із запитом | |
| String format = null; | |
| switch (messageWithoutUserName) { | |
| case | |
| Цитирования: 0,03% | id: 32 |
| "дата": | |
| format = | |
| Цитирования: 0,04% | id: 33 |
| "d.MM.yyyy" | |
| ; | |
| break; | |
| case | |
| Цитирования: 0,03% | id: 34 |
| "день": | |
| format = | |
| Цитирования: 0,01% | id: 35 |
| "d" | |

```

break;
case
    Цитирования: 0,03% id: 36
    "месяц":
format =
    Цитирования: 0,01% id: 37
    "MMMM"
;
break;
case
    Цитирования: 0,03% id: 38
    "год":
format =
    Цитирования: 0,01% id: 39
    "yyyy"
;
break;
case
    Цитирования: 0,03% id: 40
    "время":
format =
    Цитирования: 0,01% id: 41
    "H:mm:ss"
;
break;
case
    Цитирования: 0,03% id: 42
    "час":
format =
    Цитирования: 0,01% id: 43
    "H"
;
break;
case
    Цитирования: 0,03% id: 44
    "минут":
format =
    Цитирования: 0,01% id: 45
    "m"
;
break;
case
    Цитирования: 0,03% id: 46
    "секунд":
format =
    Цитирования: 0,01% id: 47
    "s"
;
break;
}
if (format != null) {
    String answer = new SimpleDateFormat(format).format(Calendar.getInstance().getTime());
    BotClient.this.sendMessage(
        Цитирования: 0,05% id: 48
        "Информация для " + split[0]
    +
        Цитирования: 0,01% id: 49
        ". "
    + answer);
}

```

2.5. Пример работы dodatka [Java](#), строга мова на якій можна вирішувати будь-яке завдання, завдяки кроссплатформенності може працювати на різних платформах що дуже важливо для клієнтської частини. Алгоритм роботи системи програми виглядає наступним чином: 1.Сервер запускається і чекає введення порту сервера; Рис. 2.2. Запуск сервера 2.Далі після введення порту запущений сервер переходить в режим очікування підключення клієнта; 3. При підключенні клієнта, запускається призначений для користувача інтерфейс; 4.Далі сервер запитує адресу, порт і ім'я клієнта(рис.2.2., рис.2.3.,

рис.2.4.); Рис.2.2. Сервер запитує адресу Рис.2.3. Сервер запитує порт Рис.2.4. Сервер запитує ім'я 5. Після успішного з'єднання сервер обробляє отриману інформацію (рис.2.5., рис.2.6.); Рис.2.5. Сервер обробляє інформацію після успішного з'єднання Рис.2.6. Сервер успішно з'єднався 6. Далі відбувається підключення Бот-клієнта і сервер запитує адресу і порт (рис.2.7.); Рис.2.7. Підключення бота і сервер запитує адресу і порт 7. Після успішного підключення Бот-клієнт виводить інформацію про те які функції він може виконувати при введенні певних команд (рис.2.8.). Рис.2.8. Вивід ботом інформацію про свій функціонал Таким чином, розроблено клієнт-серверний додаток

Цитирования: 0,03%

id: 50

“Корпоративний чат”

та протестовано його інтерфейс та функціональні можливості (додаток А). Висновки до розділу 2 В результаті розробки була спроектована структурна модель додатка

Цитирования: 0,03%

id: 51

“Корпоративний чат”

аналіз якого дозволив розробити, клієнт-серверний додаток за допомогою об'єктно-орієнтованої мови програмування [Java](#). 1. Для Розробки клієнт-серверного додатку було обрано дворівневу архітектуру. 2. Розроблений клієнт-серверний додаток на мові [Java](#), а саме: для оптимізації роботи програми, був використан набір архітектурних ідей і принципів для побудови складних систем з призначенням для користувача інтерфейсом, тобто [MVC](#); розроблений сервер для обміну текстовими повідомленнями, та розглянуті основні методи класу [Server](#); розроблен клієнт з графічним інтерфейсом і можливістю підключатися до сервера і обмінюватися повідомленнями між іншими учасниками та основні методи класу [ClientGuiController](#); розроблений бот клієнт, який може приймати запити і відправляти дані про поточну дату і час. Таким чином, був розроблен клієнт-серверний додаток

Цитирования: 0,03%

id: 52

“Корпоративний чат”

на платформі [Java](#). Після розробки клієнт-серверного додатку було виконано тестування проекту, метою якого було виявлення недоліків розробки. На підставі результатів тестування можна стверджувати, що розроблений додаток є оптимізованим і працюючим. ВИСНОВОК Під час дослідження було проаналізовано науково-технічний матеріал, аналіз якого дозволив розробити клієнт-серверний додаток

Цитирования: 0,03%

id: 53

"Корпоративний чат"

на платформі [Java](#). При цьому було отримано наступні висновки та результати: У першому розділі даної роботи був проведений огляд існуючих архітектур клієнт-серверних додатків, основних компонентів архітектур, моделі, протоколи та особливості в розробці додатків на платформі програмування [Java](#). Проведені дослідження надали можливість визначити, що додаток

Цитирования: 0,03%

id: 54

«Корпоративний чат»

буде мати клієнт-серверну архітектуру, із застосуванням мережевого протоколу [TCP](#) та технологією сокетів як інтерфейсу взаємодії клієнта та сервера для можливості обміну текстовими повідомленнями. В другому розділі роботи було розроблено клієнт-серверний додаток

Цитирования: 0,03%

id: 55

"Корпоративний чат",

а саме: логіка роботи, основні компоненти, реалізована додаткова можливість взаємодії з ботом, який надає набір команд для отримання даних про поточний час тощо. В ході реалізації програми були розглянуті і описані основні класи в розробленому додатку. Серверна частина програми створена з можливістю взаємодії з нею за допомогою консолі, клієнтська частина, реалізація якого виконана за допомогою архітектури [MVC](#), створена з можливістю взаємодії у вигляді інтерфейсу. В результаті, розроблений клієнт-серверний додаток може використовуватися в навчальному контексті, в наслідок чого програма має можливість доопрацьовуватися, додаючи новий функціонал з можливістю рефакторингу коду який служить метою поліпшити якість коду без зміни зовнішньої поведінки програми. СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ Офіційний сайт [Java™ Platform, Standard Edition 7 API Specification](#) : веб-сайт. URL: <https://docs.oracle.com/javase/7/docs/api/overview-summary.html> (дата звернення 20.12.2020 р.) [Java](#) : веб-сайт. URL: <https://uk.wikipedia.org/wiki/Java> (дата звернення 20.12.2020 р.) [Client-server model](#) : веб-сайт. URL: https://en.wikipedia.org/wiki/Client-server_model. (дата звернення 15.12.2020 р.) Офіційний сайт [IntelliJ IDEA](#). URL: <https://www.jetbrains.com/help/idea/creating-and-running-your-first-java-application.html> (дата звернення 10.12.2020 р.) [GitHub](#) : веб-сайт. URL: <https://github.com/enhorse/java-interview/blob/master/README.md#java-collections> (дата звернення 20.12.2020 р.) Блох Д. [Java](#): ефективне програмування, 3-є изд. М.: Лори, 2019. 463с. Мартін Р. Чистий код: створення, аналіз і рефакторинг. К.: Фабула, 2019. 416 с. Герберт Шілдрт, [Java](#) 8. Повне керівництво, 9-е видання. М.: Вільямс. 2015. 1376с. Гамма, Р.Хелм, Р.Джонсон, Дж. Вліссідес, Прийоми об'єктно-орієнтованого проектування. Патерни проектування. С.-Пб: Питер, 2001. 368с. Блінов, І.М., Романчик, В. С. [Java](#). Методи програмування: навчально-методичний посібник. Минск : издательство

Цитирования: 0,03%

id: 56

«Чет ре четверти»,

2013. 896с. Програмування на мові [Java](#) : веб-сайт. URL: <http://shtanyuk.tk/edu/nnit/java-new/html/11.html> (дата звернення 20.01.2021 р.) [Client-Server Application in Java](#). by [Jasmine J. Ahuja](#), [advisor Dr. Howard Blum](#). 1997. – 37. [Java ServerSocket](#) : веб-сайт. URL: <https://www.javatpoint.com/java-serversocket-getremotesocketaddress-method> (дата звернення 05.02.2021 р.) [Java SE. ServerSocket](#). URL: <http://java-online.ru/java-socket.xhtml> (дата звернення 10.03.2021 р.) [Wikipedia. James Gosling](#) : веб-сайт. URL:

при введенні коректної інформації, при підключенні до сервера; Перевірка функціональності

кнопки

Цитирования: 0,01% id: 65

"Exit"

у вікні підключення до сервера; Перевірка функціональності запропонованих команд ботом у вікні чату; Перевірка функціональності введення повідомлень у вікні чату; 4. План робіт
Задача Об'єм робіт Дата початку Дата закінчення Укладання тест-плану 12 годин
01.05.2020 02.05.2021 Виконання тестування 12 години 02.05.2020 03.05.2021 Аналіз
тестування 6 годин 03.05.2020 03.05.2021 Підведення підсумків 6 годин 03.05.2020
04.05.2021 5. Кінцеві результати 5.1 Підсумок Підсумком проведення тестування є
оформлений додаток

Цитирования: 0,01% id: 66

«Тест-план»

(цей документ) та

Цитирования: 0,03% id: 67

«Тестові випадки»,

які містять результат процесу тестування. ТЕСТОВІ ВИПАДКИ Тестування інтерфейсу
Тестова задача: тестування запуску сервера і підключення клієнту до сервера. Консоль
введення порту сервера для його запуску(Рис.А.1). Рис.А.1. Консоль введення порту для
запуску сервера Перевірка підключення клієнта до сервера: Вікно при підключенні клієнта
та для введення адреси сервера працює нормально(Рис. А.2); Рис. А.2. Вікно для введення
адреси сервера Кнопка

Цитирования: 0,01% id: 68

«OK»

працює нормально й запускає наступне вікно для введення порту. (Рис. А.3); Рис. А.3. Вікно
для введення порту сервера Кнопка

Цитирования: 0,01% id: 69

«OK»

працює нормально й запускає вікно для введення ім'я. (Рис. А.3); Рис.А. 3. Вікно для
введення ім'я Кнопка

Цитирования: 0,01% id: 70

«Cancel»

працює нормально та викликає вікно з повідомленням про не підключення до сервера(Рис.
А.4.). Рис.А. 4. Вікно з повідомленням про не підключення до сервера Кнопка

Цитирования: 0,01% id: 71

"OK"

після введення коректних даних при підключенні до сервера, викликає повідомлення про
підключення до серверу(Рис А.5.), і далі після натискання знову кнопки

Цитирования: 0,01% id: 72

"OK"

викликає вікно чату, працює нормально(Рис А.6.). Рис.А. 5. Вікно з повідомленням про
підключення до сервера Рис.А. 6. Вікно чату після підключення клієнта Кнопка

Цитирования: 0,01% id: 73

"Exit"

у вікні підключення до сервера, працює нормально(Рис. А.7). Рис.А. 7. Вікно після
натискання кнопки

Цитирования: 0,01% id: 74

"Exit"

при введенні ім'я клієнта Перевірка функціональності запропонованих команд ботом у вікні
чату, працює нормально(Рис. А.8.); Рис. А. 9. Бот виконує команди користувача. Введення
повідомлень у вікні чату, працює нормально(Рис.А.10.); Рис. А. 10. Діалог між двома
користувачами. Результати тестувань клієнт-серверного додатку на платформі [Java](#), на ПК
([Windows](#) 8.1) показали, що додаток відповідає усім встановленим вимогам. ДОДАТОК В
Вихідний код класів які організовують роботу сервера - [Server.class](#), [ConsoleHelper.class](#),
[Message.class](#), [MessageType.enum](#), [Connection.class](#), [package Chat](#);

```
import java.io.IOException;
import java.net.ServerSocket;
import java.net.Socket;
import java.util.Map;
import java.util.concurrent.ConcurrentHashMap;
public class Server {
    private static Map<String,Connection> connectionMap = new ConcurrentHashMap ();
    public static void main(String[] args) {
        ConsoleHelper.writeMessage(
```

Цитирования: 0,04% id: 75

"Введіть порт сервера:"

```
);
int port = ConsoleHelper.readInt();
try(ServerSocket serverSocket = new ServerSocket(port)){
    ConsoleHelper.writeMessage(
```

Цитирования: 0,04% id: 76

"Чат Сервер запущено."

```
);
while (true){
    Socket socket = serverSocket.accept();
    new Handler(socket).start();
```

```

}
} catch (Exception exception){
ConsoleHelper.writeMessage(
Цитирования: 0,09% id: 77
"Виникла помилка при запуску або роботі сервера."
);
}
}
private static class Handler extends Thread{
Socket socket;
public Handler(Socket socket) {
this.socket = socket;
}
@Override
public void run() {
ConsoleHelper.writeMessage(
Цитирования: 0,05% id: 78
"Установлено новое соединение с "
+ socket.getRemoteSocketAddress());
String userName = null;
try(Connection connection = new Connection(socket)){
userName = serverHandshake(connection);
sendBroadcastMessage(new Message(MessageType.USER_ADDED , userName));
notifyUsers(connection, userName);
serverMainLoop(connection , userName);
} catch (IOException | ClassNotFoundException exception){
ConsoleHelper.writeMessage(
Цитирования: 0,06% id: 79
"Ошибка при обмене данными с "
+ socket.getRemoteSocketAddress());
}
if(userName != null){
connectionMap.remove(userName);
sendBroadcastMessage(new Message(MessageType.USER_REMOVED , userName));
}
ConsoleHelper.writeMessage("Соединение с " + socket.getRemoteSocketAddress() + "
закр_то.");
}
private String serverHandshake(Connection connection) throws IOException,
ClassNotFoundException {
while (true) {
connection.send(new Message(MessageType.NAME_REQUEST));
Message message = connection.receive();
if (message.getType() != MessageType.USER_NAME) {
ConsoleHelper.writeMessage("Получено сообщение от " + socket.getRemoteSocketAddress() +
Цитирования: 0,06% id: 80
". Тип повідомлення не відповідає протоколу."
);
continue;
}
String userName = message.getData();
if (userName.isEmpty()) {
ConsoleHelper.writeMessage(
Цитирования: 0,09% id: 81
"Спроба підключення до сервера з порожнім ім'ям "
+ socket.getRemoteSocketAddress());
continue;
}
if (connectionMap.containsKey(userName)) {
ConsoleHelper.writeMessage(
Цитирования: 0,11% id: 82
"Поп_тка підключення к серверу с уже используем_м именем от "
+ socket.getRemoteSocketAddress());
continue;
}
connectionMap.put(userName, connection);
connection.send(new Message(MessageType.NAME_ACCEPTED));
return userName;
}
}
private void notifyUsers(Connection connection, String userName) throws IOException{
for(String name : connectionMap.keySet()){
if(name.equals(userName)){
continue;
}
connection.send(new Message(MessageType.USER_ADDED, name));
}
}
private void serverMainLoop(Connection connection, String userName) throws IOException,
ClassNotFoundException {
while(true){

```

```

Message message = connection.receive();
if (message.getType() == MessageType.TEXT){
String data = message.getData();
sendBroadcastMessage(new Message(MessageType.TEXT, userName +
"» Цитування: 0,01% id: 83
"; "
+ data));
}else{
ConsoleHelper.writeMessage(
"» Цитування: 0,04% id: 84
"Отримано повідомлення від "
+ socket.getRemoteSocketAddress() +
"» Цитування: 0,06% id: 85
". Тип повідомлення не відповідає протоколу."
);
}
}
}
}

public static void sendBroadcastMessage(Message message) {
for (Connection connection : connectionMap.values()) {
try{
connection.send(message);
}catch (IOException exception){
ConsoleHelper.writeMessage(
"» Цитування: 0,05% id: 86
"Не змогли надіслати повідомлення "
+ connection.getRemoteSocketAddress());
}
}
}
} package Chat;

import java.io.Closeable;
import java.io.IOException;
import java.io.ObjectInputStream;
import java.io.ObjectOutputStream;
import java.net.Socket;
import java.net.SocketAddress;

public class Connection implements Closeable {
private final Socket socket;
private final ObjectOutputStream out;
private final ObjectInputStream in;
public Connection(Socket socket) throws IOException {
this.socket = socket;
this.out = new ObjectOutputStream(socket.getOutputStream());
this.in = new ObjectInputStream(socket.getInputStream());
}
public void send(Message message)throws IOException{
synchronized (out){
out.writeObject(message);
}
}
public Message receive() throws IOException, ClassNotFoundException {
synchronized (in) {
return (Message) in.readObject();
}
}
public SocketAddress getRemoteSocketAddress(){
return socket.getRemoteSocketAddress();
}
public void close()throws IOException{
socket.close();
out.close();
in.close();
}
} package Chat;
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;

public class ConsoleHelper {
private static BufferedReader reader = new BufferedReader(new InputStreamReader(System.in));
public static void writeMessage(String message) {
System.out.println(message);
}
public static String readString() {
while (true) {
try {
String inputString = reader.readLine();
if (inputString != null) {
return inputString;
}
}
}
}
}

```

```
} catch (IOException exception) {
writeMessage(
    Цитирования: 0,11% id: 87
    "Виникла помилка при спробі введення числа. Спробуйте ще раз."
);
}
}
}
public static int readInt() {
while (true) {
try {
int inputNumber = Integer.parseInt(readString().trim());
return inputNumber;
} catch (NumberFormatException exception) {
writeMessage(
    Цитирования: 0,11% id: 88
    "Виникла помилка при спробі введення числа. Спробуйте ще раз."
);
}
}
}
} package Chat;
import java.io.Serializable;

public class Message implements Serializable {
private final MessageType type;
private final String data;
public Message(MessageType type) {
this.type = type;
this.data = null;
}
public Message(MessageType type, String data) {
this.type = type;
this.data = data;
}
public MessageType getType() {
return type;
}
public String getData() {
return data;
}
} package Chat;

public enum MessageType {
NAME_REQUEST, USER_NAME, NAME_ACCEPTED, TEXT, USER_ADDED, USER_REMOVED
} Вихідний код класів які організовують роботу клієнту та бот-клієнта -
ClientGuiController.class, Client.class, ClientGuiModel.class, ClientGuiView.class, BotClient.class.
package Chat.client;

public class ClientGuiController extends Client {
private ClientGuiModel model = new ClientGuiModel();
private ClientGuiView view = new ClientGuiView(this);
@Override
protected SocketThread getSocketThread() {
return new GuiSocketThread();
}
public ClientGuiModel getModel() {
return model;
}
@Override
public void run() {
SocketThread socketThread = getSocketThread();
socketThread.run();
}
public static void main(String[] args) {
ClientGuiController client = new ClientGuiController();
client.run();
}
@Override
protected String getServerAddress() {
return view.getServerAddress();
}
@Override
protected int getServerPort() {
return view.getServerPort();
}
@Override
protected String getUserName() {
return view.getUserName();
}
@Override
protected void sendTextMessage(String text) {
super.sendTextMessage(text);
}
public class GuiSocketThread extends SocketThread {
```



```

@Override
protected void processIncomingMessage(String message) {
    model.setNewMessage(message);
    view.refreshMessages();
}
@Override
protected void informAboutAddingNewUser(String userName) {
    model.addUser(userName);
    view.refreshUsers();
}
@Override
protected void informAboutDeletingNewUser(String userName) {
    model.deleteUser(userName);
    view.refreshUsers();
}
@Override
protected void notifyConnectionStatusChanged(boolean clientConnected) {
    view.notifyConnectionStatusChanged(clientConnected);
}
}
} package Chat.client;

```

```

import javax.swing.*;
import java.awt.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;

```

```

public class ClientGuiView {
    private final ClientGuiController controller;
    private JFrame frame = new JFrame(

```

Цитирования: **0,01%**

id: **89**

"Чат"

```

);
private JTextField textField = new JTextField(50);
private JTextArea messages = new JTextArea(10, 50);
private JTextArea users = new JTextArea(10, 10);
public ClientGuiView(ClientGuiController controller) {
    this.controller = controller;
    initView();
}
private void initView() {
    textField.setEditable(false);
    messages.setEditable(false);
    users.setEditable(false);
    frame.getContentPane().add(textField, BorderLayout.NORTH);
    frame.getContentPane().add(new JScrollPane(messages), BorderLayout.WEST);
    frame.getContentPane().add(new JScrollPane(users), BorderLayout.EAST);
    frame.pack();
    frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    frame.setVisible(true);
    textField.addActionListener(new ActionListener() {
        public void actionPerformed(ActionEvent e) {
            controller.sendTextMessage(textField.getText());
            textField.setText(

```

Цитирования: **0%**

id: **90**

""

```

});
});
}
public String getServerAddress() {
    return JOptionPane.showInputDialog(
        frame,

```

Цитирования: **0,04%**

id: **91**

"Введите адрес сервера:",

```

"Конфигурация клиента",
JOptionPane.QUESTION_MESSAGE);
}
public int getServerPort() {
    while (true) {
        String port = JOptionPane.showInputDialog(
            frame,
            " Введіть порт сервера:

```

Цитирования: **0,01%**

id: **92**

" ,
"

Конфігурація клієнта

Цитирования: **0,28%**

id: **93**

```

" ,
JOptionPane.QUESTION_MESSAGE);
try {

```

```
return Integer.parseInt(port.trim());
} catch (Exception e) {
OptionPane.showMessageDialog(
frame,
"
```

Введено некоректний порт сервера. спробуйте ще раз.

Цитирования: **0,01%**

id: **94**

```
"
"
```

Конфігурація клієнта

Цитирования: **0,21%**

id: **95**

```
"
OptionPane.ERROR_MESSAGE);
}
}
}
public String getUsername() {
return JOptionPane.showInputDialog(
frame,
"
```

Введите ваше имя:

Цитирования: **0,01%**

id: **96**

```
"
"
```

Конфигурация клиента

Цитирования: **0,29%**

id: **97**

```
"
OptionPane.QUESTION_MESSAGE);
}

public void notifyConnectionStatusChanged(boolean clientConnected) {
textField.setEditable(clientConnected);
if (clientConnected) {
OptionPane.showMessageDialog(
frame,
"
```

З'єднання з сервером встановлено

Цитирования: **0,01%**

id: **98**

```
"
"
```

Чат

Цитирования: **0,15%**

id: **99**

```
"
OptionPane.INFORMATION_MESSAGE);
} else {
OptionPane.showMessageDialog(
frame,
"
```

Клиент не подключен к серверу

Цитирования: **0,01%**

id: **100**

```
"
"
```

Чат

Цитирования: **0,2%**

id: **101**

```
"
OptionPane.ERROR_MESSAGE);
}
}
public void refreshMessages() {
messages.append(controller.getModel().getNewMessage() + "
"
```

```
\n"
```

Цитирования: **0,37%**

id: **102**

```
");
}
public void refreshUsers() {
ClientGuiModel model = controller.getModel();
StringBuilder sb = new StringBuilder();
for (String userName : model.getAllUserNames()) {
sb.append(userName).append("
\n");
}
users.setText(sb.toString());
}
} package Chat.client;
```

```
import Chat.Connection;
import Chat.ConsoleHelper;
import Chat.Message;
```

```
import Chat.MessageType;
import java.io.IOException;
import java.net.Socket;

public class Client {
    protected Connection connection;
    private volatile boolean clientConnected = false;
    public static void main(String[] args) {
        Client client = new Client();
        client.run();
    }
    protected String getServerAddress() {
        ConsoleHelper.writeMessage("Введіть адресу сервера:");
        ");
        return ConsoleHelper.readString();
    }
    protected int getServerPort() {
        ConsoleHelper.writeMessage("
Введіть порт сервера:");
        ");
        return ConsoleHelper.readInt();
    }
    protected String getUserNamе() {
        ConsoleHelper.writeMessage("
Введіть ваше ім'я:");
        return ConsoleHelper.readString();
    }
    protected boolean shouldSendTextFromConsole() {
        return true;
    }
    public class SocketThread extends Thread {
        @Override
        public void run() {
            try {
                connection = new Connection(new Socket(getServerAddress(), getServerPort()));
                clientHandshake();
                clientMainLoop();
            } catch (IOException | ClassNotFoundException e) {
                notifyConnectionStatusChanged(false);
            }
        }
        protected void clientHandshake() throws IOException, ClassNotFoundException {
            while (true) {
                Message message = connection.receive();
                if (message.getType() == MessageType.NAME_REQUEST) {
                    String name = getUserNamе();
                    connection.send(new Message(MessageType.USER_NAME, name));
                } else if (message.getType() == MessageType.NAME_ACCEPTED) {
                    notifyConnectionStatusChanged(true);
                    return;
                } else {
                    throw new IOException("Unexpected MessageType");
                }
            }
        }
        protected void clientMainLoop() throws IOException, ClassNotFoundException {
            while (true) {
                Message message = connection.receive();
                if (message.getType() == MessageType.TEXT) {
                    processIncomingMessage(message.getData());
                } else if (MessageType.USER_ADDED == message.getType()) {
                    informAboutAddingNewUser(message.getData());
                } else if (MessageType.USER_REMOVED == message.getType()) {
                    informAboutDeletingNewUser(message.getData());
                } else {
                    throw new IOException("Unexpected MessageType");
                }
            }
        }
        protected void processIncomingMessage(String message){
            ConsoleHelper.writeMessage(message);
        }
        protected void informAboutAddingNewUser(String userName){
            ConsoleHelper.writeMessage("Учасник " + userName +
        ");
        }
        protected void informAboutDeletingNewUser(String userName){
            ConsoleHelper.writeMessage(
        ");
        }
    }
}
```

Цитирования: 0,15% id: 103

Цитирования: 0,15% id: 104

Цитирования: 0,05% id: 105

Цитирования: 0,03% id: 106

```

"Участник '"
+ userName +
" Цитирования: 0,04% id: 107
" покинул чат"
);
}
protected void notifyConnectionStatusChanged(boolean clientConnected) {
Client.this.clientConnected = clientConnected;
synchronized (Client.this) {
Client.this.notify();
}
}
}
protected void sendTextMessage(String text) {
try {
connection.send(new Message(MessageType.TEXT, text));
} catch (IOException exception) {
ConsoleHelper.writeMessage(
" Цитирования: 0,04% id: 108
"Невозможно відправити повідомлення "
);
clientConnected = false;
}
}
protected SocketThread getSocketThread(){
return new SocketThread();
}
public void run(){
SocketThread socketThread = getSocketThread();
socketThread.setDaemon(true);
socketThread.start();
try{
synchronized (this){
wait();
}
}catch (InterruptedException exception){
ConsoleHelper.writeMessage(
" Цитирования: 0,08% id: 109
"Виникла помилка під час роботи клієнта."
);
return;
}
while(clientConnected){
String text = ConsoleHelper.readString();
if(text.equalsIgnoreCase(
" Цитирования: 0,01% id: 110
" exit "
)){
break;
}
if(shouldSendTextFromConsole()){
sendTextMessage(text);
}
}
}
package Chat.client;
import java.util.Collections;
import java.util.Set;
import java.util.TreeSet;

public class ClientGuiModel {
private final Set String allUserNames = new TreeSet ();
private String newMessage;
public Set String getAllUserNames() {
return Collections.unmodifiableSet(allUserNames);
}
public String getNewMessage() {
return newMessage;
}
public void setNewMessage(String newMessage) {
this.newMessage = newMessage;
}
public void addUser(String newUserName) {
allUserNames.add(newUserName);
}
public void deleteUser(String userName) {
allUserNames.remove(userName);
}
} package Chat.client;

import Chat.ConsoleHelper;

```

```
import java.io.IOException;
import java.text.SimpleDateFormat;
import java.util.Calendar;

public class BotClient extends Client {
    @Override
    protected SocketThread getSocketThread() {
        return new BotSocketThread();
    }
    @Override
    protected boolean shouldSendTextFromConsole() {
        return false;
    }
    @Override
    protected String getUsername() {
        return "date_bot_" + (int) (Math.random() * 100);
    }
    public static void main(String[] args) {
        Client client = new BotClient();
        client.run();
    }
    public class BotSocketThread extends SocketThread {
        @Override
        protected void clientMainLoop() throws IOException, ClassNotFoundException {
            String hello = " Привіт чатику. Я робот. Розумію команди: дата, день, місяць, рік, час, година, хвилини, секунди.
```

Цитирования: **0,68%** id: 111

```
        ";
        BotClient.this.sendMessage(hello);
        super.clientMainLoop();
    }
    @Override
    protected void processIncomingMessage(String message) {
        ConsoleHelper.writeMessage(message);
        String userNameDelimiter = ":";
        String[] split = message.split(userNameDelimiter);
        if (split.length != 2) return;
        String messageWithoutUserName = split[1];
        String format = null;
        switch (messageWithoutUserName) {
            case "
```

дата

Цитирования: **0,04%** id: 112

```
        ";
        format = "
```

d.MM.YYYY

Цитирования: **0,05%** id: 113

```
        ";
        break;
        case "
```

день

Цитирования: **0,04%** id: 114

```
        ";
        format = "
```

d

Цитирования: **0,05%** id: 115

```
        ";
        break;
        case "
```

місяць

Цитирования: **0,04%** id: 116

```
        ";
        format = "
```

MMMM

Цитирования: **0,05%** id: 117

```
        ";
        break;
        case "
```

год

Цитирования: **0,04%** id: 118

```
        ";
        format = "
```

yyyy

Цитирования: **0,05%** id: 119

```
        ";
        break;
        case "
```

время

```
        "
```

```
": Цитирования: 0,04% id: 120
format = "
H:mm:ss
" Цитирования: 0,05% id: 121
",
break;
case "
час
" Цитирования: 0,04% id: 122
":
format = "
H
" Цитирования: 0,05% id: 123
",
break;
case "
хвилини
" Цитирования: 0,04% id: 124
":
format = "
m
" Цитирования: 0,05% id: 125
",
break;
case "
секунди:
format =
" Цитирования: 0,01% id: 126
" "
";
break;
}
if (format != null) {
String answer = new SimpleDateFormat(format).format(Calendar.getInstance().getTime());
BotClient.this.sendMessage(
"Цитирования: 0,05% id: 127
"Інформація для " + split[0]
+
" Цитирования: 0,01% id: 128
": "
+ answer);
}
}
```

Этот отчет должен быть правильно истолкован и проанализирован квалифицированным специалистом, который несет ответственность за оценку!

Любая информация, представленная в этом отчете, не является окончательной и подлежит ручному просмотру и анализу. Пожалуйста, следуйте инструкциям: **Рекомендации по оценке**