Міністерство освіти і науки України
Національний технічний університет України
“Київський політехнічний інститут імені Ігоря Сікорського”
Факультет інформатики та обчислювальної техніки
Кафедра інформаційних систем та технологій

Лабораторна робота №7

Технології розроблення програмного забезпечення

«ШАБЛОНИ «MEDIATOR», «FACADE», «BRIDGE», «TEMPLATE METHOD».»

FTP-server

Виконав
студент групи ІА–22:
Білокур Євгеній
Перевірив:
Мягкий Михайло Юрійович

Київ 2024

Зміст

Тема: шаблони «MEDIATOR», «FACADE», «BRIDGE», «TEMPLATE METHOD».

Мета: ознайомитися з шаблонами проектування «MEDIATOR», «FACADE», «BRIDGE», «TEMPLATE METHOD» та набути практичних навичок їх застосування. Реалізувати частину функціоналу програми за допомогою одного з розглянутих шаблонів для досягнення конкретних функціональних можливостей та забезпечення ефективної взаємодії між класами.

Хід роботи

..22 FTP-server (state, builder, memento, template method, visitor, client-server)

FTP-сервер повинен вміти коректно обробляти і відправляти відповіді по протоколу FTP, з можливістю створення користувачів (з паролями) і доступних їм папок, розподілу прав за стандартною схемою (rwe), ведення статистики з'єднань, обмеження максимальної кількості підключень і максимальної швидкості поширення глобально і окремо для кожного облікового запису.

Короткі теоретичні відомості

Принципи проектування:

1. Принцип Don't Repeat Yourself (DRY) свідчить, що в початкових кодах програмного продукту не повинно бути повторень. Ці повторення можуть спостерігатися на різних рівнях - від повторень наборів рядків коду (абсолютно аналогічні рядки в різних місцях проекту) до різної реалізації функціональності, що повторюється. Повторень уникати слід з кількох причин: передусім, код без повторень значно менше і значно легше читається; по-друге, при знаходженні помилки в коді, що має повторення, є високий шанс не знати/забути виправити цю ж помилку у іншому місці; по-третє, при створенні ідентичного коду ви не лише копіюєте функціональність, але і помилки; по-четверте, при необхідності додавання нових можливостей скрізь де є повторення необхідно внести відповідні коректури. Повторення легко виправити застосувавши відповідну техніку рефакторинга - винесення метода/интерфейса/класса та ін.
2. Принцип Keep it simple, Stupid! (KISS) пропонує уникати зайвого ускладнення компонентів системи. Вважається, що система, яка складається з безлічі маленьких простих частин, працює значно надійніше, ніж одна велика складна система. Даною філософією слідує

співтовариство Unix, де кожна програма виконує рівно одну функцію, але при цьому виконує її абсолютно добре (Do one thing right). Це також сприяє задоволенню відомого афоризму «The system obviously has no errors rather than system has no obvious errors» (очевидно, що в системі немає помилок; в системі немає очевидних помилок»). Використання простих конструкцій спрощує читаємість і сприйняття вихідних кодів.

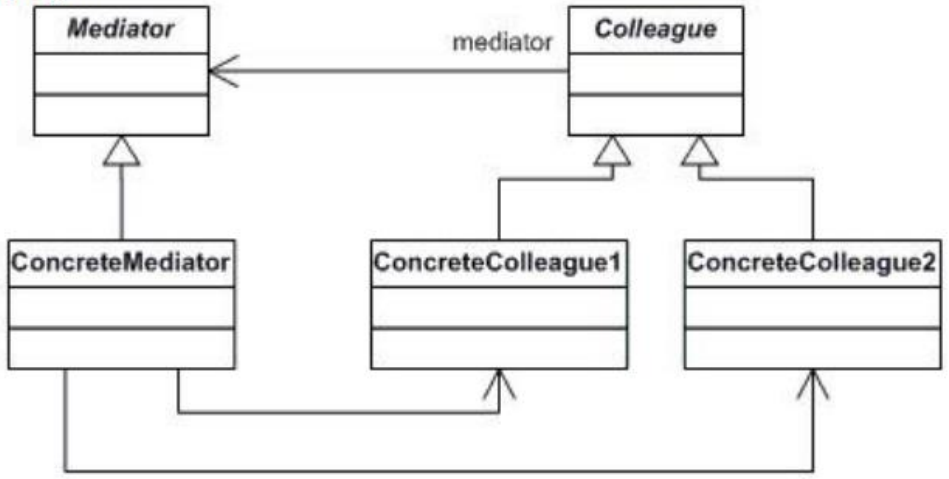
3. Принцип You only load it once! (YOLO) вказує на необхідність підвантаження ініціалізаційних і конфігураційних змінних один раз при запуску програми, щоб уникнути проблем зі швидкістю (повторні зчитування даних з вінчестера).

4. Принцип Парето застосовується до безлічі різних подій і визначає відсоткове співвідношення 80-20. Декілька прикладів застосування принципу Парето: - 80% навантаження на сервер створює 20% додатка; - 80% всього вихідного коду програми пишеться за 20% часу; - 80% помилок програмного забезпечення можна усунути закривши лише 20% багів. Таким чином, необхідно пам'ятати про те, що для вирішення більшості проблем необхідно докласти лише малу частину зусиль. Однак для доведення додатку до стану досконалості може коштувати величезну суму (80%).

5. Принцип You ain’t gonna need it пропонує відмовитися від деяких рішень в силу того, що вони просто не знадобляться. Досить відомий факт, що програмісти люблять узагальнювати та надавати універсальні рішення на всі випадки життя. Дуже часто досить простого і працюючого рішення замість універсального, оскільки інші випадки навряд чи будуть розглядатися в подальшому. Є необхідний набір функціональності, який необхідно реалізувати; додавання зайвої гнучкості не тільки захарастить код, але і зменшить хід процесу розробки.

Шаблон «Mediator» (Посередник):

Структура:



Призначення:
Шаблон «Mediator» забезпечує взаємодію між об'єктами через окремий об'єкт-посередник, замість прямого зв'язку між компонентами. Це дозволяє уникнути складних залежностей і робить код більш гнучким і повторно використовуваним.

Проблема:
При взаємодії елементів діалогу, таких як текстові поля, кнопки та чекбокси, складна логіка може ускладнити повторне використання компонентів у різних контекстах.

Рішення:
Посередник координує взаємодії між елементами, знижуючи їх залежність один від одного і дозволяючи використовувати компоненти в різних контекстах.

Приклад з життя:
Пілоти літаків не спілкуються безпосередньо між собою, а через диспетчера, який координує їх взаємодію.

Переваги:

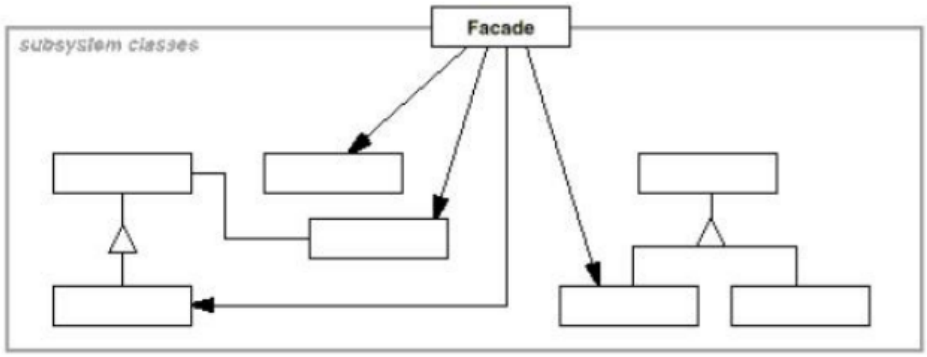
- Знижує залежності між компонентами.
- Спрощує взаємодію та централізує управління.

Недоліки:

- Посередник може стати занадто великим і складним.

Шаблон «Facade» (Фасад):

Структура:



Призначення:
Шаблон «Facade» створює єдиний, уніфікований спосіб доступу до складної підсистеми, приховуючи її внутрішні деталі. Це дозволяє спростити взаємодію з підсистемою, надаючи користувачу простий інтерфейс, при цьому приховуючи складність реалізації.

Проблема:
При роботі з великою кількістю об'єктів складної бібліотеки або фреймворка бізнес-логіка може переплітатися з деталями реалізації сторонніх класів, що ускладнює підтримку коду.

Рішення:
Фасад надає простий інтерфейс для взаємодії з підсистемою, зберігаючи тільки необхідну функціональність для клієнта, приховуючи інші складності.

Приклад з життя:
Якщо ви телефонуєте в магазин для оформлення замовлення, співробітник служби підтримки є фасадом, який спрощує доступ до різних служб (замовлення, оплата, доставка).

Переваги:

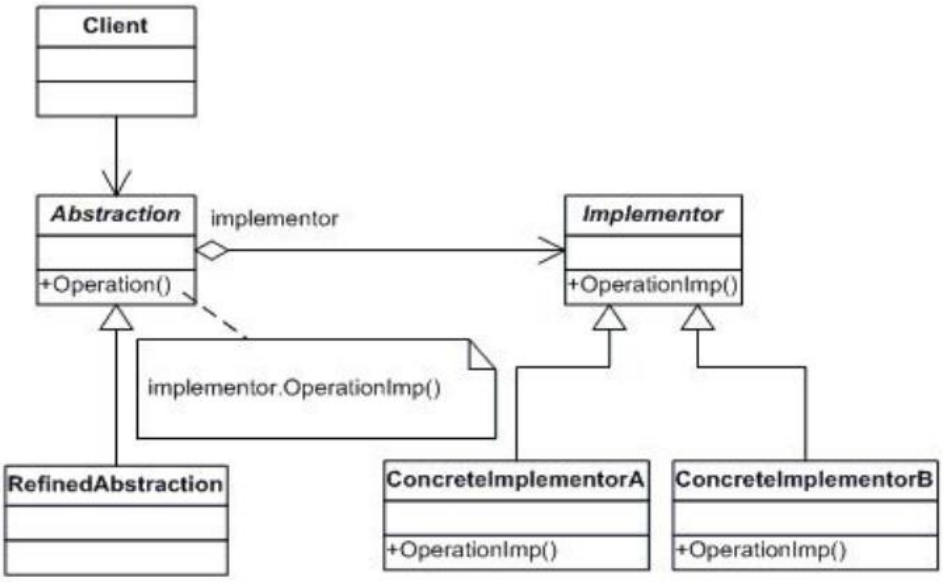
- Ізолює клієнта від складної підсистеми.
- Спрощує використання складних бібліотек або фреймворків.

Недоліки:

- Фасад може стати занадто великим об'єктом, що взаємодіє з усіма компонентами програми.

Шаблон «Bridge» (Міст):

Структура:



Призначення:
Шаблон «Bridge» використовується для поділу абстракції та її реалізації. Це особливо корисно, коли існують різні абстракції, і їх можна реалізувати різними способами. Він дозволяє відокремити абстракцію від її реалізації та додавати нові абстракції незалежно від реалізацій.

Проблема:
Якщо ви намагаєтесь комбінувати кілька властивостей (наприклад, колір і форма), для кожної нової комбінації доведеться створювати нові класи, що призводить до різкого збільшення кількості класів та ускладнення підтримки. Наприклад, у разі додавання нових фігур або кольорів потрібно створювати нові підкласи для кожної можливості, що швидко стає неефективним.

Рішення:
Шаблон «Bridge» вирішує цю проблему, замінюючи спадкування агрегацією або композицією. Винятковим чином, можна створити окрему ієрархію для кожної «площини» (наприклад, колір), а потім використовувати ці об'єкти в класі абстракції (наприклад, фігури). Таким чином, при додаванні нових кольорів не потрібно змінювати класи фігур і навпаки.

Приклад з життя:
Уявіть два міста, розділені річкою. Для з'єднання між ними будують міст. Кожне місто розвивається за своїми законами, але міст залишається сталим, і міста підлаштовуються під міст. Міст є «мостом» між містами, дозволяючи їм взаємодіяти, не змінюючи їхню власну сутність.

Переваги:

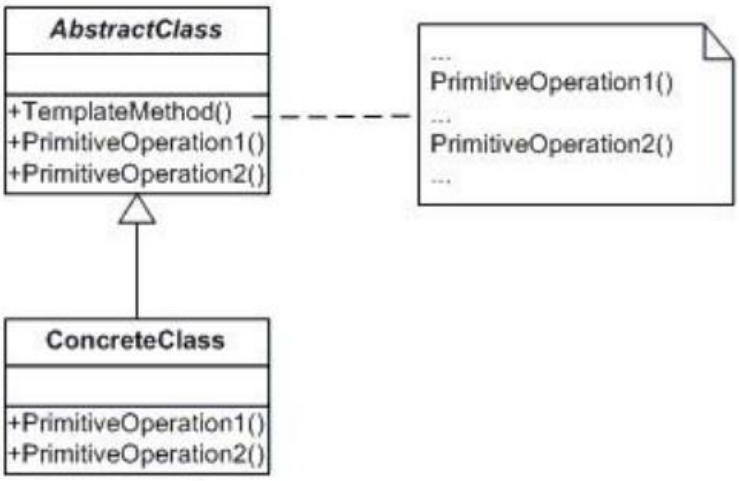
- Дозволяє створювати платформо-незалежні програми.
- Приховує зайві або небезпечні деталі реалізації від клієнтського коду.
- Реалізує принцип відкритості/закритості (легкість додавання нових функціональностей без зміни існуючого коду).

Недоліки:

- Ускладнює код через додавання нових класів і шарів абстракції.

Шаблон «Template Method»

Структура:



Призначення патерну:
Шаблон «Template Method» (шаблонний метод) дозволяє реалізувати покроковий алгоритм в абстрактному класі, але залишити специфіку реалізації для підкласів. Він дозволяє визначити загальний шаблон для алгоритму, а конкретні кроки цього алгоритму залишати на розсуд підкласів. Це зручно, коли потрібно забезпечити однакову структуру алгоритмів, але з можливістю змінювати конкретні кроки в залежності від потреб.

Приклад використання: Уявіть собі систему, яка формує веб-сторінки. Алгоритм формування сторінки включає кілька кроків:

- Формування заголовків,
- Формування контенту,
- Формування файлів, що додаються,
- Формування нижньої частини сторінки.

Ці кроки будуть однаковими для різних типів сторінок (наприклад, HTML, PHP, ASP.NET), але специфіка реалізації деяких кроків (наприклад, додавання вмісту) може варіюватися в залежності від типу сторінки.

Ось приклад на мові C # використання даного шаблону:

```
public class PageFormer
{
     void FormHeaders() { … }
     void FormFooters() { … }
     void FormAddedFiles() { … }
     abstract void FormContent();
     void FormPage()
     {
          FormHeaders();
          FormContent();
          FormFooters();
          FormAddedFiles();
}
}

public class AspNetCompiler : PageFormer
{
     override void FormContent() { … }
}
```

Проблема:
При розробці програми для дата-майнінгу документів, де потрібно обробляти різні формати (наприклад, PDF, DOC, CSV), спочатку обробляються лише DOC-файли. Пізніше додаються інші формати, і з'являється багато повторень в коді для кожного типу файлів, оскільки процес вилучення даних є схожим у всіх випадках.

Рішення:
Шаблон «Template Method» дозволяє описати загальний алгоритм обробки документів, при цьому залишити деталі реалізації (наприклад, як саме зчитувати DOC, CSV або PDF) для підкласів. Таким чином, спільний код з вилучення даних можна реалізувати в базовому класі, а конкретні кроки, які відрізняються для кожного типу файлів, залишити для перевизначення в підкласах.

Приклад з життя:
Будівельники можуть використовувати шаблонний метод при будівництві типових будинків. Зазначено, що будівництво включає стандартні етапи (фундамент, стіни, дах), але на кожному етапі можливі невеликі варіації, наприклад, типи матеріалів або техніка виконання. Саме ці варіації підклас будівельників може реалізувати, при цьому дотримуючись загального плану.

Переваги та недоліки:

Переваги:

- Полегшує повторне використання коду, оскільки спільні частини алгоритму реалізуються в базовому класі.
- Спрощує підтримку, оскільки можна змінювати конкретні кроки, не торкаючись загальної структури алгоритму.

Недоліки:

- Жорстко обмежує можливість зміни структури алгоритму через базовий клас.
- Може порушити принцип підстановки Барбари Лісков, якщо підклас змінює поведінку одного з кроків таким чином, що алгоритм більше не працює коректно.
- При великій кількості кроків шаблонний метод може стати занадто складним для підтримки.

Цей патерн дуже корисний, коли є необхідність забезпечити повторне використання загальних кроків алгоритму, при цьому дозволяючи підкласам змінювати або доповнювати лише частини алгоритму.

Реалізувати не менше 3-х класів відповідно до обраної теми

Структура класів

Структура проекту з реалізованими класами зображена на рисунку 1

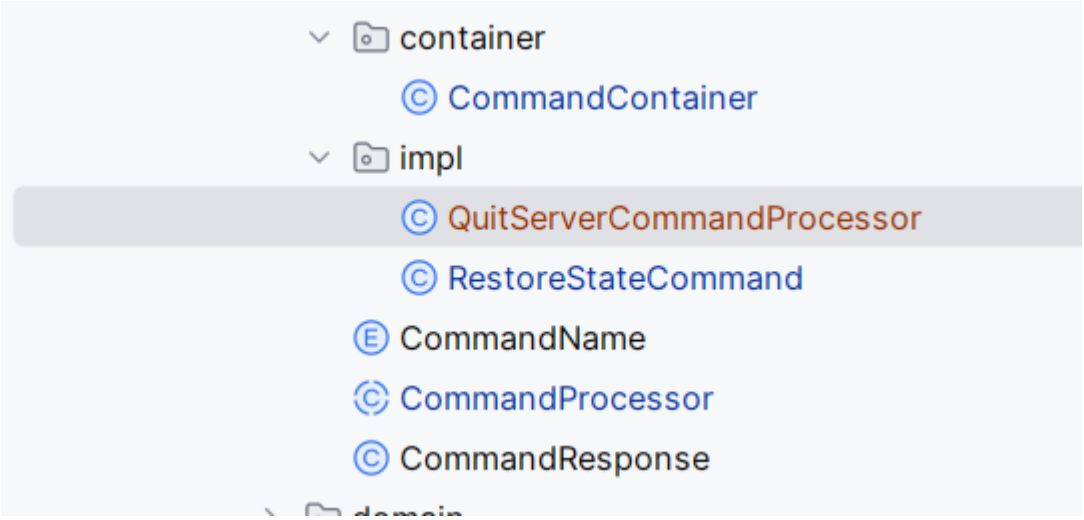


Рисунок 1 – Структура реалізованих класів

Опис класів

CommandProcessor

Цей абстрактний базовий клас реалізує шаблонний метод для обробки FTP-команд.

execute: фінальний метод, який виконує такі дії:

Перевіряє доступ через метод allowAccess.

Перевіряє аргументи за допомогою validArguments.

Виконує специфічну логіку команди в методі process.

Обробляє помилки, повертаючи відповідь з кодом помилки.

Хуки: Методи allowAccess і validArguments мають базову реалізацію, але можуть бути перевизначені.

Абстрактний метод: process обов’язковий для реалізації в підкласах.

2. QuitServerCommandProcessor

Клас реалізує команду QUIT FTP-сервера.

validArguments: Перевіряє, що команда не має аргументів.

process: Завершує сесію, викликаючи setQuitCommandLoop(true), і повертає відповідь з кодом 221 Goodbye.

3. RestoreStateCommand

Клас реалізує команду відновлення стану сесії.

process: Викликає restoreState у сесії користувача та повертає відповідь з кодом 200 State restored successfully.

Реалізувати один з розглянутих шаблонів за обраною темою

Опис шаблону

Template Method у контексті FTP-сервера

Шаблон Template Method дозволяє визначити структуру виконання операції, виділивши загальну логіку в базовому класі, а змінювані частини реалізувати в підкласах. Це зменшує дублювання коду і забезпечує його гнучкість.

В реалізації FTP-сервера шаблон використовується для виконання команд, забезпечуючи чіткий алгоритм, який включає:

Перевірку доступу (allowAccess).

Валідацію аргументів (validArguments).

Виконання специфічної логіки команди (process).

Обробку помилок.

Компоненти реалізації

CommandProcessor (Base Class):

Містить фінальний метод execute, який визначає загальний алгоритм виконання команди.

Містить хуки allowAccess та validArguments, які можна перевизначати.

Метод process є обов'язковим для реалізації в класах-нащадках.

Конкретні команди (QuitServerCommandProcessor, RestoreStateCommand):

Реалізують логіку для специфічних FTP-команд.

Можуть перевизначати validArguments для перевірки аргументів або інші методи за необхідності.

Переваги використання Template Method

- Уніфікація логіки: Загальний алгоритм роботи команди визначений у базовому класі, що зменшує дублювання коду.
- Гнучкість: Дозволяє легко додавати нові команди шляхом реалізації лише змінюваних частин.
- Легкість тестування: Спільна логіка тестується в одному місці, а специфічна — окремо для кожної команди.
- Простота підтримки: Чіткий поділ відповідальностей полегшує додавання нових функцій і зміну існуючих.

Проблема, яку вирішує Template Method

До використання шаблону кожна команда мала б власну повну реалізацію логіки виконання. Це призвело б до дублювання коду та ускладнення підтримки. Template Method спрощує розробку, зосереджуючи загальні кроки у базовому класі, а специфічні деталі — у підкласах. Це робить систему більш масштабованою, дозволяючи легко додавати або змінювати функціонал.

Діаграма класів

Діаграма класів, які реалізовують паттерн Template Method зображена на рисунку 2

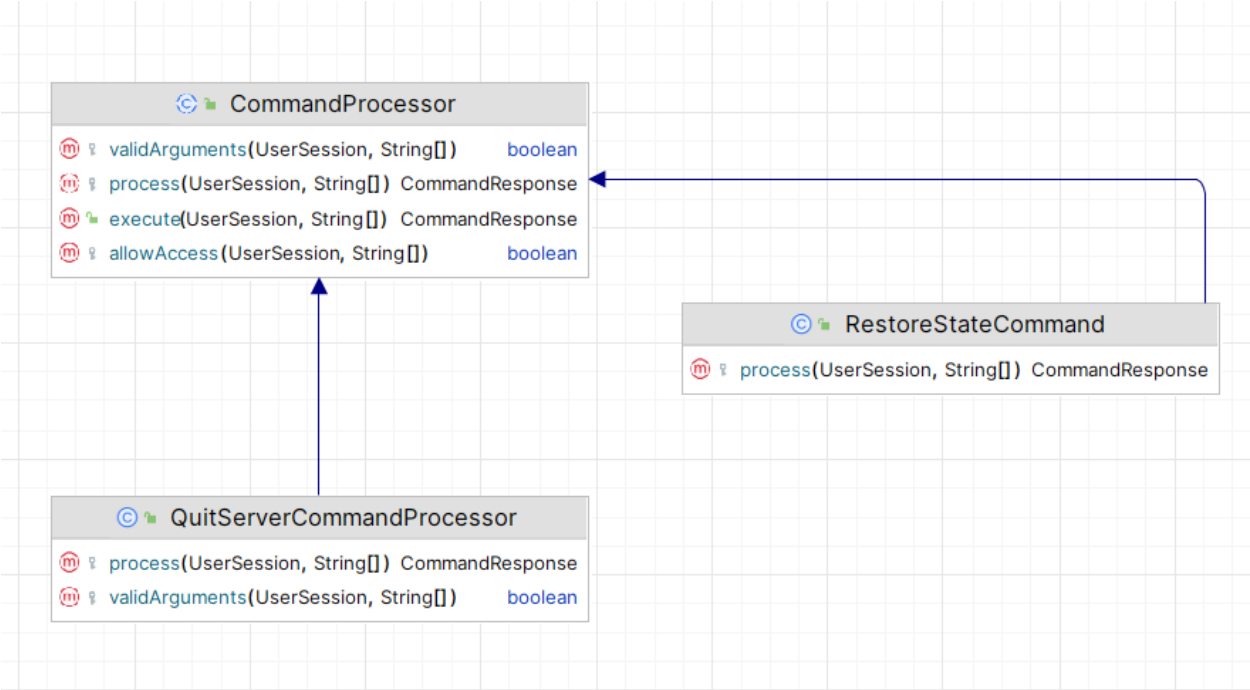


Рисунок 2 – реалізація паттерну Template Method

Висновки та код

Код можна знайти за посиланням - https://github.com/B1lok/trpz

Висновок: У ході виконання лабораторної роботи було реалізовано шаблон проектування **Template Method**. Це дозволило структурувати виконання команд, виділити загальні етапи у базовому класі та спростити додавання нових команд через створення підкласів.

Реалізований підхід продемонстрував переваги уніфікації, зменшення дублювання коду та підвищення масштабованості системи. Використання шаблону вирішило проблему організації складних команд, що підвищило читабельність і легкість підтримки коду.