Введение в программирование на Java
Домашнее задание 2

24 апреля 2025

Описание задания

В рамках данного домашнего задания требуется реализовать небольшой аналог почтового сервера с консольным интерфейсом.

Требования к реализации

Список поддерживаемых команд

Программа должна поддерживать ряд команд, вводимых с клавиатуры.

add — добавление нового пользователя. Для пользователя задаётся имя, по которому его можно идентифицировать.

list — вывод списка всех пользователей.

send – отправка сообщения от одного пользователя другому. Для сообщения указывается заголовок и его текст. Сообщение добавляется в список отправленных у отправителя. Если сообщение не является спамом с точки зрения получателя, то оно добавляется в его список входящих, иначе — в список спама.

inbox — отображение входящих сообщений для конкретного пользователя, не помеченных как спам.

spam — отображение входящих сообщений для конкретного пользователя, помеченных как спам.

outbox — отображение исходящих сообщений для конкретного пользователя.

setfilter — установка спам-фильтра для конкретного пользователя.

Спам-фильтр одного пользователя может состоять из нескольких подфильтров. Если хотя бы один из подфильтров считает сообщение спамом, то сообщение помечается как спам.

Список требуемых к реализации типов фильтров:

- `simple` — сообщение считается спамом, если в заголовке или тексте содержится слово[1] "spam" (в любом регистре).
- `keywords` — сообщение считается спамом, если заголовок или текст содержит одно из ключевых слов (в любом регистре). Список ключевых слов вводится с клавиатуры через пробел.

[1] Здесь и далее употребляется понятие "слово". Под словом подразумевается числобуквенная последовательность, разделённая любыми другими символами. Например, строка "Привет, как дела? Hello,___h0w___ar3___y0u?" содержит слова "Привет", "как", "дела", "Hello", "h0w", "ar3", "y0u".

- `repetition` — сообщение считается спамом, если какое-либо слово в тексте (но не заголовке) встречается чаще, чем заданное число раз. К примеру, при `repetition 3` текст "Hello goodbye hello goodbye hello" не является спамом, но при `repetition 2` (и меньших значениях) — является. Максимальное число повторений вводится с клавиатуры как целое положительное десятичное число.
- `sender` — все сообщения конкретного отправителя считаются спамом. Имя отправителя вводится с клавиатуры.

При вводе команды **setfilter** программа переходит в режим чтения спам-фильтров. Спам-фильтры указываются друг за другом. При вводе строки `done` в качестве типа спам-фильтра их чтение завершается, а получившийся спам-фильтр назначается указанному пользователю.

По умолчанию у пользователя нет ни одного спам-фильтра.

Примеры работы программы (зелёное — пользовательский ввод, чёрное — вывод программы):

```
> add
Enter user name: Vasya
User 'Vasya' added
> add
Enter user name: Petya
User 'Petya' added
> add
Enter user name: Vasya
User 'Vasya' already exists
```

```
> add
Enter user name: Vasya
User 'Vasya' added
> add
Enter user name: Petya
User 'Petya' added
> list
* Vasya
* Petya
Total: 2 users
```

```
> send
Enter sender name: Vasya
Enter receiver name: Petya
Caption:
Заголовок
Text:
Привет, Петя! С уважением, Вася.
Message sent
```

```
> setfilter
Enter user name: Petya
Enter filter: keywords
Enter keywords: гараж
Enter filter: done
Filter is set
> send
Enter sender name: Vasya
Enter receiver name: Petya
Caption:
Заголовок
Text:
Привет, Петя! С уважением, Вася.
Message sent
> send
Enter sender name: Vasya
Enter receiver name: Petya
Caption:
Объявление
Text:
Продам гараж
Message sent
> inbox
Enter user name: Petya
========================================
Заголовок
Привет, Петя! С уважением, Вася.
========================================
> spam
Enter user name: Petya
========================================
Объявление
Продам гараж
========================================
> outbox
Enter user name: Vasya
========================================
Заголовок
Привет, Петя! С уважением, Вася.
========================================
Объявление
Продам гараж
========================================
```

Примечание: конкретный вариант ввода-вывода остаётся на ваше усмотрение. Текст сообщений может быть любым (но на русском или английском языке). Главное — программа должна иметь вышеописанные команды (имена команд менять запрещается), а сообщения должны быть понятны пользователю.

Нюансы программной реализации

В программной реализации обязательно должны быть представлены следующие классы и интерфейсы:

- `Message` — класс, хранящий данные о сообщении.
  - В классе должны быть представлены поля:
    - `String caption`
    - `String text`
    - `User sender`
    - `User receiver`
  - Для всех полей класса должны быть реализованы методы-геттеры.
- `User` — класс, хранящий информацию о пользователе.
  - В классе должны быть представлены поля:
    - `String userName`
    - `Message[] inbox` (разрешается использовать список вместо массива)
    - `Message[] outbox` (разрешается использовать список вместо массива)
    - `Message[] spam` (разрешается использовать список вместо массива)
    - `SpamFilter spamFilter`
  - Для полей `userName`, `inbox`, `outbox`, `spam` должны быть реализованы методы-геттеры, не нарушающие инкапсуляцию.
  - Для поля `spamFilter` должен быть реализован метод-сеттер, позволяющий заменить спам-фильтр пользователя.
  - Класс должен иметь публичный метод, отправляющий указанное сообщение от текущего пользователя пользователю `receiver`:
    - `void sendMessage(String caption, String text, User receiver)`
- `UserStorage` — класс, хранящий всех существующих пользователей.
  - Класс должен иметь публичные методы для:
    - Добавления нового пользователя
    - Получения пользователя по имени
    - Проверки существования пользователя с указанным именем
- `SpamFilter` — интерфейс, который реализуют все спам-фильтры.
  - Интерфейс должен иметь метод `boolean isSpam(Message message)`. Реализации этого метода должны возвращать `true`, если сообщение является спамом.
- `SimpleSpamFilter` — спам-фильтр, считающий спамом сообщения, заголовок или текст которых содержит слово "spam" (в любом регистре). Реализует интерфейс `SpamFilter`.
- `KeywordsSpamFilter` — спам-фильтр, считающий спамом сообщения, заголовок или текст которых содержит хотя бы одно из указанных ключевых слов (в любом регистре). Реализует интерфейс `SpamFilter`.
- `RepetitionSpamFilter` — спам-фильтр по количеству повторений слов в тексте сообщения. Максимальное количество повторений указывается при создании. Сообщение считается спамом, если какое-либо слово в тексте встречается чаще, чем указанное максимальное число раз. Реализует интерфейс `SpamFilter`.
- `SenderSpamFilter` — спам-фильтр, который считает спамом все сообщения отправителя с указанным именем. Реализует интерфейс `SpamFilter`.
- `CompositeSpamFilter` — спам-фильтр, который состоит из нескольких других спам-фильтров. Реализует интерфейс `SpamFilter`. Метод `isSpam` должен возвращать `true`, если хотя бы один из составляющих спам-фильтров вернул `true`.

Примечание: добавлять новые классы, интерфейсы и методы разрешается (и рекомендуется).

Обработка некорректных входных данных

Все некорректные входные данные (например, отправка сообщения несуществующему пользователю или ввод некорректного числа) должны быть корректно обработаны. При вводе некорректных данных программа должна сообщать об ошибке, но продолжать работать.

Все возможные ситуации ошибочных данных заведомо не приводятся. Найти и правильно обработать такие входные данные: одна из задач данного домашнего задания.

Тестирование

Для программы должны быть реализованы модульные тесты с использованием **JUnit5**.

Инструкцию по настройке JUnit можно найти в репозитории курса. Неправильно настроенная библиотека ведёт к снижению оценки.

К оформлению модульных тестов применяются те же правила кодирования, что и ко всему остальному коду.

Помимо этого:

- Все модульные тесты должны что-то проверять (т.е. иметь вызов `assert...` или подобного метода для выполняемых действий). Не считается тестами код, который ничего не проверяет, а лишь делает видимость таковой проверки с целью увеличения покрытия.
- Все модульные тесты должны быть небольшими и тестировать одну конкретную вещь.
- Все модульные тесты должны быть независимыми — результат прохождения тестов не должен зависеть от порядка их запуска.

Правила кодирования

Программа должна быть реализована в объектно-ориентированном стиле.

Исходный код программы должен быть совместим с Java 21 (при разработке рекомендуется использовать OpenJDK 21).

Код **не должен** быть написан в одном единственном классе `Main` и его методе `main`. Программа должна быть декомпозирована на небольшие классы, интерфейсы и методы, каждый из которых решает свою задачу.

Классы должны инкапсулировать свои данные. Не должно быть возможности привести объект в некорректное состояние при помощи каких-то из его методов.

Каждый класс, интерфейс и прочие определения типов должны находиться в отдельных файлах, каждый в своём (исключением являются только статические и нестатические вложенные классы/интерфейсы/т.п.).

Модификаторы доступа должны выбираться осознанно. Отсутствие модификатора доступа (т.е. `package-private` доступ) должно быть обоснованно. Не допускается делать все поля и методы публичными.

Все файлы с исходным кодом должны быть сгруппированы по пакетам. Файлы, не принадлежащие никакому пакету, не допускаются.

Прочие правила кодирования, в соответствии с которыми должна быть оформлена программа, можно найти в соответствующем документе.

Критерии оценивания

Оценка состоит из трёх компонент: реализация, тестирование и оформление.

- **Реализация (4 балла)**
  - 4 балла — описанный функционал полностью реализован, программа корректно обрабатывает все ошибочные ситуации.
  - Оценка понижается за частично реализованный функционал, неправильно обработанные ошибки или прочие отклонения от указанных требований.
- **Тестирование (3 балла)**
  - 3 балла — покрытие строк кода составляет 75% и более. Все тесты успешно проходят. Все тесты соответствуют требованиям к модульным тестам.
  - 2 балла — покрытие строк кода составляет 50-74%, либо при небольших отклонениях от требований к тестам.
  - 1 балл — покрытие строк кода составляет 25-49%, либо при сильных отклонениях от требований к тестам.
  - 0 баллов — покрытие строк кода составляет менее 25% или тесты вовсе отсутствуют.
- **Оформление (3 балла)**
  - 3 балла — код следует всем указанным правилам кодирования.
  - Оценка понижается за нарушение правил кодирования (в зависимости от регулярности и серьёзности).

Отдельные случаи:

- Если код программы не компилируется, выставляется оценка 0 (допускаются исключения на усмотрение проверяющего, если ошибка легко исправляется, а программа в целом корректно реализована).
- При обнаружении несамостоятельного выполнения задания все участники получают оценку 0 (в том числе и настоящий автор программы).

Формат сдачи задания

Задание должно быть загружено в **Smart LMS**.

Дедлайн: 12 мая 2025, 23:59.

В качестве решения должен быть приложен zip-архив, содержащий проект в IntelliJ IDEA с решением, в том числе:

- Директорию .idea и файл *.iml, находящиеся в корне проекта.
- Директорию src с исходными файлами *.java.
- Директорию test с исходными файлами *.java, содержащими модульные тесты на основе JUnit.

При наличии деталей реализации, о которых хотелось бы уведомить проверяющего, рекомендуется создать текстовый файл README с необходимым описанием. Файл README прикладывается в архив.

Прочих "мусорных" файлов в архиве быть **не должно**.

Архив должен иметь имя `HW2_<ГРУППА>_<ФИО>.zip`, например `HW2_ББИ2401_ИвановИванИванович.zip`.

Нарушение правил именования архива, наличие в нём лишних файлов или отсутствие необходимых влечёт за собой **снижение оценки**.

Пример корректного архива:

```
– HW2_ББИ2401_ИвановИванИванович.zip
    – .idea
        – workspace.xml
        – misc.xml
        – modules.xml
        – ...
    – Mailing.iml
    – README.txt
    – src
        – examplepackage
            – User.java
            – ...
    – test
        – examplepackage
            – UserTest.java
            – ...
```