Промежуточная аттестация Модуль 1 «Java Core»

Формулировка задания:

Необходимо реализовать приложение, предоставляющее функциональность работы с пользователем: добавление пользователя в список (регистрация), поиск пользователя по идентификатору, выгрузка информации обо всех пользователях, редактирование пользователя, удаление пользователя/пользователей.

Подробное описание функционала приложения

1. Исходный файл — список пользователей в следующем формате:

f5a8a3cb-4ac9-4b3b-8a65-c424e129b9d2|2023-12-25T19:10:11.556|noisemc_99|789ghs|789ghs|Крылов|Виктор|Павлович|25|true

2. Реализовать классы проекта в папке model:

Класс User

- id типа String – гарантированно уникальный ID пользователя. Состоит из букв и цифр.
- дата LocalDateTime добавления в систему, по умолчанию сегодня, формат: дата и время;
- login типа String, не может быть только из цифр, содержит буквы, цифры, знак подчеркивания, меньше 20 символов
- password и confirmPassword типа String, одинаковые, не может быть только из букв, содержит буквы, цифры, знак подчеркивания, меньше 20 символов
- фамилия - строка, состоит только из букв;
- имя - строка, состоит только из букв;
- отчество - строка, состоит только из букв, может отсутствовать;
- возраст – целое число, может отсутствовать;
- isWorker – является ли сотрудником предприятия, по умолчанию false.

3. Реализовать интерфейс по работе с пользователем в папке repositories:

Интерфейс UsersRepository со следующими абстрактными методами:

- Метод void create(User user) – создание пользователя и запись его в файл;

- Метод User findById(String id) — поиск пользователя в файле по идентификатору;

- Метод List<User> findAll() - выгрузка всех пользователей из файла;

- Метод void update(User user) — обновление полей существующего в файле пользователя;

- Метод void deleteById(String id) — удаление пользователя по идентификатору.

- Метод void deleteAll() — удаление всех пользователей.

4. Реализовать имплементацию UsersRepositoryFileImpl интерфейса UsersRepository по работе с пользователем в папке repositories.

- Метод void create(User user) – создает пользователя и записывает его в список пользователей последним номером. Обновляется файл, туда дописывается пользователь. При формировании объекта User может использоваться функциональный подход с созданием Mapper.

- Метод User findById(String id) — поиск пользователя в файле по идентификатору. Возвращает пользователя или выбрасывает исключение: «Пользователя с заданным идентификатором не существует».

- Метод List<User> findAll() - выгрузка всех пользователей из файла.

- Метод void update(User user) — обновление полей существующего в файле пользователя. Метод находит в файле пользователя с id user-а и заменяет его значения. При отсутствии id в списке формируется сообщение об отсутствии данных о пользователе и создается новый пользователь. При ошибке в требованиях к полям класса выбрасывается исключение.

Примечание: Реализовать замену данных в файле с полной перезаписью файла (заменить в списке параметры пользователя и записать заново в файл

всех пользователей).

- Метод void deleteById(String id) — удаление пользователя по идентификатору. После удаления пользователя переписать файл и записать список без удаленного пользователя. Если пользователя с таким идентификатором нет — выбросить исключения: «Пользователя с заданным идентификатором не существует».

- Метод void deleteAll() — удаление всех пользователей из списка и из файла.

- Дополнительно. Поиск списка пользователей по заданному полю. Например, выгрузка пользователей по возрасту findByAge, выгрузка пользователей-работников предприятия findByIsWorker.

Пользователи хранятся в коллекции ArrayList. Для работы с пользователями предварительно весь список читается из файла. Ошибки чтения/записи в файл должны быть обработаны.

Наименование файла хранится как константа в имплементации интерфейса UsersRepositoryFileImpl.

5. Реализовать класс App для проверки работоспособности приложения\

a. Сформировать файл с набором пользователей.

b. В методе main создать экземпляр класса UsersRepositoryFileImpl. Проверить все функции класса — создание, поиск по id, выгрузка всех пользователей, обновление данных пользователя, удаление по id, удаление всех пользователей, поиск по полю (при наличии данной функции).

6. Дополнительно. Для удобства работы с приложением может быть создан набор своих custom исключений в отдельной папке.

7. Настроить папку с unit-тестами проекта. Подключить к проекту библиотеку JUnit5. Сгенерировать автоматически каркас для тестов с помощью

Intellij Idea.

8. Создать набор unit-тестов для проверки функционала приложения. Создать минимум 4 теста. Тесты должны быть следующих типов:

- Позитивные тесты. Проверка корректности работы методов с обычными данными.

- Тесты на вызов исключений. Проверка работы методов с ошибкой формата ввода.

Программа реализуется в отдельной ветке git attestation/attestation01. При сохранении состояния программы (коммиты) пишется сообщение с описанием хода работы по задаче.

В корне папки с программой должен быть файл .gitignore.

Программа локально коммитится и публикуется в репозиторий GitHub на проверку.

Планируемый результат:

1. Ссылка на программу в репозитории GitHub;
2. Отчёт со скринами выполнения задач - постановка задачи, код задачи и результат в консоли Intellij Idea.

Описания плана работы:

Итоговая работа с отработкой тем Java Core, проверкой приложения через unit-тесты. Обратить внимание на следующие разделы:

- Классы: поля, свойства, методы;
- Класс Object. Класс String;
- Система контроля версий Git;
- Понятия ООП: наследование, инкапсуляция, полиморфизм;
- Инкапсуляция. Модификаторы доступа в Java;
- Абстрактные классы и интерфейсы. Лямбда выражения;
- Java Collections. Stream API;
- Иерархия исключений в Java;
- Ошибки компиляции и ошибки выполнения;
- Понятие unit-тестирования. Работа с программой через unit-тесты.

Перечень инструментов, необходимых для реализации деятельности:

Персональный компьютер, JDK 17 (либо OpenJDK 17), Intellij Idea для разработки на Java, GIT, Tortoise GIT, JUnit5.