

Федеральное государственное автономное образовательное учреждение  
высшего образования “Национальный исследовательский университет ИТМО”

Факультет Программной Инженерии И Компьютерной Техники

Лабораторная работа №5

Вариант 2875

Выполнила:

Абдуллаева София Улугбековна

Группа Р3108

Проверил:

Миху Вадим Дмитриевич

Санкт-Петербург 2025

## Задание

Реализовать консольное приложение, которое реализует управление коллекцией объектов в интерактивном режиме. В коллекции необходимо хранить объекты класса **Organization**, описание которого приведено ниже.

Разработанная программа должна удовлетворять следующим требованиям:

- Класс, коллекцией экземпляров которого управляет программа, должен реализовывать сортировку по умолчанию.
- Все требования к полям класса (указанные в виде комментариев) должны быть выполнены.
- Для хранения необходимо использовать коллекцию типа **java.util.LinkedHashSet**
- При запуске приложения коллекция должна автоматически заполняться значениями из файла.
- Имя файла должно передаваться программе с помощью: **переменная окружения**.
- Данные должны храниться в файле в формате **csv**
- Чтение данных из файла необходимо реализовать с помощью класса **java.io.InputStreamReader**
- Запись данных в файл необходимо реализовать с помощью класса **java.io.BufferedOutputStream**
- Все классы в программе должны быть задокументированы в формате javadoc.
- Программа должна корректно работать с неправильными данными (ошибки пользовательского ввода, отсутствие прав доступа к файлу и т.п.).

В интерактивном режиме программа должна поддерживать выполнение следующих команд:

- **help** : вывести справку по доступным командам
- **info** : вывести в стандартный поток вывода информацию о коллекции (тип, дата инициализации, количество элементов и т.д.)
- **show** : вывести в стандартный поток вывода все элементы коллекции в строковом представлении
- **add {element}** : добавить новый элемент в коллекцию
- **update id {element}** : обновить значение элемента коллекции, id которого равен заданному
- **remove\_by\_id id** : удалить элемент из коллекции по его id
- **clear** : очистить коллекцию
- **save** : сохранить коллекцию в файл
- **execute\_script file\_name** : считать и исполнить скрипт из указанного файла. В скрипте содержатся команды в таком же виде, в котором их вводит пользователь в интерактивном режиме.
- **exit** : завершить программу (без сохранения в файл)
- **add\_if\_max {element}** : добавить новый элемент в коллекцию, если его значение превышает значение наибольшего элемента этой коллекции
- **add\_if\_min {element}** : добавить новый элемент в коллекцию, если его значение меньше, чем у наименьшего элемента этой коллекции
- **history** : вывести последние 15 команд (без их аргументов)

- **remove\_all\_by\_annual\_turnover annualTurnover** : удалить из коллекции все элементы, значение поля annualTurnover которого эквивалентно заданному
- **sum\_of\_annual\_turnover** : вывести сумму значений поля annualTurnover для всех элементов коллекции
- **max\_by\_postal\_address** : вывести любой объект из коллекции, значение поля postalAddress которого является максимальным

### Формат ввода команд:

- Все аргументы команды, являющиеся стандартными типами данных (примитивные типы, классы-оболочки, String, классы для хранения дат), должны вводиться в той же строке, что и имя команды.
- Все составные типы данных (объекты классов, хранящиеся в коллекции) должны вводиться по одному полю в строку.
- При вводе составных типов данных пользователю должно показываться приглашение к вводу, содержащее имя поля (например, "Введите дату рождения:")
- Если поле является enum'ом, то вводится имя одной из его констант (при этом список констант должен быть предварительно выведен).
- При некорректном пользовательском вводе (введена строка, не являющаяся именем константы в enum'e; введена строка вместо числа; введенное число не входит в указанные границы и т.п.) должно быть показано сообщение об ошибке и предложено повторить ввод поля.
- Для ввода значений null использовать пустую строку.
- Поля с комментарием "Значение этого поля должно генерироваться автоматически" не должны вводиться пользователем вручную при добавлении.

### Описание хранимых в коллекции классов:

```
public class Organization {
    private Long id; //Поле не может быть null, Значение поля должно быть больше 0, Значение
    этого поля должно быть уникальным, Значение этого поля должно генерироваться
    автоматически
    private String name; //Поле не может быть null, Строка не может быть пустой
    private Coordinates coordinates; //Поле не может быть null
    private java.time.ZonedDateTime creationDate; //Поле не может быть null, Значение этого поля
    должно генерироваться автоматически
    private long annualTurnover; //Значение поля должно быть больше 0
    private OrganizationType type; //Поле может быть null
    private Address postalAddress; //Поле не может быть null
}
public class Coordinates {
    private Float x; //Значение поля должно быть больше -947, Поле не может быть null
    private Long y; //Поле не может быть null
}
public class Address {
    private String street; //Длина строки не должна быть больше 122, Поле не может быть null
}
public enum OrganizationType {
    COMMERCIAL,
    PUBLIC,
    TRUST,
    OPEN_JOINT_STOCK_COMPANY;
}
```

The diagram illustrates a Command-based architecture with the following components and relationships:

- Command Classes (Blue):**
  - Command:** Base interface for commands. Methods: `execute(String)` returns `ExecutionResponse`. Attributes: `commandName` (String), `description` (String), `equals(Object)` (boolean), `toString()` (String), `hashCode()` (int), `compareTo` (String) (String), `commandName` (String).
  - RemoveAllByAnnualTurnoverCommand:** Implements `Command`. Methods: `execute(String)` returns `ExecutionResponse`.
  - AddCommand:** Implements `Command`. Methods: `execute(String)` returns `ExecutionResponse`.
  - SaveCommand:** Implements `Command`. Methods: `execute(String)` returns `ExecutionResponse`.
  - MaxByPostalAddressCommand:** Implements `Command`. Methods: `execute(String)` returns `ExecutionResponse`.
  - UpdateCommand:** Implements `Command`. Methods: `execute(String)` returns `ExecutionResponse`.
  - InfoCommand:** Implements `Command`. Methods: `execute(String)` returns `ExecutionResponse`.
  - ExecuteScriptCommand:** Implements `Command`. Methods: `execute(String)` returns `ExecutionResponse`.
  - SumOfAnnualTurnoverCommand:** Implements `Command`. Methods: `execute(String)` returns `ExecutionResponse`.
  - AddIfMaxCommand:** Implements `Command`. Methods: `execute(String)` returns `ExecutionResponse`.
  - HelpCommand:** Implements `Command`. Methods: `execute(String)` returns `ExecutionResponse`.
  - ShowCommand:** Implements `Command`. Methods: `execute(String)` returns `ExecutionResponse`.
- Executable Classes (Green):**
  - Executable:** Interface for executable objects. Method: `execute(String)` returns `ExecutionResponse`.
  - HistoryCommand:** Implements `Executable`. Methods: `execute(String)` returns `ExecutionResponse`.
  - ExitCommand:** Implements `Executable`. Methods: `execute(String)` returns `ExecutionResponse`.
  - ClearCommand:** Implements `Executable`. Methods: `execute(String)` returns `ExecutionResponse`.
  - RemoveByIdCommand:** Implements `Executable`. Methods: `execute(String)` returns `ExecutionResponse`.
  - AddIfMinCommand:** Implements `Executable`. Methods: `execute(String)` returns `ExecutionResponse`.
- Other Classes (Grey):**
  - Organization:** Data class with attributes: `organization` (Long, String, Coordinates, ZonedDateTime, long), `postalAddress` (Address, long), `annualTurnover` (long), `creationDate` (ZonedDateTime), `coordinates` (Coordinates), `type` (OrganizationType), `id` (Long), `name` (String), `compareTo` (Model) (int), `hashCode()` (String), `validate()` (boolean), `toString()` (String), `name` (String), `coordinates` (Coordinates), `postalAddress` (Address, long), `id` (Long), `creationDate` (ZonedDateTime), `annualTurnover` (long), `type` (OrganizationType).
  - Coordinates:** Data class with attributes: `coordinates` (Float, Long), `x` (Float), `y` (Long), `validate()` (boolean), `toString()` (String), `x` (Float), `y` (Long).
  - Address:** Data class with attributes: `address` (String), `street` (String), `validate()` (boolean), `toString()` (String), `street` (String).
  - CommandName:** Data class with attributes: `commandName` (String), `valueOf(String)` (CommandName), `valueOf()` (CommandName).
  - CollectionManager:** Data class with attributes: `collectionManager` (FileManager), `organizationCollection` (LinkedHashSet<Organization>), `lastSaveTime` (ZonedDateTime), `lastSaveTime` (ZonedDateTime), `generateNewId()` (Long), `getObjectsById(Long)` (Organization), `removeAllByAnnualTurnover(Long)` (void), `addIfMin(Organization)` (boolean), `addIfMax(Organization)` (void), `addIfMin(Organization)` (void), `addIfMax(Organization)` (void), `clearCollection()` (void), `addIfMax(Organization)` (boolean), `addIfMin(Organization)` (void), `lastSaveTime` (String), `lastSaveTime` (ZonedDateTime), `organizationCollection` (LinkedHashSet<Organization>), `lastSaveTime` (ZonedDateTime).
  - OrganizationType:** Data class with attributes: `organizationType` (String), `organizationNameList` (String), `valueOf(String)` (OrganizationType), `valueOf()` (OrganizationType).
  - BasicFormation:** Data class with attributes: `form` (String), `form` (String).
  - CoordinatesForm:** Data class with attributes: `coordinatesForm` (Console, AppLogger), `ask(Console, AppLogger)` (Float), `form` (Coordinates), `ask(Console, AppLogger)` (Long).
  - AddressForm:** Data class with attributes: `addressForm` (Console, AppLogger), `form` (Address), `ask(Console, AppLogger)` (String).
  - ScriptManager:** Data class with attributes: `scriptManager` (Console), `checkRecurser` (String, Scanner), `executeScript(String, Runner)` (ExecutionResponse).
  - FileManager:** Data class with attributes: `fileManager` (Console, AppLogger), `convertCSVToCollection` (String) (LinkedHashSet<Organization>?), `readOrgCsvFile()` (String), `readCollectionFromCsv(Collection<Organization>)` (void), `loadFromFileFromEnvironment` (Variable()) (Set<String>), `writeCollectionToCsv(Collection<Organization>)` (void), `orgCsvFilePath` (String).
  - Console:** Data class with attributes: `console` (String), `prompt` (String), `printStream` (PrintStream), `prompt` (String), `console` (String), `print(Object)` (void), `hasNextInput()` (boolean), `fromArray(String[])` (AppLogger) (Organization?), `toArray(Organization)` (String), `useConsoleScanner()` (void), `useFileScanner(Scanner)` (void), `readFile()` (String), `prompt` (String), `console` (String), `printStream` (PrintStream).
  - AppLogger:** Data class with attributes: `appLogger` (Class<?>), `console` (String), `info(String)` (void), `error(String)` (void).
  - CommandManager:** Data class with attributes: `commandManager` (Console, CollectionManager), `commandHistory` (Queue<String>), `commands` (Map<String, Command>), `initCommands()` (Map<String, Command>), `addCommandToHistory()` (void), `commands` (Map<String, Command>), `commandHistory` (Queue<String>).
  - OrgCsvParser:** Data class with attributes: `orgCsvParser` (String), `fromArray(String[])` (AppLogger) (Organization?), `toArray(Organization)` (String), `useConsoleScanner()` (void), `useFileScanner(Scanner)` (void), `readFile()` (String), `prompt` (String), `console` (String), `printStream` (PrintStream).
  - Runner:** Data class with attributes: `runner` (Console, CommandManager, ScriptManager), `launchCommand(String)` (ExecutionResponse), `interactiveMode` (void).
  - ConsoleApp:** Data class with attributes: `consoleApp` (String), `main(String)` (void).
  - NotInLimitsException:** Data class with attributes: `notInLimitsException` (String), `notInLimitsException()` (String).
  - FormBreak:** Data class with attributes: `formBreak` (String), `formBreak()` (String).
  - Validatable:** Data class with attributes: `validatable` (boolean), `validate()` (boolean).
  - Model:** Data class with attributes: `model` (String), `annualTurnover` (long).

Можно посмотреть в репозитории:  
[https://github.com/LunarSonic/programming\\_lab5](https://github.com/LunarSonic/programming_lab5)

В процессе выполнения лабораторной работы я применила знания ООП и принципы SOLID и познакомилась с параметризованными типами. Кроме того, я изучила новые интерфейсы, такие как Map, Set, Queue, Deque, Iterable и Iterator. Впервые узнала об интерфейсах Comparator и Comparable, о потоках ввода-вывода и потоках-фильтрах и научилась работать с ними.