Министерство образования и науки Российской Федерации
Федеральное государственное автономное образовательное учреждение высшего образования

САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ, МЕХАНИКИ И ОПТИКИ

Факультет систем управления и робототехники

Отчет по лабораторной работе №5
по дисциплине «Программирование»
Вариант № 311789

Выполнил: студент гр. R3135
Хачатрян Георгий Робертович
Преподаватель:
Лаздин Артур Вячеславович

Санкт-Петербург, 2021 г.

Текст задачи

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.Stack`
- При запуске приложения коллекция должна автоматически заполняться значениями из файла.
- Имя файла должно передаваться программе с помощью: **аргумент командной строки**.
- 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.OutputStreamWriter`
- Все классы в программе должны быть задокументированы в формате javadoc.
- Программа должна корректно работать с неправильными данными (ошибки пользовательского ввода, отсутсвие 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)
- `shuffle` : перемешать элементы коллекции в случайном порядке
- `remove_greater {element}` : удалить из коллекции все элементы, превышающие заданный
- `remove_lower {element}` : удалить из коллекции все элементы, меньшие, чем заданный
- `filter_by_location location` : вывести элементы, значение поля location которых равно заданному
- `print_ascending` : вывести элементы коллекции в порядке возрастания
- `print_field_ascending_height` : вывести значения поля height всех элементов в порядке возрастания

Формат ввода команд:

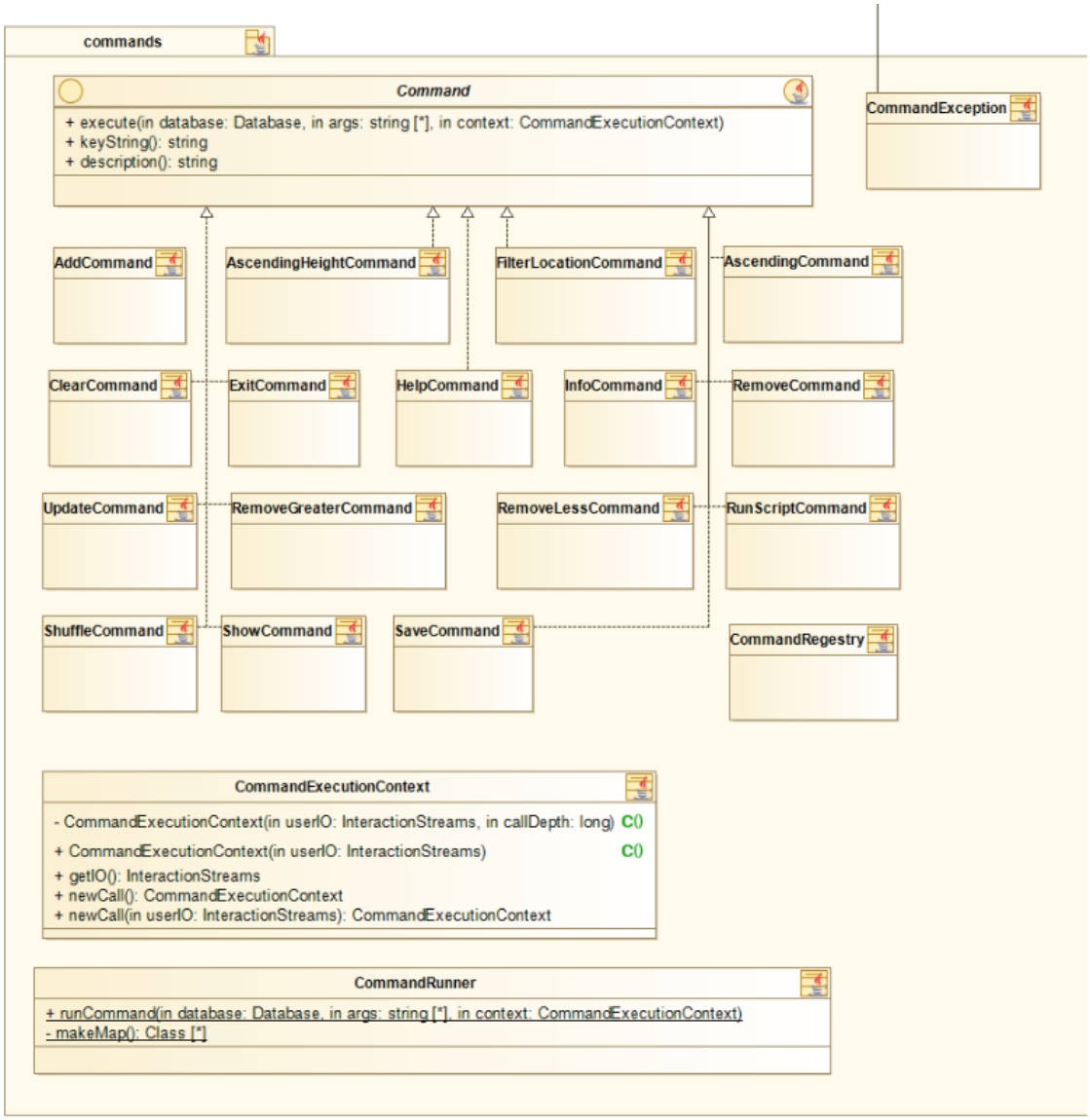
- Все аргументы команды, являющиеся стандартными типами данных (примитивные типы, классы-оболочки, String, классы для хранения дат), должны вводиться в той же строке, что и имя команды.
- Все составные типы данных (объекты классов, хранящиеся в коллекции) должны вводиться по одному полю в строку.
- При 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'е; введена строка вместо числа; введённое 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.

Исходный код

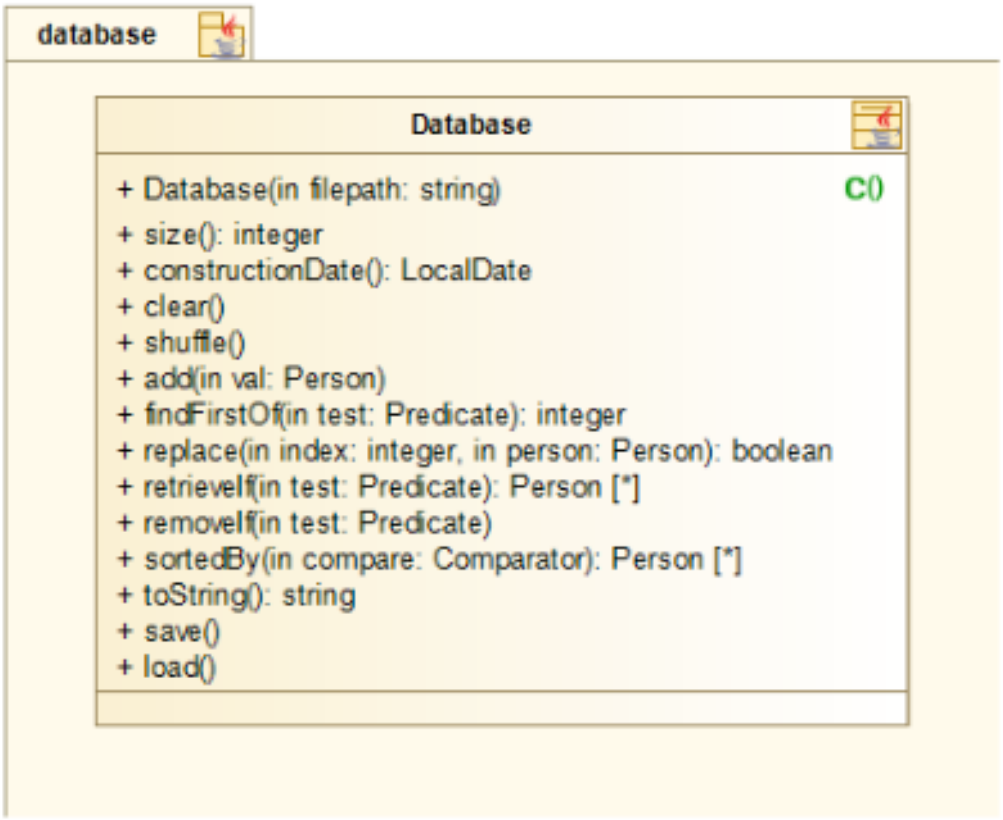
Структура проекта:

Весь исходный код проекта разбит на 3 пакета:

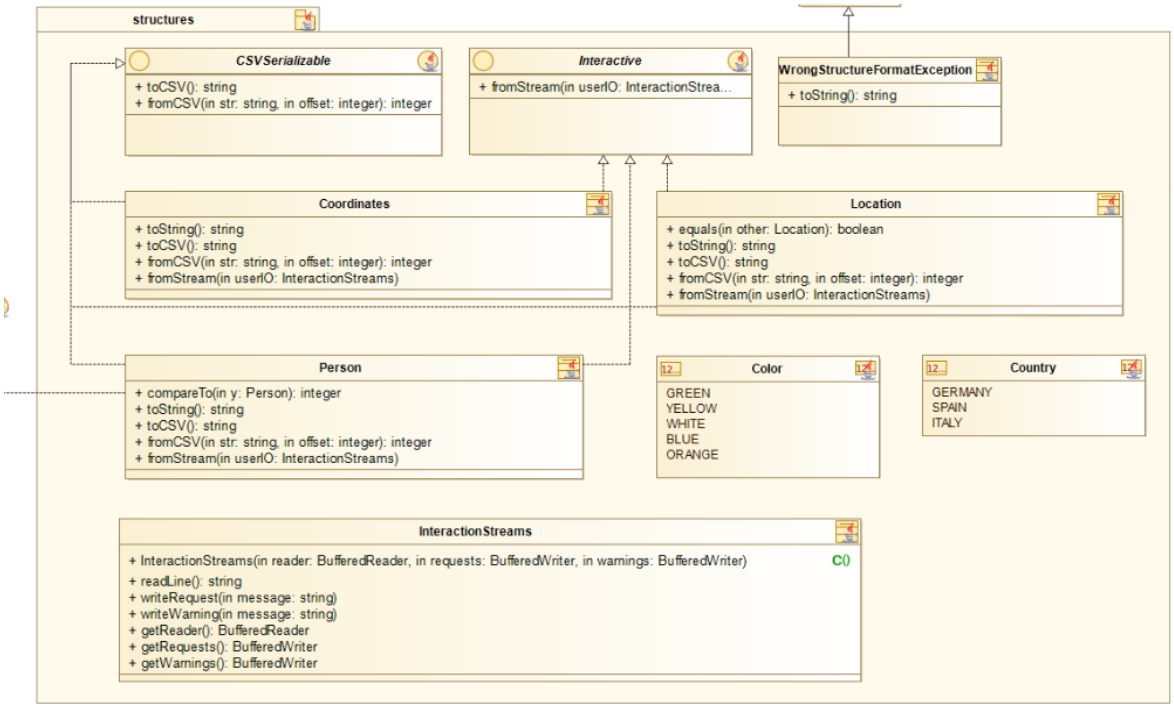
- `commands` – команды и все что надо для их выполнения



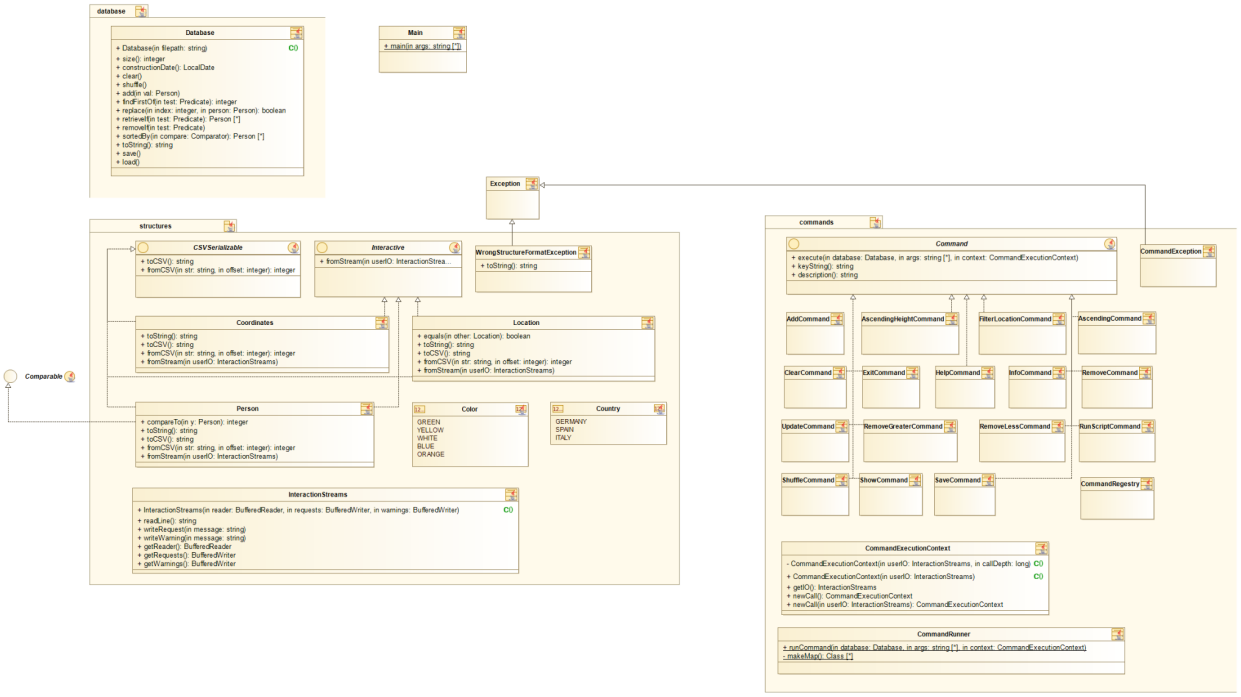
- `database` – база данных



- `structures` – структуры данных и вспомогательные классы для работы с ними



[Общая диаграмма классов]



Компиляция:
Команда для компиляции, выполняется в директории “`./src`”
```
javac -d ../build Main.java
```

Создание jar архива:
Команда для создания jar архива, выполняется в директории “`./build`”
```
jar cvmf ../Manifest.txt ./Lab4.jar .
```

Создание javadoc:
Команда для создания jar архива, выполняется в директории “`./src`”
```
javadoc -tag brief -tag warning -tag usage -d doc .\*
```

Запуск:
```
java -jar Lab4.jar ../assets/database.csv
```

Вывод:

1. Моделирование, UML в частности.
   При разработке приложений с использованием объектно-ориентированных языков часто прибегают к моделированию частей или всего приложения с помощью диаграмм. Предлогом для этого являются “Действенная и эффективная коммуникация” и “Полезная и стабильная абстракция”, а также другие менее существенные причины. Обратной стороной использования подобных практик является сложность или невозможность

смены курса разработки приложения при ошибке на первом этапе - моделировании. Ошибку же на этом этапе совершить очень легко так как разработка еще не началась, большинство нюансов и подводных камней не найдено.

2. Объектно-ориентированное программирование.
   Данная парадигма программирования предоставляет несколько базовых концепций:
   a. Инкапсуляция - объединение данных и методов работы с ними. Часто добавляется еще одно значение - сокрытие, т.е. разделение данных и методов на публичные, приватные и др.
   b. Наследование - создание подклассов, в которых сохраняется функциональность суперкласса.
   c. Полиморфизм - перегрузка и переопределение методов при создании или наследовании классов.

   Результатом использования этих концепций является код, в котором все данные являются независимыми, дискретными сущностями, находящимися в случайных местах памяти. Т.е. программа постоянно платит цену плохой локальности данных, постоянных промахов кэша и неверно предсказанных ветвлений. К тому же концепции, которые были на первый взгляд плюсами, в итоге мешают добавлению или изменению функциональности. Сокрытие не дает добраться до нужных, но приватных данных, наследование вытекает в избыточную зависимость подклассов от суперкласса.
3. Результаты повсеместного применения объектно-ориентированного подхода.
   Медленно и неверно работающий софт стал нормой. Никого не удивляет то, что текстовые редакторы открывают файлы в пару тысяч строк несколько секунд, IDE загружают свои XML проекты десятки секунд, а полная компиляция относительно небольшой программы может занять минуты. Говорить о программах от Adobe и MS не вижу смысла. Все это - результат того, что в индустрии никого не волнует как работает их продукт. Проектами руководят люди, которые видят только мнимые плюсы решения и не видят никаких минусов. Последние 20 лет все минусы “инновационных решений” компенсировались прогрессом в железе. Но может ли это продолжаться бесконечно?