

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, 2020 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, 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)
- `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'e; 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

```
import java.io.*;
import database.Database;
import java.lang.Exception;
import commands.CommandRunner;

public class Main {

    public static void main(String[] args) {
        if (args.length == 0) {
            System.out.println("The database path is not present.
Provide one in the first argument.");
```

```

        return;
    }

    Database database = new Database(args[0]);

    BufferedReader reader = new BufferedReader(new
InputStreamReader(System.in));
    BufferedWriter writer = new BufferedWriter(new
OutputStreamWriter(System.out));

    try {
        database.load();
    } catch (FileNotFoundException e) {
        System.out.println("Database file not found");
    } catch (Exception e) {
        System.out.println("Database file is corrupted.");
    }

    for (;;) {
        try {
            String str = reader.readLine();
            String[] substrs = str.split(" ");
            if (substrs.length == 1 &&
substrs[0].equals("exit")) break;

            CommandRunner.runCommand(substrs, database, reader,
writer);
        } catch (Exception e) {
            System.out.println(e.toString());
        }
    }

}

package commands;

import commands.Command;
import commands.CommandException;
import commands.CommandRegistry;

```

```

import database.Database;
import structures.Person;
import structures.WrongStructureFormatException;
import java.io.*;
import java.lang.Class;

public class AddCommand implements Command {
    @Override
    public void execute(Database database, BufferedReader reader,
        BufferedWriter writer, String[] args) throws CommandException, IOException
    {
        if (args.length != 0)
            CommandException.throwTooManyArgs(keyString(), args);

        try {
            Person person = Person.fromStream(reader, writer);
            database.add(person);
        } catch (IOException | WrongStructureFormatException e) {
            writer.write(e.toString());
            writer.flush();
        }
    }

    @Override
    public String keyString() { return "add"; }

    @Override
    public String description() { return "Add value to the database."; }
}

package commands;

import commands.Command;
import commands.CommandException;
import commands.CommandRegistry;
import database.Database;
import structures.Person;
import java.util.List;

```

```

import java.lang.Class;
import java.lang.Long;
import java.io.*;

public class AscendingCommand implements Command {
    @Override
    public void execute(Database database, BufferedReader reader,
        BufferedWriter writer, String[] args) throws CommandException, IOException
    {
        if (args.length != 0)
            CommandException.throwTooManyArgs(keyString(), args);

        List<Person> persons = database.sortedBy(
            (x, y) -> (x.location.z > y.location.z ? 1 : x.location.z
        < y.location.z ? -1 : 0)
        );

        for (Person person : persons)
            writer.write(person.toString());
        writer.flush();
    }

    @Override
    public String keyString() { return "print_ascending"; }

    @Override
    public String description() { return "Print values in the ascending
order."; }
}

package commands;

import commands.Command;
import commands.CommandException;
import commands.CommandRegistry;
import database.Database;
import structures.Person;
import java.util.List;
import java.lang.Class;

```

```

import java.lang.Long;
import java.io.*;

public class AscendingHeightCommand implements Command {
    @Override
    public void execute(Database database, BufferedReader reader,
        BufferedWriter writer, String[] args) throws CommandException, IOException
    {
        if (args.length != 0)
            CommandException.throwTooManyArgs(keyString(), args);

        List<Person> persons = database.sortedBy(
            (x, y) -> (x.height > y.height ? 1 : x.height < y.height
? -1 : 0)
        );

        for (Person person : persons)
            writer.write(person.toString());
        writer.flush();
    }

    @Override
    public String keyString() { return "print_field_ascending_height"; }

    @Override
    public String description() { return "Print values in the ascending
height order."; }
}

package commands;

import commands.Command;
import commands.CommandException;
import commands.CommandRegistry;
import database.Database;
import structures.Person;
import java.lang.Class;
import java.lang.Long;
import java.io.*;

```

```

public class ClearCommand implements Command {
    @Override
    public void execute(Database database, BufferedReader reader,
        BufferedWriter writer, String[] args) throws CommandException, IOException
    {
        if (args.length != 0)
            CommandException.throwTooManyArgs(keyString(), args);

        database.clear();
    }

    @Override
    public String keyString() { return "clear"; }

    @Override
    public String description() { return "Clear the database."; }
}

```

```

package commands;

```

```

import commands.CommandException;
import database.Database;
import java.io.*;

```

```

/**
 * @brief Interface for all of the commands.
 *      If you want to write one, add it to the CommandRegistry.
 */

```

```

public interface Command {

```

```

    /**
     * @brief Execute the command.
     *
     * @param database for command to change it.
     * @param reader for command to interact with the user.
     * @param writer for command to interact with the user.
     * @param args trailing strings that are given with the command.
     * @throws CommandException if the command fails.
    }
}

```

```

        * @throws IOException if the read or write streams fail.
        */
        void execute(Database database, BufferedReader reader,
BufferedWriter writer, String[] args)
            throws CommandException, IOException;

    /**
     * @return The string that is used to call the command.
     */
    String keyString();

    /**
     * @return Description of the command.
     */
    String description();
}

package commands;
import java.lang.Exception;

/**
 * @brief Thrown when command cannot be executed.
 */
public class CommandException extends Exception {
    public CommandException(String message) { this.message = message; }
    public String toString() { return message; }

    /**
     * @brief Throws CommandException with a formatted message.
     * @param cmd Command name.
     * @param args Extra arguments.
     * @throws CommandException with a formatted message.
     */
    public static void throwTooManyArgs(String cmd, String[] args)
throws CommandException {
        String message = "Command \"" + cmd + "\" does not expect any
arguments, but ";
        for (int i = 0; i < args.length; i++, message += i <
args.length ? ", " : " are given.")

```



```

        message += "\"" + args[i] + "\"";
        throw new CommandException(message);
    }

    private String message;
}

package commands;

import commands.*;
import database.Database;
import java.io.OutputStream;
import java.lang.Class;

/**
 * @brief CommandRegistry is a class that stores all of the available
 * command classes.
 *      If you want to write a new command, you can just add it here and
 * it will be
 *      added to the interactive user interface.
 */
public class CommandRegistry {

    /**
     * @brief Actual array of commands
     */
    public static final Class[] registry = {
        HelpCommand.class,
        InfoCommand.class,
        ShowCommand.class,
        AddCommand.class,
        UpdateCommand.class,
        RemoveCommand.class,
        ClearCommand.class,
        SaveCommand.class,
        RunScriptCommand.class,
        ExitCommand.class,
        ShuffleCommand.class,
        RemoveGreaterCommand.class,
        RemoveLessCommand.class,
        FilterLocationCommand.class,
    }

```

```

        AscendingCommand.class,
        AscendingHeightCommand.class,
    };
}

package commands;

import java.io.*;
import commands.CommandRegistry;
import commands.Command;
import database.Database;
import java.lang.Exception;
import java.lang.Error;
import java.io.IOException;
import java.io.OutputStreamWriter;
import java.util.HashMap;
import java.util.Map;
import java.util.Arrays;

/**
 * @brief Class that holds a list of all commands and can dispatch between
 * them.
 */
public class CommandRunner {

    /**
     * @brief Execute one of the regestred commands.
     * @param args trailing strings that are given with the command.
     * @param database for command to change it.
     * @param reader for command to interact with the user.
     * @param writer for command to interact with the user.
     */
    public static void runCommand(String[] args, Database database,
BufferedReader reader, BufferedWriter writer) throws CommandException,
IOException {
        Class commandClass = commandMap.get(args[0]);

        try {
            if (commandClass != null) {

```

```

        Command command =
(Command)commandClass.newInstance();
        command.execute(database, reader, writer,
Arrays.copyOfRange(args, 1, args.length));
    } else {
        writer.write("Unknown command. Use \"help\" to list
all commands.\n");
    }
    } catch (IllegalAccessException | InstantiationException e) {
        writer.write("Internal error: corrupted command
registry.\n");
    }

    writer.flush();
}

/**
 * @brief Populate the command map with the data from a command
registry.
 */
private static Map<String, Class> makeMap() {
    Map<String, Class> map = new HashMap<String, Class>();

    try {
        for (int i = 0; i < CommandRegistry.registry.length; i++)

map.put(((Command)CommandRegistry.registry[i].newInstance()).keyString(),
CommandRegistry.registry[i]);
    } catch (Exception e) {
        throw new Error("Internal error: corrupted command
registry.");
    }

    return map;
}

/**
 * @brief commandMap is used to find the command class corresponding
to the user input.
 */

```

```

        private static Map<String, Class> commandMap = makeMap();
    }

package commands;

import commands.Command;
import commands.CommandException;
import database.Database;
import java.io.*;

public class ExitCommand implements Command {
    public void execute(Database database, BufferedReader reader,
        BufferedWriter writer, String[] args) throws CommandException, IOException
    {
        if (args.length != 0)
            CommandException.throwTooManyArgs(keyString(), args);
        throw new CommandException("Internal error: exit cannot be
executed.");
    }

    public String keyString() { return "exit"; };
    public String description() { return "Exit without saving."; };
}

package commands;

import commands.Command;
import commands.CommandException;
import commands.CommandRegistry;
import database.Database;
import structures.Person;
import structures.WrongStructureFormatException;
import structures.Location;
import java.util.List;
import java.lang.Class;
import java.lang.Long;
import java.io.*;

public class FilterLocationCommand implements Command {
    @Override

```

```

        public void execute(Database database, BufferedReader reader,
BufferedWriter writer, String[] args) throws CommandException, IOException
{
    if (args.length != 0)
        CommandException.throwTooManyArgs(keyString(), args);

    try {
        Location location = Location.fromStream(reader, writer);
        List<Person> persons = database.retrieveIf(x ->
x.location.equals(location));

        for (Person person : persons)
            writer.write(person.toString());
        writer.flush();
    } catch (WrongStructureFormatException e) {
        writer.write(e.toString());
        writer.flush();
    }
}

@Override
public String keyString() { return "filter_by_location"; }

@Override
public String description() { return "Print all of the values with a
certain location."; }

}

```

```

package commands;

```

```

import commands.Command;
import commands.CommandException;
import commands.CommandRegistry;
import database.Database;
import java.io.*;
import java.lang.Class;
import java.lang.Exception;

```

```

public class HelpCommand implements Command {

```

```

        @Override
        public void execute(Database database, BufferedReader reader,
BufferedWriter writer, String[] args) throws CommandException, IOException
{
            if (args.length != 0)
                CommandException.throwTooManyArgs(keyString(), args);

            try {
                for (int i = 0; i < CommandRegistry.registry.length; i++)
{
                    Command command =
(Command)(CommandRegistry.registry[i].newInstance());
                    writer.write(String.format("%d: %s: %s\n", i,
command.keyString(), command.description()));
                }
            } catch (Exception e) {
                writer.write(String.format("Internal error: corrupted
command registry."));
            }

            writer.flush();
        }

        @Override
        public String keyString() { return "help"; }

        @Override
        public String description() { return "Print help for all commands.";
}

}

package commands;

import commands.Command;
import commands.CommandException;
import commands.CommandRegistry;
import database.Database;
import java.io.*;
import java.lang.Class;

```

```

public class InfoCommand implements Command {
    @Override
    public void execute(Database database, BufferedReader reader,
        BufferedWriter writer, String[] args) throws CommandException, IOException
    {
        if (args.length != 0)
            CommandException.throwTooManyArgs(keyString(), args);

        writer.write(String.format("Type: Stack<Person>, Size: %d,
Date: ", database.size()) + database.constructionDate().toString() +
"\n");
        writer.flush();
    }

    @Override
    public String keyString() { return "info"; }

    @Override
    public String description() { return "Information about the
database."; }
}

```

```

package commands;

```

```

import commands.Command;
import commands.CommandException;
import commands.CommandRegistry;
import database.Database;
import structures.Person;
import java.lang.Class;
import java.lang.Long;
import java.io.*;

```

```

public class RemoveCommand implements Command {
    @Override
    public void execute(Database database, BufferedReader reader,
        BufferedWriter writer, String[] args) throws CommandException, IOException
    {

```

```

        if (args.length != 1) {
            String message = "Command \"remove_by_id\" expects one
argument - id, but ";
            if (args.length == 0) {
                message += "none are given.";
            } else {
                for (int i = 0; i < args.length; i++, message += i
< args.length ? ", " : " are given.")
                    message += "\"" + args[i] + "\"";
            }

            throw new CommandException(message);
        }

        final long id;
        try {
            id = Long.parseLong(args[0]);
        } catch (NumberFormatException e) {
            writer.write("Unable to parse the id.\n");
            writer.flush();
            return;
        }

        database.removeIf(x -> x.id == id);
    }

    @Override
    public String keyString() { return "remove_by_id"; }

    @Override
    public String description() { return "Remove the value from the
database."; }

}

package commands;

import commands.Command;
import commands.CommandException;
import commands.CommandRegistry;

```



```

import database.Database;
import structures.Person;
import java.util.List;
import java.lang.Class;
import java.lang.Integer;
import java.io.*;

public class RemoveGreaterCommand implements Command {
    @Override
    public void execute(Database database, BufferedReader reader,
        BufferedWriter writer, String[] args) throws CommandException, IOException
    {
        if (args.length != 1) {
            String message = "Command \"remove_greater\" expects one
argument - location.z, but ";
            if (args.length == 0) {
                message += "none are given.";
            } else {
                for (int i = 0; i < args.length; i++, message += i
< args.length ? ", " : " are given.")
                    message += "\"" + args[i] + "\"";
            }

            throw new CommandException(message);
        }

        final int z;
        try {
            z = Integer.parseInt(args[0]);
        } catch (NumberFormatException e) {
            writer.write("Unable to parse the location.z.\n");
            writer.flush();
            return;
        }

        database.removeIf(x -> x.location.z > z);
    }

    @Override
    public String keyString() { return "remove_greater"; }
}

```

```

        @Override
        public String description() { return "Remove values that have a
location.z greater than given."; }

    }

package commands;

import commands.Command;
import commands.CommandException;
import commands.CommandRegistry;
import database.Database;
import structures.Person;
import java.util.List;
import java.lang.Class;
import java.lang.Long;
import java.io.*;

public class RemoveLessCommand implements Command {
    @Override
    public void execute(Database database, BufferedReader reader,
BufferedWriter writer, String[] args) throws CommandException, IOException
    {
        if (args.length != 1) {
            String message = "Command \"remove_less\" expects one
argument - location.z, but ";
            if (args.length == 0) {
                message += "none are given.";
            } else {
                for (int i = 0; i < args.length; i++, message += i
< args.length ? ", " : " are given.")
                    message += "\"" + args[i] + "\"";
            }

            throw new CommandException(message);
        }

        final int z;
        try {

```

```

        z = Integer.parseInt(args[0]);
    } catch (NumberFormatException e) {
        writer.write("Unable to parse the location.z.\n");
        writer.flush();
        return;
    }

    database.removeIf(x -> x.location.z < z);
}

@Override
public String keyString() { return "remove_less"; }

@Override
public String description() { return "Remove values that have a
location.z less than given."; }

}

package commands;

import commands.Command;
import commands.CommandException;
import commands.CommandRegistry;
import database.Database;
import structures.Person;
import java.io.*;
import java.lang.Class;

class NullOutputStream extends OutputStream {
    @Override
    public void write(int a) throws IOException {}
}

public class RunScriptCommand implements Command {
    @Override
    public void execute(Database database, BufferedReader reader,
        BufferedWriter writer, String[] args) throws CommandException, IOException
    {
        if (args.length == 0)

```

```
        throw new CommandException("Command \"execute_script\"  
expects one argument - script_file_path, but none are given.");
```

```
String filepath = "";  
for (String arg : args)  
    filepath += arg;
```

```
File file = new File(filepath);  
FileInputStream fileIStream;
```

```
try {  
    fileIStream = new FileInputStream(file);  
} catch (FileNotFoundException e) {  
    writer.write("File not found\n");  
    writer.flush();  
    return;  
}
```

```
BufferedReader fileReader = new BufferedReader(new  
InputStreamReader(fileIStream));
```

```
BufferedWriter fileWriter = new BufferedWriter(new  
OutputStreamWriter(new NullOutputStream()));
```

```
for (;;) {  
    try {  
        String str = fileReader.readLine();  
        if (str == null) break;  
        String[] substrs = str.split(" ");  
        if (substrs.length == 1 &&  
substrs[0].equals("exit")) break;
```

```
        CommandRunner.runCommand(substrs, database,  
fileReader, fileWriter);
```

```
    } catch (IOException e) {  
        writer.write(e.toString());  
        break;  
    }
```

```
}
```

```
writer.flush();
```

```

        fileInputStream.close();
    }

    @Override
    public String keyString() { return "execute_script"; }

    @Override
    public String description() { return "Execute script from a file."; }
}

}

package commands;

import commands.Command;
import commands.CommandException;
import commands.CommandRegistry;
import database.Database;
import java.io.*;
import java.lang.Class;

public class SaveCommand implements Command {
    @Override
    public void execute(Database database, BufferedReader reader,
        BufferedWriter writer, String[] args) throws CommandException, IOException
    {
        if (args.length != 0)
            CommandException.throwTooManyArgs(keyString(), args);

        try {
            database.save();
        } catch (FileNotFoundException e) {
            writer.write("Database file not found\n");
            writer.flush();
        }
    }

    @Override
    public String keyString() { return "save"; }
}

```

```

        @Override
        public String description() { return "Save the database as a CSV
file."; }

    }

package commands;

import commands.Command;
import commands.CommandException;
import commands.CommandRegistry;
import database.Database;
import structures.Person;
import java.io.*;
import java.lang.Class;

public class ShowCommand implements Command {
    @Override
    public void execute(Database database, BufferedReader reader,
BufferedWriter writer, String[] args) throws CommandException, IOException
    {
        if (args.length != 0)
            CommandException.throwTooManyArgs(keyString(), args);

        writer.write(database.toString());
        writer.flush();
    }

    @Override
    public String keyString() { return "show"; }

    @Override
    public String description() { return "Print all the values of the
database."; }

}

package commands;

import commands.Command;

```

```

import commands.CommandException;
import commands.CommandRegistry;
import database.Database;
import structures.Person;
import java.io.*;
import java.lang.Class;

public class ShuffleCommand implements Command {
    @Override
    public void execute(Database database, BufferedReader reader,
        BufferedWriter writer, String[] args) throws CommandException, IOException
    {
        if (args.length != 0)
            CommandException.throwTooManyArgs(keyString(), args);
        database.shuffle();
    }

    @Override
    public String keyString() { return "shuffle"; }

    @Override
    public String description() { return "Shuffle the values in the
        database."; }
}

package commands;

import commands.Command;
import commands.CommandException;
import commands.CommandRegistry;
import database.Database;
import structures.Person;
import structures.WrongStructureFormatException;
import java.lang.Class;
import java.lang.Long;
import java.io.*;

public class UpdateCommand implements Command {
    @Override

```

```

        public void execute(Database database, BufferedReader reader,
        BufferedWriter writer, String[] args) throws CommandException, IOException
        {
            if (args.length != 1) {
                String message = "Command \"update\" expects one argument
- id, but ";

                if (args.length == 0) {
                    message += "none are given.";
                } else {
                    for (int i = 0; i < args.length; i++, message += i
< args.length ? ", " : " are given.")
                        message += "\"" + args[i] + "\"";
                }

                throw new CommandException(message);
            }

            final long id;
            try {
                id = Long.parseLong(args[0]);
            } catch (NumberFormatException e) {
                writer.write("Unable to parse the id.\n");
                writer.flush();
                return;
            }

            int index = database.findFirstOf(x -> x.id == id);
            if (index == -1) {
                writer.write("Element not found.\n");
                writer.flush();
                return;
            }

            try {
                Person person = Person.fromStream(reader, writer);
                person.id = id;
                database.replace(index, person);
            } catch (WrongStructureFormatException e) {
                writer.write(e.toString());
                writer.flush();
            }
        }
    }
}

```



```

        }
    }

    @Override
    public String keyString() { return "update"; }

    @Override
    public String description() { return "Update the value in the
database."; }

}

package database;

import java.util.Stack;
import java.util.List;
import java.util.ArrayList;
import java.util.Collections;
import java.util.function.Predicate;
import java.util.Comparator;
import java.time.LocalDate;
import java.io.*;
import structures.Person;

/**
 * @brief This class holds datastructure and the metadata for the database.
 */
public class Database {
    /**
     * @brief Constructs database with associated database file.
     * @param filepath Path to a database file.
     */
    public Database(String filepath) { this.filepath = filepath; stack =
new Stack(); date = LocalDate.now(); }

    /**
     * @return Size of the underlying datastructure.
     */
    public int size() { return stack.size(); }

```

```

/**
 * @return Date when underlying datastructure was constructed.
 */
public LocalDate constructionDate() { return date; }

/**
 * @brief Clear the underlying datastructure.
 */
public void clear() { stack.clear(); }

/**
 * @brief Shuffle the elements in the underlying datastructure.
 */
public void shuffle() { Collections.shuffle(stack); }

/**
 * @brief Add new element to the database.
 * @param val Element to add.
 */
public void add(Person val) {
    long maxId = 0;
    for (Person person : stack)
        if (person.id > maxId) maxId = person.id;
    val.id = maxId + 1;
    stack.push(val);
}

/**
 * @brief Find the first element that passes the test.
 * @param test Test function.
 * @return Either index of the element or -1.
 */
public int findFirstOf(Predicate<Person> test) {
    for (int i = 0; i < stack.size(); i++)
        if (test.test(stack.get(i))) return i;
    return -1;
}

/**
 * @brief Replace element at index with a given.

```

```

* @param index Index at which new element should be placed.
* @param person New element to insert.
* @return Success bool.
*/
public boolean replace(int index, Person person) {
    if (index >= stack.size()) return false;
    stack.set(index, person);
    return true;
}

/**
* @brief Retrieve all of the elements that are passing the test.
* @param test Test function.
* @return List of passed elements.
*/
public List<Person> retrieveIf(Predicate<Person> test) {
    List<Person> ret = new ArrayList();
    for (Person person : stack)
        if (test.test(person)) ret.add(person);
    return ret;
}

/**
* @brief Remove all of the elements that pass the test.
* @param test Test function.
*/
public void removeIf(Predicate<Person> test) {
    for (int i = 0; i < stack.size(); i++) {
        if (test.test(stack.get(i))) {
            stack.removeElementAt(i);
        } else {
            i++;
        }
    }
}

/**
* @brief Retrieve all of the elements in the sorted order.
* @param compare Comparison function
*/

```

```

public List<Person> sortBy(Comparator<Person> compare) {
    List<Person> ret = (Stack<Person>)stack.clone();
    Collections.sort(ret, compare);
    return ret;
}

@Override
public String toString() {
    String ret = "";
    for (Person person : stack)
        ret += person.toString();
    return ret;
}

/**
 * @brief Save database into the associated file
 */
public void save() throws IOException, FileNotFoundException {
    File file = new File(filepath);
    FileOutputStream fileOutputStream;

    fileOutputStream = new FileOutputStream(file);
    OutputStreamWriter fileWriter = new
OutputStreamWriter(fileOutputStream);

    fileWriter.write(date.toString() + "\n");

    for (Person person : stack)
        fileWriter.write(person.toCSV() + "\n");

    fileWriter.flush();
    fileOutputStream.close();
}

/**
 * @brief Load database from the associated file
 */
public void load() throws IOException, FileNotFoundException {
    File file = new File(filepath);
    FileInputStream fileInputStream;

```

```

        fileIStream = new FileInputStream(file);
        BufferedReader fileReader = new BufferedReader(new
InputStreamReader(fileIStream));

```

```

        String str = fileReader.readLine();
        if (str != null) {
            date = LocalDate.parse(str);

```

```

            for (;;) {
                str = fileReader.readLine();
                if (str == null) break;
                Person person = new Person();
                person.fromCSV(str, 0);
                stack.push(person);

```

```

            }
        }

```

```

        fileIStream.close();

```

```

    }

```

```

    private Stack<Person> stack;
    private LocalDate date;
    private String filepath;

```

```

}

```

```

package structures;

```

```

public enum Color {
    GREEN,
    YELLOW,
    WHITE,
    BLUE,
    ORANGE;
}

```

```

package structures;

```

```

import structures.WrongStructureFormatException;
import structures.CSV;

```

```

import structures.CSVSerializable;
import java.lang.Integer;
import java.lang.Float;
import java.lang.NumberFormatException;
import java.io.*;

public class Coordinates implements CSVSerializable {
    /* private */ public int x;
    /* private */ public float y;

    @Override
    public String toString() {
        String ret = "Coordinates {";

        ret += "\n    " + (new Integer(x)).toString();
        ret += "\n    " + (new Float(y)).toString();

        return ret + "\n}\n";
    }

    @Override
    public String toCSV() {
        String ret = "";

        ret += CSV.decorate((new Integer(x)).toString());
        ret += "," + CSV.decorate((new Float(y)).toString());

        return ret;
    }

    @Override
    public int fromCSV(String str, int offset) {
        String val;

        val = CSV.nextCell(str, offset);
        offset += val.length();
        x = Integer.parseInt(CSV.undecorate(val));

        val = CSV.nextCell(str, offset);
        offset += val.length();
    }
}

```

```

        y = Float.parseFloat(CSV.undecorate(val));

        return offset;
    }

    /**
     * @brief Make coordinates from stream in an interactive manner.
     *
     * @param reader Input stream from the user.
     * @param writer Output stream to the user.
     * @return Constructed coordinates.
     * @throws IOException if either of the streams fails.
     * @throws WrongStructureFormatException if the user fails to input the
coordinates.
    */
    public static Coordinates fromStream(BufferedReader reader,
BufferedWriter writer) throws IOException,
        WrongStructureFormatException {
        Coordinates ret = new Coordinates();

        writer.write("Coordinates:\n");
        writer.flush();

        for (;;) {
            writer.write("X: ");
            writer.flush();

            String str = reader.readLine();
            if (str == null) throw new WrongStructureFormatException();

            try {
                ret.x = Integer.parseInt(str);
                break;
            } catch (NumberFormatException e) {
                writer.write("Unable to parse the value.\n");
                writer.flush();
            }
        }
    }
}

```

```

        for (;;) {
            writer.write("Y: ");
            writer.flush();

            String str = reader.readLine();
            if (str == null) throw new WrongStructureFormatException();

            try {
                float y = Float.parseFloat(str);
                if (y <= 971) {
                    ret.y = y;
                    break;
                } else {
                    writer.write("Value is out of range [-inf,
971].\n");
                    writer.flush();
                }
            } catch (NumberFormatException e) {
                writer.write("Unable to parse the value.\n");
                writer.flush();
            }

        }

        return ret;
    }
}

```

```
package structures;
```

```

public enum Country {
    GERMANY,
    SPAIN,
    ITALY;
}

```

```
package structures;
```

```
import java.io.*;
```



```

/**
 * @brief Just some functions used to help with CSV parsing
 */
public class CSV {

    /**
     * @brief Formats the string so it can be used as a CSV cell.
     *      String is quoted if it has quotes or commas.
     *      All of the quotes are duplicated ie " -> "".
     *
     * @warning This function doesn't add commas, so do that yourself.
     *
     * @param val String that you want to store as CSV
     * @return String with all of the formatting needed
     */
    public static String decorate(String val) {
        boolean hasQuotes = val.indexOf("\"") != -1;
        boolean hasCommas = val.indexOf(",") != -1;

        if (hasQuotes) {
            String str = "";

            for (int i = 0; i < val.length(); i++) {
                char c = val.charAt(i);
                if (c == '\"') str += "\\\"";
                else str += c;
            }

            val = str;
        }

        if (hasQuotes || hasCommas)
            val = "\"" + val + "\"";

        return val;
    }

    /**
     * @brief String all of the CSV specific stuff. See the decorate

```

```

*          function to learn more.
*
* @warning This function assumes that the comma either not present
or leading.
*
* @param val String that represents CSV cell with or without leading
comma.
* @return String that is stored in the CSV cell.
*/
public static String undecorate(String val) {
    int firstChar = val.charAt(0) == ',' ? 1 : 0;
    boolean quoted = val.charAt(firstChar) == '\"';
    if (!quoted) return val.substring(firstChar, val.length());

    String str = "";

    for (int i = firstChar + (quoted ? 1 : 0); i + 1 <
val.length();) {
        char c = val.charAt(i);
        if (c == '\"') { str += "\""; i += 2; }
        else { str += c; i += 1; }
    }

    return str;
}

/**
* @brief Find the string that represents a single cell in a
*          multi-cell input string with a known offset from the start.
*
* @param val String to parse.
* @param offset Offset into the val
* @return String that represents a single cell
*
* @usage Add length of the returned string to the offset to parse
the next cell
*/
public static String nextCell(String val, int offset) {
    int firstChar = val.charAt(offset) == ',' ? 1 : 0;
    boolean quoted = val.charAt(firstChar + offset) == '\"';

```

```

        int end = offset + (quoted ? 1 : 0) + firstChar;
        for (int i = end; i < val.length();) {
            boolean thisIsQuote = val.charAt(i) == '\"';
            boolean nextIsEnd    = i + 1 == val.length();

            // "Random quoted string",
            //           We are here ^
            if (quoted && thisIsQuote && (nextIsEnd || val.charAt(i +
1) == ',')) {
                end = i + 1;
                break;
            }

            // Random non-quoted string,
            //           We are here ^
            if (!quoted && (nextIsEnd || val.charAt(i + 1) == ',')) {
                end = i + 1;
                break;
            }

            // "Random quoted", string",
            //   We are here ^
            if (quoted && thisIsQuote && val.charAt(i + 1) == '\"')
                i += 2;
            else
                i += 1;
        }

        return val.substring(offset, end);
    }
}

```

```
package structures;
```

```
public interface CSVSerializable {
```

```

    /**
     * @brief Convert this object to CSV string

```

```

    */
    public String toCSV();

    /**
    * @brief Load this object from the CSV string at a certain offset.
    *
    * @param str CSV string representing this object.
    * @param offset Offset into the string.
    * @return New offset into the string.
    */
    public int fromCSV(String str, int offset);
}

```

```
package structures;
```

```

import structures.WrongStructureFormatException;
import structures.CSV;
import structures.CSVSerializable;
import java.lang.Integer;
import java.lang.Long;
import java.lang.Double;
import java.lang.NumberFormatException;
import java.io.*;

```

```

public class Location implements CSVSerializable {
    /* private */ public Long x;
    /* private */ public double y;
    /* private */ public int z;
    /* private */ public String name;

    public boolean equals(Location other) {
        return x.equals(other.x) && y == other.y && z == other.z &&
name.equals(other.name);
    }

    @Override
    public String toString() {
        String ret = "Location {";

```

```

        ret += "\n    " + x.toString();
        ret += "\n    " + (new Double(y)).toString();
        ret += "\n    " + (new Integer(z)).toString();
        ret += "\n    " + (name == null ? "null" : name);

    return ret + "\n}\n";
}

@Override
public String toCSV() {
    String ret = "";

    ret += CSV.decorate(x.toString());
    ret += "," + CSV.decorate((new Double(y)).toString());
    ret += "," + CSV.decorate((new Integer(z)).toString());
    ret += "," + CSV.decorate(name == null ? "null" : name);

    return ret;
}

@Override
public int fromCSV(String str, int offset) {
    String val;

    val = CSV.nextCell(str, offset);
    offset += val.length();
    x = Long.parseLong(CSV.undecorate(val));

    val = CSV.nextCell(str, offset);
    offset += val.length();
    y = Double.parseDouble(CSV.undecorate(val));

    val = CSV.nextCell(str, offset);
    offset += val.length();
    z = Integer.parseInt(CSV.undecorate(val));

    val = CSV.nextCell(str, offset);
    offset += val.length();
    val = CSV.undecorate(val);

```

```

        if (val.equals("null"))
            name = null;
        else
            name = val;

        return offset;
    }

    /**
     * @brief Make location from stream in an interactive manner
     *
     * @param reader Input stream from the user
     * @param writer Output stream to the user
     * @return Constructed location
     * @throws IOException if either of the streams fails
     * @throws WrongStructureFormatException if the user fails to input a
location
    */
    public static Location fromStream(BufferedReader reader,
BufferedWriter writer) throws IOException,
        WrongStructureFormatException {
        Location ret = new Location();

        writer.write("Location:\n");
        writer.flush();

        for (;;) {
            writer.write("X: ");
            writer.flush();

            String str = reader.readLine();
            if (str == null) throw new WrongStructureFormatException();

            try {
                ret.x = new Long(Long.parseLong(str));
                break;
            } catch (NumberFormatException e) {
                writer.write("Unable to parse the value.\n");
                writer.flush();
            }
        }
    }

```

```

    }

    for (;;) {
        writer.write("Y: ");
        writer.flush();

        String str = reader.readLine();
        if (str == null) throw new WrongStructureFormatException();

        try {
            ret.y = Double.parseDouble(str);
            break;
        } catch (NumberFormatException e) {
            writer.write("Unable to parse the value.\n");
            writer.flush();
        }
    }

    for (;;) {
        writer.write("Z: ");
        writer.flush();

        String str = reader.readLine();
        if (str == null) throw new WrongStructureFormatException();

        try {
            ret.z = Integer.parseInt(str);
            break;
        } catch (NumberFormatException e) {
            writer.write("Unable to parse the value.\n");
            writer.flush();
        }
    }

    {
        writer.write("Name: ");
        writer.flush();
    }

```

```

        String name = reader.readLine();
        if (name.length() > 0)
            ret.name = name;
        else
            ret.name = null;
    }

    return ret;
}

}

package structures;

import structures.WrongStructureFormatException;
import structures.Coordinates;
import structures.Location;
import structures.Color;
import structures.CSV;
import structures.CSVSerializable;
import java.time.LocalDate;
import java.lang.Long;
import java.lang.Comparable;
import java.io.*;

public class Person implements Comparable<Person>, CSVSerializable {
    // I'm not ready to spend my time here and in the other structures
    // on the bs getters and setters for absolutly no reason.

    /* private */ public long id;
    /* private */ public String name;
    /* private */ public Coordinates coordinates;
    /* private */ public LocalDate creationDate;
    /* private */ public long height;
    /* private */ public Color eyeColor;
    /* private */ public Color hairColor;
    /* private */ public Country nationality;
    /* private */ public Location location;

    @Override

```



```

    public int compareTo(Person y) {
        return location.z > y.location.z ? 1 : location.z < y.location.z ?
-1 : 0;
    }

```

```

@Override
public String toString() {
    String ret = "Person {";

    ret += "\n    " + (new Long(id)).toString();
    ret += "\n    " + name.toString();

    String[] lines = coordinates.toString().split("\n");
    for (String line : lines)
        ret += "\n    " + line;

    ret += "\n    " + creationDate.toString();
    ret += "\n    " + (new Long(height)).toString();
    ret += "\n    " + eyeColor.toString();
    ret += "\n    " + (hairColor == null ? "null" :
hairColor.toString());
    ret += "\n    " + nationality.toString();

    lines = location.toString().split("\n");
    for (String line : lines)
        ret += "\n    " + line;

    return ret + "\n}\n";
}

```

```

@Override
public String toCSV() {
    String ret = "";

    ret += CSV.decorate((new Long(id)).toString());
    ret += "," + CSV.decorate(name.toString());
    ret += "," + coordinates.toCSV();
    ret += "," + CSV.decorate(creationDate.toString());
    ret += "," + CSV.decorate((new Long(height)).toString());
    ret += "," + CSV.decorate(eyeColor.toString());

```

```

        ret += "," + CSV.decorate((hairColor == null ? "null" :
hairColor.toString()));
        ret += "," + CSV.decorate(nationality.toString());
        ret += "," + location.toCSV();

        return ret;
    }

```

```

@Override
public int fromCSV(String str, int offset) {
    String val;

    val = CSV.nextCell(str, offset);
    offset += val.length();
    id = Long.parseLong(CSV.undecorate(val));

    val = CSV.nextCell(str, offset);
    offset += val.length();
    name = CSV.undecorate(val);

    coordinates = new Coordinates();
    offset = coordinates.fromCSV(str, offset);

    val = CSV.nextCell(str, offset);
    offset += val.length();
    creationDate = LocalDate.parse(CSV.undecorate(val));

    val = CSV.nextCell(str, offset);
    offset += val.length();
    height = Long.parseLong(CSV.undecorate(val));

    val = CSV.nextCell(str, offset);
    offset += val.length();
    eyeColor = Color.valueOf(CSV.undecorate(val));

    val = CSV.nextCell(str, offset);
    offset += val.length();
    val = CSV.undecorate(val);
    if (val.equals("null"))
        hairColor = null;
}

```

```

        else
            hairColor = Color.valueOf(val);

        val = CSV.nextCell(str, offset);
        offset += val.length();
        nationality = Country.valueOf(CSV.undecorate(val));

        location = new Location();
        offset = location.fromCSV(str, offset);

        return offset;
    }

    /**
     * @brief Make person from stream in an interactive manner.
     *
     * @param reader Input stream from the user.
     * @param writer Output stream to the user.
     * @return Constructed person.
     * @throws IOException if either of the streams fails.
     * @throws WrongStructureFormatException if the user fails to input a
     person.
     */
    public static Person fromStream(BufferedReader reader, BufferedWriter
writer) throws IOException,
        WrongStructureFormatException {
        Person ret = new Person();
        ret.creationDate = LocalDate.now();

        writer.write("Person:\n");
        writer.flush();

        for (;;) {
            writer.write("Name: ");
            writer.flush();

            String str = reader.readLine();
            if (str == null) throw new WrongStructureFormatException();

            if (str.length() > 0) {

```

```

        ret.name = str;
        break;
    } else {
        writer.write("Illigal name string.\n");
        writer.flush();
    }
}

ret.coordinates = Coordinates.fromStream(reader, writer);

for (;;) {
    writer.write("Height: ");
    writer.flush();

    String str = reader.readLine();
    if (str == null) throw new WrongStructureFormatException();

    try {
        long height = Long.parseLong(str);
        if (height > 0) {
            ret.height = height;
            break;
        } else {
            writer.write("Value is out of range [1,
9223372036854775807].\n");
            writer.flush();
        }
    } catch (NumberFormatException e) {
        writer.write("Unable to parse the value.\n");
        writer.flush();
    }
}

for (;;) {
    writer.write("Eye color:\n");
    for (Color value : Color.values()) {
        writer.write(value.toString());
        writer.write("\n");
    }
    writer.flush();
}

```

```

String str = reader.readLine();
if (str == null) throw new WrongStructureFormatException();

try {
    ret.eyeColor = Color.valueOf(str);
    break;
} catch (IllegalArgumentException e) {
    writer.write("Unable to parse the value.\n");
    writer.flush();
}
}

for (;;) {
    writer.write("Hair color:\n");
    for (Color value : Color.values()) {
        writer.write(value.toString());
        writer.write("\n");
    }
    writer.flush();

    String str = reader.readLine();
    if (str == null) throw new WrongStructureFormatException();

    if (str.length() == 0) {
        ret.hairColor = null;
        break;
    }

    try {
        ret.hairColor = Color.valueOf(str);
        break;
    } catch (IllegalArgumentException e) {
        writer.write("Unable to parse the value.\n");
        writer.flush();
    }
}

for (;;) {
    writer.write("Nationality:\n");

```

```

        for (Country value : Country.values()) {
            writer.write(value.toString());
            writer.write("\n");
        }
        writer.flush();

        String str = reader.readLine();
        if (str == null) throw new WrongStructureFormatException();

        try {
            ret.nationality = Country.valueOf(str);
            break;
        } catch (IllegalArgumentException e) {
            writer.write("Unable to parse the value.\n");
            writer.flush();
        }
    }

    ret.location = Location.fromStream(reader, writer);

    return ret;
}

package structures;
import java.lang.Exception;

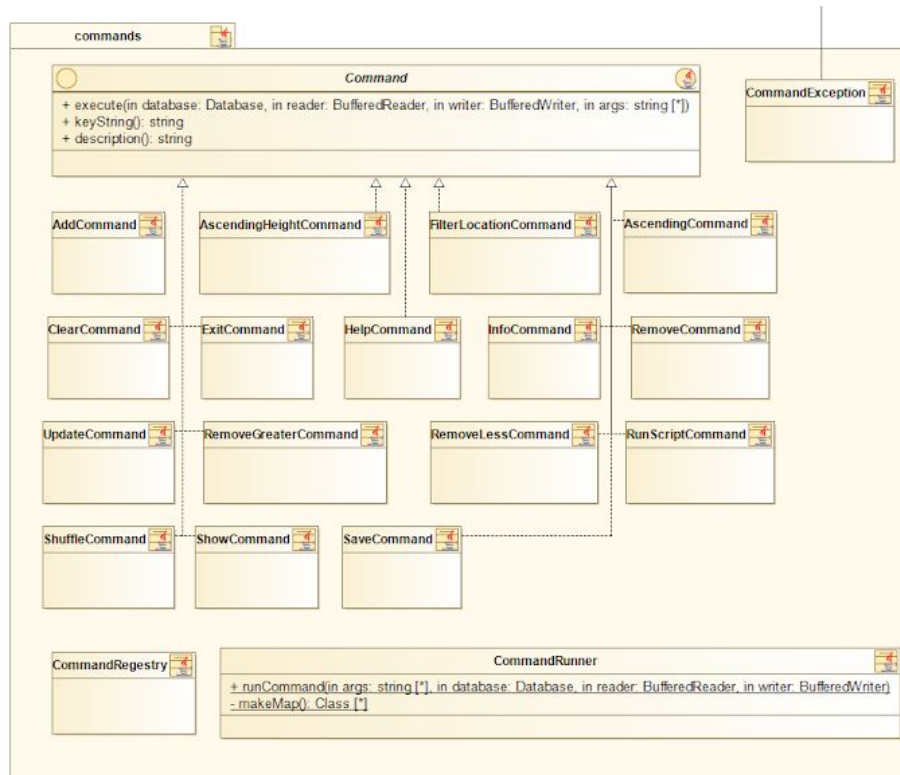
/**
 * @brief Thrown when structure cannot be loaded.
 */
public class WrongStructureFormatException extends Exception {
    public String toString() { return "Illegal value format"; }
}

```

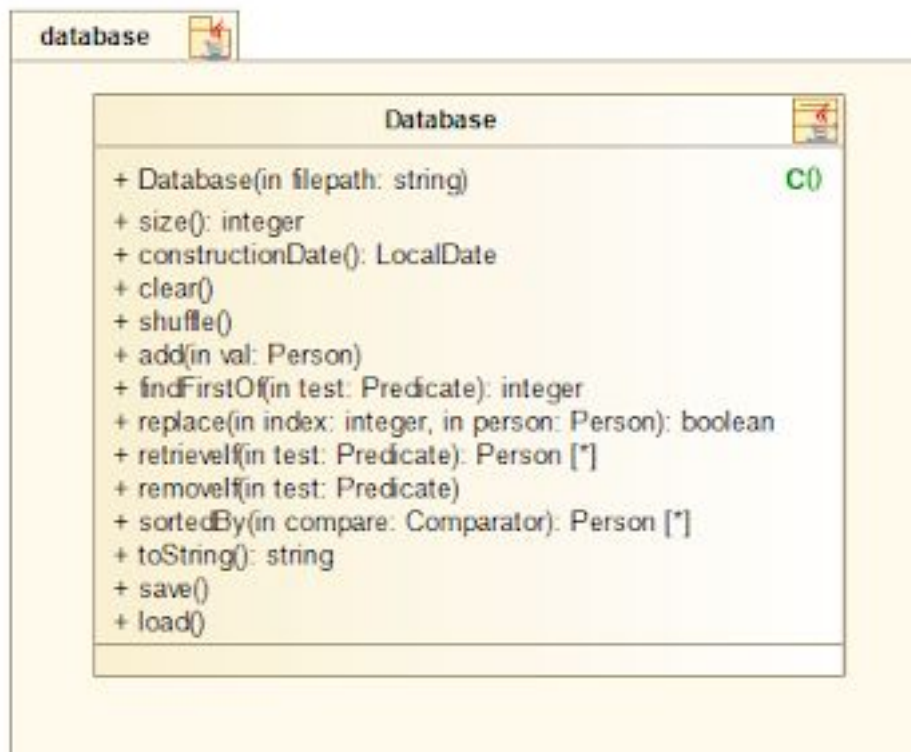
Структура проекта:

Весь исходный код проекта разбит на 3 пакета:

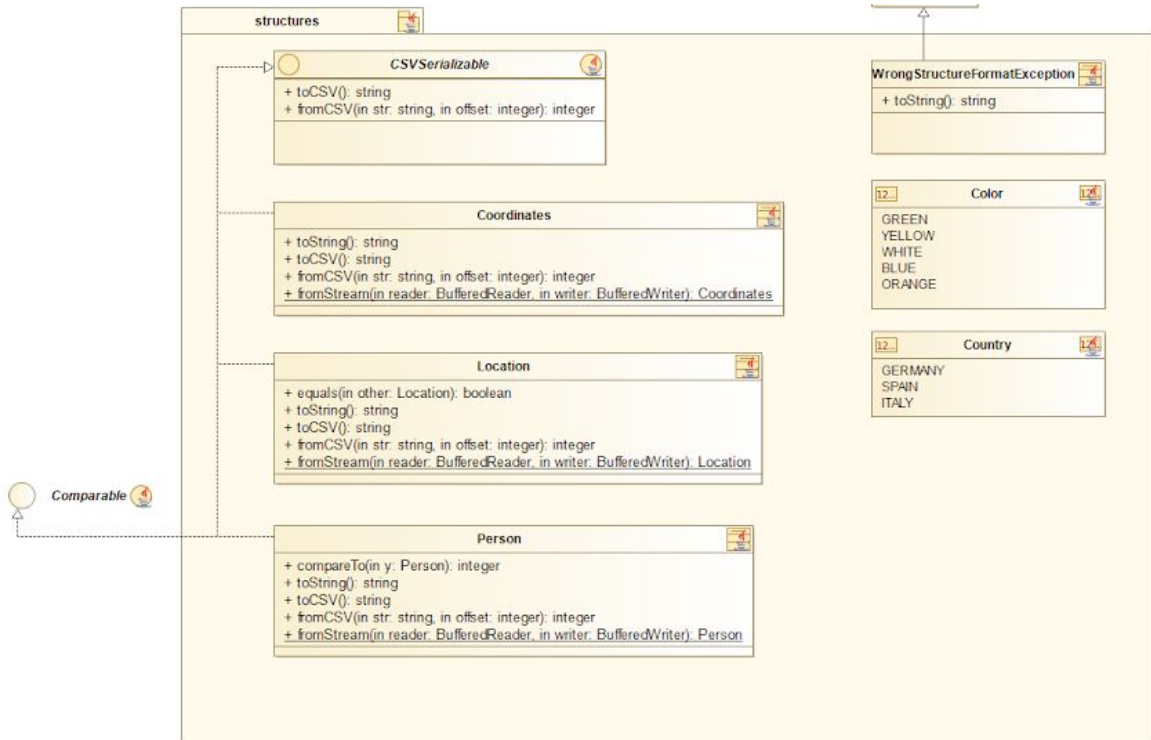
- commands – команды и все что надо для их выполнения



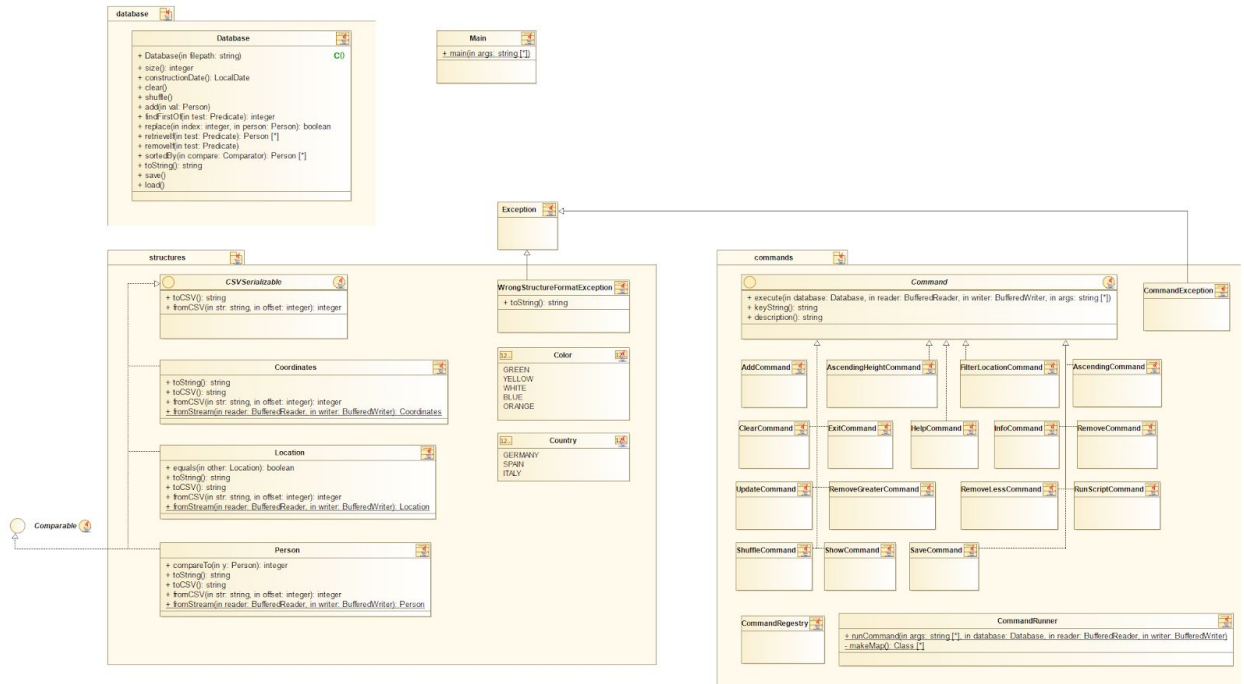
- database – база данных



- 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?