POST OFFICE SIMULATOR

Project Documentation

Karolina Kołek

1. Project description

This project is the post office simulator. It is a console application written in JAVA. After starting the application, user the user follows the instructions on the screen.

2. Description of the application's functionality

At the beginning, information from file named `packageRegistry.csv` about sent packages are being saved in map named `packages` in class named `PackageManager` and information containing sent letters from file named `lettersRegistry.csv` are being saved in map named `letters` in class `LetterManager`. Then user see menu. User can choose one of the following options:

- send a letter
- send a package
- send a telegram
- change recipient or path to file of letter that has already been sent before
- change recipient or description of package that has already been sent before
- check if there is any letter for user
- check if there is any package for user
- cancel letter's shipment
- cancel package's shipment
- exit.

2.1. Sending a letter

To send the letter, user has to enter his or her name, surname, address and then the same information about recipient. Next user has to provide the path to the file which contains letter content. At the end the sending date is automatically assigned and letter is being added into list named `letters` in class `LetterManager`.

2.2. Sending a package

To send the package, user has to enter his or her name, surname, address and then the same information about recipient. Next user has to provide the package's weight and package's description. At the end the sending date is automatically assigned and package is being added into list named `packages` in class `PackageManager`.

2.3. Sending a telegram

To send the telegram, user has to enter recipient's phone number, message and his/her name and surname. Then the message is being immediately printed on the screen.

2.4. Changing recipient or path to file of letter that has already been sent before and changing recipient or description of package that has already been sent before

To change this information, user has to enter his/her name, surname and address. Then every letter/package that has been sent by this user appears on the screen and user can choose if he/she wants to change something in this letter/package or not. If user wants to change something in letter or package that hasn't been sent, the following information appears on the screen:

`There's nothing to update.`

2.5. Checking if there is any letter or package for user

To check this information, user has to enter his/her name, surname and address. Then every letter/package that has been sent to this user appears on the screen. If there isn't any letter or package for user, the following information appears on the screen:

`There's nothing for you.`

2.6. Canceling letter's or package's shipment

To cancel letter's or package's shipment, user has to enter his/her name, surname and address. Then every letter/package that has been sent by this user appears on the screen and user can choose if he/she wants to cancel it or not. If user wants to cancel letter or package that hasn't been sent, the following information appears on the screen:

`There's nothing to cancel.`

2.7. Exit

Before the program finishes running, data from `letters` and `packages` are being saved in files, then these maps are being cleared and following information appears on the screen:

`Thank you for using our services. See you soon!`

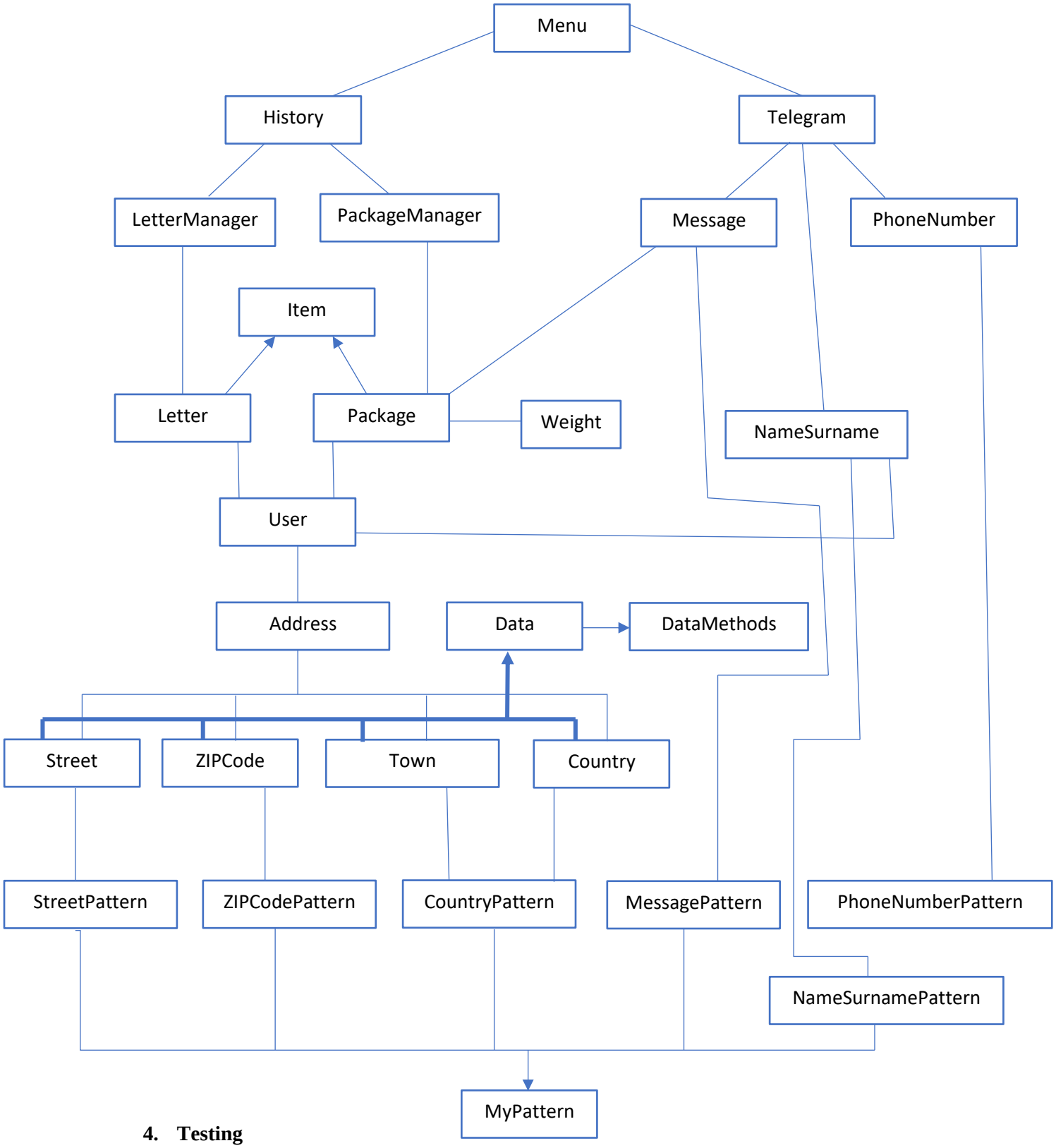
3. Internal specification

3.1. Classes:

- `Address` – class responsible for retrieving and storing senders' and recipients' addresses. This class stores Street, Country and ZIPCode objects.
- `Country` – class responsible for retrieving and storing senders' and recipients' country.
- `CountryPattern` – class contains pattern of what a country name should look like. This class is used for towns too.
- `Data` – class that is inherited by `Street`, `Country`, `ZIPCode`, `PhoneNumber`, `Message` and `NameSurname`. This class stores `String` variable and method that is responsible for get from user information and check if it is correct (if data entered by user matches specified pattern).
- `DataMethods` – interface extended by `Data` class.

- `History` - abstract class after which inherit `PackageManager` and `LetterManager.`
- `Item` - abstract class after which inherit `Package` and `Letter.`
- `Letter` - class extends `Item`. `Letter` is responsible for creating and getting information about new objects (new letters).
- `LetterManager` - class extends class `History`. `LetterManager` is responsible for saving and processing data related to letters' shipment.
- `Menu` – class responsible for printing menu and executing methods that have been chosen by the user.
- `Message` – class responsible for getting and storing message or description from user.
- `MessagePattern` - class contains pattern of what a message should look like. This class is used for description too.
- MyPattern – class that is being extended by `NameSurnamePattern, MessagePattern, StreetPattern, CountryPattern, PhoneNumberPattern` and `ZIPCodePattern.`
- `NameSurname` - class responsible for getting and storing user's name and surname.
- `NameSurnamePattern` - class contains pattern of what a user's name and surname should look like.
- `Package` - class extends `Item`. `Package` is responsible for creating and getting information about new objects (new packages).
- `PackageManager` - class extends class `History`. `PackageManager` is responsible for saving and processing data related to packages' shipment.
- `PhoneNumber`– class responsible for getting and storing phone number from user.
- `PhoneNumberPattern` - class contains pattern of what a phone number should look like.
- `Street`– class responsible for getting and storing street name and house number from user.
- `StreetPattern` - class contains pattern of what a street name and house number should look like.
- `Telegram` - class responsible for getting form user information about telegram and sending telegrams. This class stores `PhoneNumber`, `Message` and `NameSurname` objects.
- `Town` – class responsible for getting and storing town name from user.
- `User` – class responsible for getting and storing information about senders and recipients. This class stores `NameSurname` and `Address` objects.
- `Weight` – class responsible for getting and storing information about packages' weight.
- `ZIPCode` – class responsible for getting and storing ZIP code number from user.
- `ZIPCodePattern` - class contains pattern of what a ZIP code should look like.

3.2. Diagram showing the class hierarchy



4. Testing

I created unit tests that check correctness of the patterns. I also manually check various program usage scenarios.