



Post-Center Application Project

420-SF2-RE Data Structures
and Object Oriented
Programming

Nathan Cheng

Table of Contents

Project Description	1
Scenario	1
Program features	2
Hierarchical classes	2
User-Defined Interface	3
Text I/O	5
Comparators & Comparable	7
Unit Testing.....	8
User-Defined Exceptions	8
User Interface	9
Passwords	10
Challenges	11
Unimplementations.....	11
Issues.....	11
Learning Outcome	12

PROJECT DESCRIPTION

In separate deliverables, this individual project assignment will show the student's progression and understanding in developing an application under specific rules and requirements. Designing and implementing this work will focus on several objectives and criteria such as:

- Implementing class hierarchy with various variables and functions.
- Leveraging diverse data structure to efficiently manage data.
- Managing exceptions and I/O files, incorporating user-defined interfaces, polymorphism and overloading/overwriting methods.
- Following Test-Driven Development standards.

As well as the fundamental approaches in implementing functions and applications like documentations and objects, the work assigned to the student must highlight the following design requirements:

- Appropriate data structures that best fit the data.
- At least two hierarchies of classes, each one has two layers (such as User, Teacher, Student) is required for the project.
- At least one user-defined interface with at least one abstract method in it.
- At least one runtime-polymorphism
- At least one text I/O (reading and writing) function
- At least one class implement Comparable, and at least one Comparator class.
- Unit testing for user-defined methods

SCENARIO

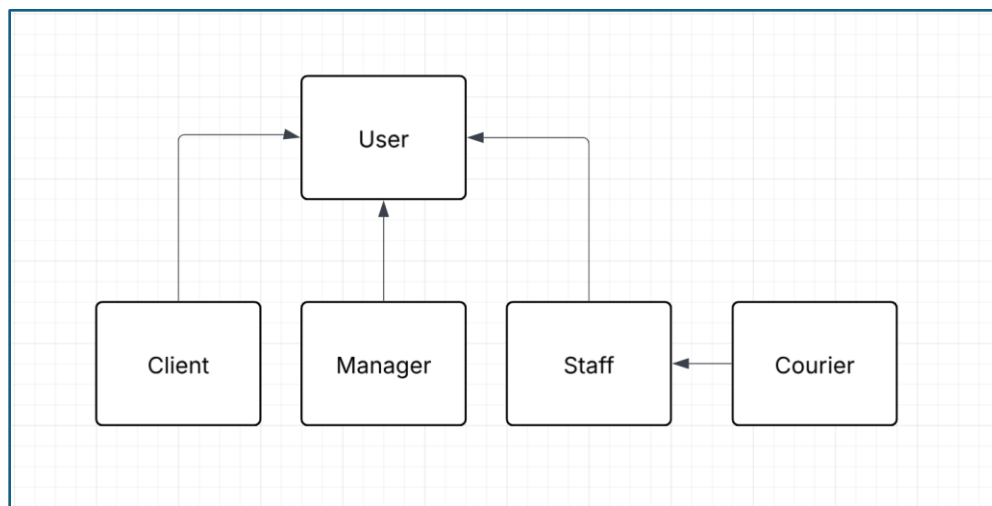
This project assignment will follow a rudimentary structure or a similar system that of a post office center. For this project, it will encapsulate basic functionalities such as managing and monitoring deliveries, monitoring requests and tickets, storing data and deliveries, and associating deliveries and roles to each user who uses the application.

PROGRAM FEATURES

HIERARCHICAL CLASSES

To satisfy a first requirement, the project includes two different hierarchies:

To begin with, the User family can be divided into 3 subclasses: Clients, Manager, and Staff classes, including a Courier class that descends from the Staff class. In this hierarchy, the User class is abstracted, and it will hold all basic actions that a user can perform like register and sending objects to each other. For the Client subclass, the clients are able to register their shared information, send deliveries, and monitor their inbox. Like the clients, the staffs can also manage deliveries and tickets in the system. The manager can all operations the staffs could, and he or she can reassign the roles of any Staff or Courier. The courier can only distribute assigned parcels.



```

7 public class Client extends User implements Req
8     private static int nextId = 0; 2 usages
9
10     private int clientId; 7 usages
11     private String address; 7 usages
12
13     public Client(String name, String email,
14         super(name, email);
15         this.clientId = nextId++;
16         this.address = address;
17     }
18
19     /**
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100

```

Tests passed: 0 of 1 test

```

3
4 Enter your email :
5 NathanCheng@hotmail.com
6 Enter your password :
7 Turtle
8 Welcome to the client Menu!
9
10 [1] View delivery inbox
11 [2] Send mail
12 [3] Send parcel
13 [4] Remove delivery from inbox
14 [5] Create bug report
15 [6] Create support request
16 [7] Exit

```

```

7 public class Staff extends User implements Req
8     protected static int nextId = 0; 2 usages
9     protected static Queue<Parcel> processedPa
10
11     protected int staffId; 9 usages
12     private Queue<Ticket> ongoingTickets = new
13     protected Role role; 7 usages
14
15     public Staff(String name, String email) {
16         super(name, email);
17         this.staffId = nextId++;
18         this.role = Staff.Role.INDOORS;
19     }
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100

```

Tests passed: 0 of 1 test

```

3
4 Enter your email :
5 Nathan@hotmail.com
6 Enter your password :
7 Pinotta
8 Welcome to the staff Menu!
9
10 [1] View delivery inbox
11 [2] Send mail
12 [3] Send parcel
13 [4] Process ongoing parcel
14 [5] View client list
15 [6] View Staff list
16 [7] Review next ticket
17 [8] Remove delivery from inbox
18 [9] Remove delivery from system
19 [10] View all the deliveries
20 [11] Exit

```

```

7 public class Courier extends Staff {
8     private List<Parcel> parcels; 10 usages
9
10     public Courier(String name, String email) {
11         super(name, email);
12         this.role = Role.COURIER;
13         this.parcels = new ArrayList<>();
14     }
15
16     /**
17
18     * displays all the parcel in the Courier's
19     * @param sorting the sorting format of the
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100

```

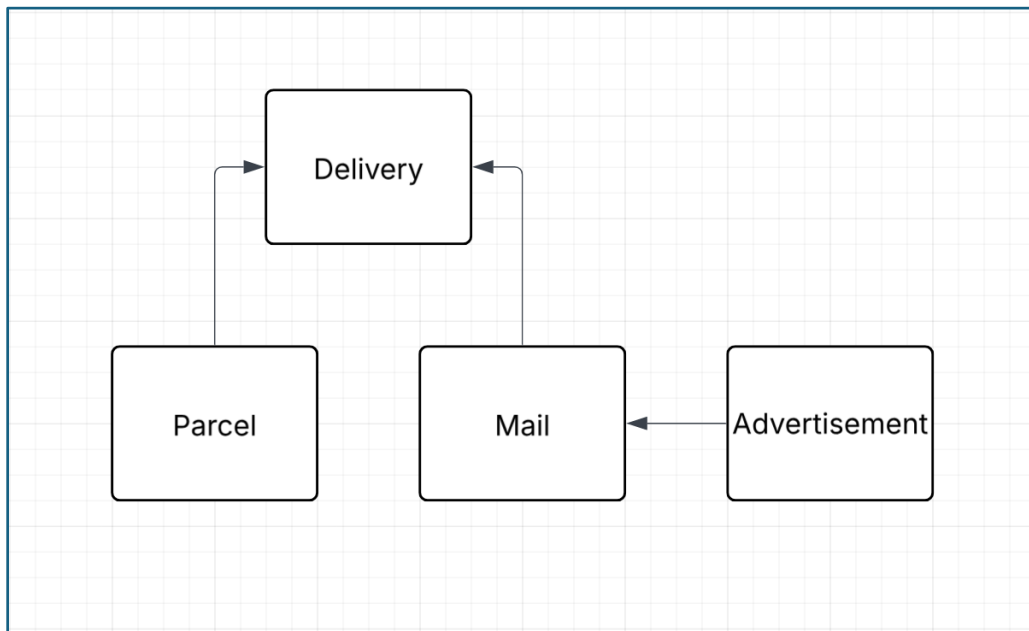
ests passed: 0 of 1 test

```

3
4 [4] Exit
5
6 Enter your email :
7 Queng@hotmail.com
8 Enter your password :
9 Personnel5
10 Welcome to the courier menu!
11
12 [1] View delivery inbox
13 [2] Send mail
14 [3] View client list
15 [4] View Staff list
16 [5] Remove delivery from inbox
17 [6] View ongoing parcels
18 [7] Deliver parcels
19 [8] Exit

```

For the second group, the Delivery abstract class is divided into 2 subclasses: Parcel and Mail, including an Advertisement class that is specified from the Mail class. Since these classes are essentially objects, they do not contain any functions to operate. However, the Delivery abstract class holds common variables such as address, details, and the time of its arrival. On the other hand, the Parcel class has an Item object, and the Advertisement class has its respective company name as a variable.



Each hierarchy has at least 2 layers or subclasses, fitting for the given task.

USER-DEFINED INTERFACE

In the development of this project, two user-defined interfaces are implemented:

- **Registerable:** The Registerable interface provides one abstract method called register() to the Client and Staff. The implementation of this interface allows the User subclasses to register their shared information into the system's data files and essentially create an account. In addition, this method is used to differently depending on the User subclass that calls it, displaying an example of runtime-polymorphism. When a Staff calls, the Staff themselves is registered to the Staff database, and it is similar for the Client class.

```
Staff.java x UsersDeliveryManaging.java x Client.java x Registerable.java x M: OpenProjectDescripti...
5 public class Staff extends User {
13 public Staff(String name, String email, String address, String password) {
15     this.staffId = nextId++;
16     this.role = Staff.Role.INDOORS;
17 }
18
19 /**
20  * registers the Staff to the Post-Office database
21  * updates the Post-Office data after registration
22  */
23 @Override @SuppressWarnings("Nathanael")
24 public void register(String password) {
25     PostOffice.staffs.add(this);
26     PostOffice.staffSecurityPass.put(this, password);
27 }
28 }

7 public class Client extends User {
13 public Client(String name, String email, String address, String password) {
15     this.clientId = nextId++;
16     this.address = address;
17 }
18
19 /**
20  * registers the client to the Post-Office database
21  * updates the Post-Office data after registration
22  */
23 @Override @SuppressWarnings("Nathanael")
24 public void register(String password) {
25     PostOffice.clients.add(this);
26     PostOffice.clientSecurityPass.put(this, password);
27 }
28 }

1 package org.example;
2
3 public interface Registerable {
4     void register(String password);
5 }
6 }

Tests passed: 0 of 1 test

1
Enter your name :
Filippo
Enter your email :
Bruno@gmail.com
Enter a password :
example
Enter your address :
Notre-Dame
Registered!

Welcome to Canada Post :

[1] Register as a client
[2] Register as a staff
[3] Log in
[4] Exit

Welcome to Canada Post :

[1] Register as a client
[2] Register as a staff
[3] Log in
[4] Exit

2
Enter your name :
Viken
Enter your email :
vk@hotmail.com
Enter a password :
life
Registered!
```

The top and bottom frames of the console show respectively the Client register menu and the Staff register menu. The difference between the two menu is that the Staff register menu does not ask for the user's address, which means that the register function reacts differently when the User is a Staff or a Client.

- **UsersDeliveryManaging:** The UsersDeliveryManaging interface consists of user-defined methods that Staff and Manager use as tools in order to directly operate with post office's system and its data.

```
[7] Review next ticket
[8] Remove delivery from inbox
[9] Remove delivery from system
[10] View all the deliveries
[11] Exit

10

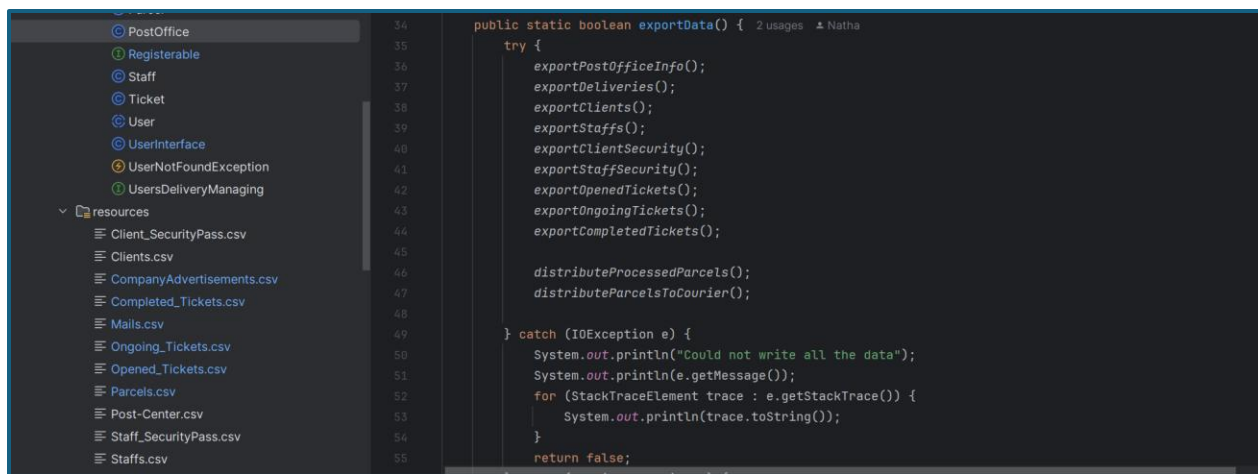
[1] Latest
[2] Reverse
[3] Exit

1
Message : Title : Testing , Delivery ID : 2, Address : Rue de boheme, Detail : This message is a test, Acquisition Date : 2025-05-14T00:55:38.564560900, Quantity : 2, Courier ID : 3, Delivery ID : 1,
Message : Title : Assignment, Delivery ID : 3, Address : 100e avenue, Detail : I need help, Acquisition Date : 2025-05-14T00:55:38.564560900, Quantity : 1, Courier ID : 2, Delivery ID : 0, Address :
Parcel : Parcel ID : 1, Item : Gym Equipment 30.00kg 2025-05-14T00:55:38.564560900, Quantity : 2, Courier ID : 3, Delivery ID : 1,
Parcel : Parcel ID : 0, Item : Mouse 1.60kg 2025-05-14T00:55:38.564560900, Quantity : 1, Courier ID : 2, Delivery ID : 0, Address :
```

From the UsersDeliveryManaging, one of the defined methods called viewAllDelivery() is called here by the a Staff user to display all the deliveries in the post office's system. These methods are not given to the Client class.

TEXT I/O

The data of the post office's system is exported from the system into files, and then they are imported from files to the system during importation. In the Post Office class, the exportData() and the importData() implemented functions each have its own inner functions dedicated to managing different data like tickets, staffs' & clients' information, etc. :



The .csv files on the side are where the data are stored.

ExportData() example:

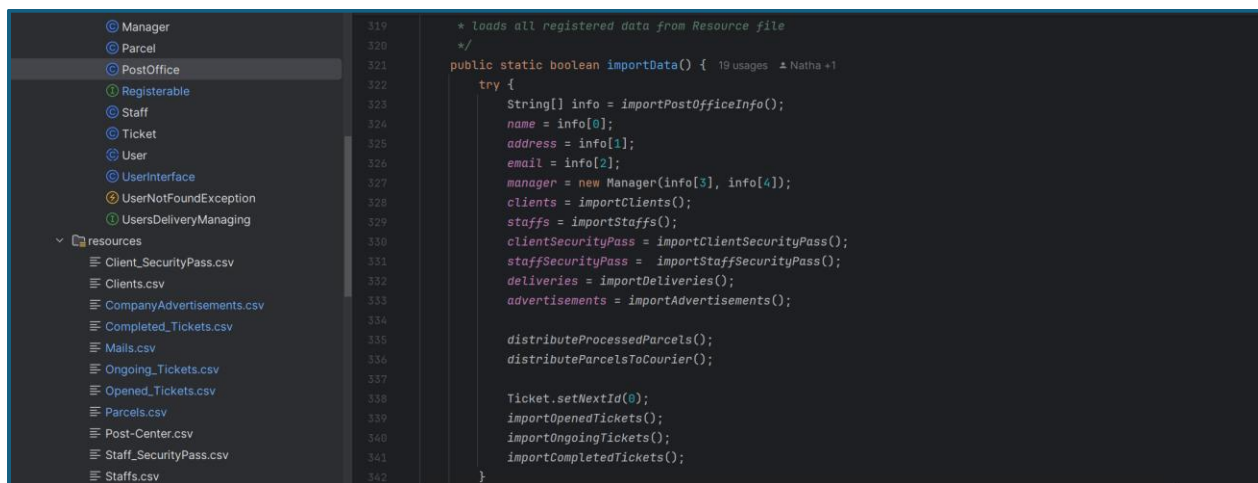
```

/**
 * inner function for exportData() function that exports all the post-center's shared information
 * @throws IOException Output data exceptions during writing data
 * @throws RuntimeException General unchecked exception
 */
private static void exportPostOfficeInfo() throws IOException, RuntimeException { 1usage ±Natha +1
    File file = new File(POSTCENTER_FILE_PATH);

    try (FileWriter fw = new FileWriter(file)) {
        fw.write(str: name + ",");
        fw.write(str: address + ",");
        fw.write(str: email + ",");
        fw.write(str: manager.getName() + ",");
        fw.write(str: manager.getEmail() + "\n");
    }
}

```

The importData() also utilizes the .csv files where the exportData() used to store data:

**importData () example:**

```

/**
 * inner function for importData() that imports all information data from Post-Center.csv
 * @return a String array containing the shared information of the post-office
 * @throws RuntimeException general unchecked exception
 */
private static String[] importPostOfficeInfo() throws RuntimeException { 1usage ±Natha +1
    File file = new File(POSTCENTER_FILE_PATH);

    String[] info = new String[]{};

    try (Scanner input = new Scanner(file)) {
        info = input.nextLine().split(regex: ",");
    } catch (FileNotFoundException fnfe) {
        handler(fnfe, POSTCENTER_FILE_PATH);
    }

    return info;
}

```


COMPARTORS & COMPARABLE

Most classes inside the project like the Ticket, Staff, Client, Parcel, and Mail is implemented with a comparator, for the user might want the application to display these data in a specific order.

```

1
Client : Client ID : 1, Name : Jacques, Email : Jpc@hotmail.com, Address : 100e avenue,
Client : Client ID : 0, Name : Nathan, Email : NathanChg@hotmail.com, Address : rue de boheme,

[1] By Name alphabetically
[2] By Name reverse alphabetically
[3] By Id reverse
[4] By Id
[5] Exit

2
Client : Client ID : 0, Name : Nathan, Email : NathanChg@hotmail.com, Address : rue de boheme,
Client : Client ID : 1, Name : Jacques, Email : Jpc@hotmail.com, Address : 100e avenue,

[1] By Name alphabetically
[2] By Name reverse alphabetically
[3] By Id reverse
[4] By Id
[5] Exit

3
Client : Client ID : 1, Name : Jacques, Email : Jpc@hotmail.com, Address : 100e avenue,
Client : Client ID : 0, Name : Nathan, Email : NathanChg@hotmail.com, Address : rue de boheme,

[1] By Name alphabetically
[2] By Name reverse alphabetically
[3] By Id reverse
[4] By Id
[5] Exit

4
Client : Client ID : 0, Name : Nathan, Email : NathanChg@hotmail.com, Address : rue de boheme,
Client : Client ID : 1, Name : Jacques, Email : Jpc@hotmail.com, Address : 100e avenue,

```

Unlike all other classes, the Advertisement class utilizes a comparable instead of comparator to format its order since an advertisement does not need to be order in a specific format.

```

public class Advertisement extends Mail implements Comparable<Advertisement> {
    // naturally sorts the advertisements by company names
    // @param o the object to be compared
    // @return Integer value of comparison
    //
    @Override
    public int compareTo(Advertisement o) {
        return this.companyName.compareTo(o.companyName);
    }
}

Tests passed: 0 of 1 test

Nathan@hotmail.com
Enter your password :
P1n0tt4
Welcome to the staff Menu!

[1] View delivery inbox
[2] Send mail
[3] Send parcel
[4] Process ongoing parcel
[5] View client list
[6] View Staff list
[7] Review next ticket
[8] Remove delivery from inbox
[9] Remove delivery from system
[10] View all the deliveries
[11] View all advertisements
[12] Exit

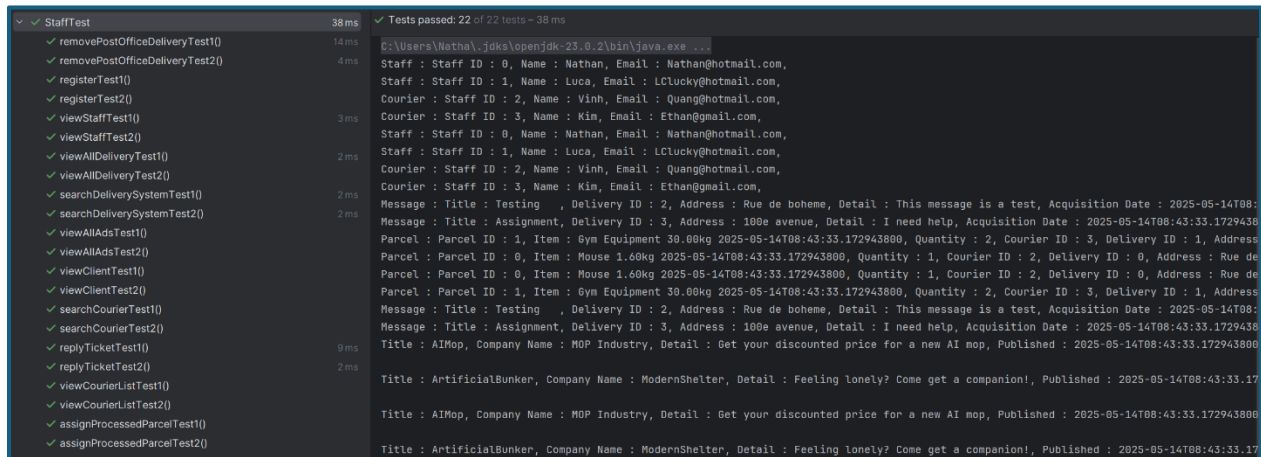
11
Title : AIMop, Company Name : MOP Industry, Detail : Get your discounted price for a new AI mop, Published : 2025-05-14T08:43:33.
Title : ArtificialSunkers, Company Name : ModernShelter, Detail : Feeling lonely? Come get a companion!, Published : 2025-05-14T08:

```

UNIT TESTING

To begin with, the `UserInterfaceTest` and the `PostOfficeIOTest` are implemented first before all the other tests classes to initialize the system's data and examine in depth the data when running. Regardless, each class with at least a public user-defined method is given an Unit Testing class to ensure that each function produce its intended outcome.

Example: StaffTest



```
✓ StaffTest 38 ms ✓ Tests passed: 22 of 22 tests - 38 ms
  ✓ removePostOfficeDeliveryTest1() 14 ms
  ✓ removePostOfficeDeliveryTest2() 4 ms
  ✓ registerTest1()
  ✓ registerTest2()
  ✓ viewStaffTest1() 3 ms
  ✓ viewStaffTest2()
  ✓ viewAllDeliveryTest1() 2 ms
  ✓ viewAllDeliveryTest2()
  ✓ searchDeliverySystemTest1() 2 ms
  ✓ searchDeliverySystemTest2() 2 ms
  ✓ viewAllAdsTest1()
  ✓ viewAllAdsTest2()
  ✓ viewClientTest1()
  ✓ viewClientTest2()
  ✓ searchCourierTest1()
  ✓ searchCourierTest2()
  ✓ replyTicketTest1() 9 ms
  ✓ replyTicketTest2() 2 ms
  ✓ viewCourierListTest1()
  ✓ viewCourierListTest2()
  ✓ assignProcessedParcelTest1()
  ✓ assignProcessedParcelTest2()

C:\Users\Natha\jdk\openjdk-23.0.2\bin\java.exe ...
Staff : Staff ID : 0, Name : Nathan, Email : Nathan@hotmail.com,
Staff : Staff ID : 1, Name : Luca, Email : LClucky@hotmail.com,
Courier : Staff ID : 2, Name : Vinh, Email : Quang@hotmail.com,
Courier : Staff ID : 3, Name : Kim, Email : Ethan@gmail.com,
Staff : Staff ID : 0, Name : Nathan, Email : Nathan@hotmail.com,
Staff : Staff ID : 1, Name : Luca, Email : LClucky@hotmail.com,
Courier : Staff ID : 2, Name : Vinh, Email : Quang@hotmail.com,
Courier : Staff ID : 3, Name : Kim, Email : Ethan@gmail.com,
Message : Title : Testing , Delivery ID : 2, Address : Rue de boheme, Detail : This message is a test, Acquisition Date : 2025-05-14T08:
Message : Title : Assignment, Delivery ID : 3, Address : 100e avenue, Detail : I need help, Acquisition Date : 2025-05-14T08:43:33.1729438
Parcel : Parcel ID : 1, Item : Gym Equipment 30.00kg 2025-05-14T08:43:33.172943800, Quantity : 2, Courier ID : 3, Delivery ID : 1, Address
Parcel : Parcel ID : 0, Item : Mouse 1.60kg 2025-05-14T08:43:33.172943800, Quantity : 1, Courier ID : 2, Delivery ID : 0, Address : Rue de
Parcel : Parcel ID : 0, Item : Mouse 1.60kg 2025-05-14T08:43:33.172943800, Quantity : 1, Courier ID : 2, Delivery ID : 0, Address : Rue de
Parcel : Parcel ID : 1, Item : Gym Equipment 30.00kg 2025-05-14T08:43:33.172943800, Quantity : 2, Courier ID : 3, Delivery ID : 1, Address
Message : Title : Testing , Delivery ID : 2, Address : Rue de boheme, Detail : This message is a test, Acquisition Date : 2025-05-14T08:
Message : Title : Assignment, Delivery ID : 3, Address : 100e avenue, Detail : I need help, Acquisition Date : 2025-05-14T08:43:33.1729438
Title : AIMop, Company Name : MOP Industry, Detail : Get your discounted price for a new AI mop, Published : 2025-05-14T08:43:33.172943800
Title : ArtificialBunker, Company Name : ModernShelter, Detail : Feeling lonely? Come get a companion!, Published : 2025-05-14T08:43:33.17
Title : AIMop, Company Name : MOP Industry, Detail : Get your discounted price for a new AI mop, Published : 2025-05-14T08:43:33.172943800
Title : ArtificialBunker, Company Name : ModernShelter, Detail : Feeling lonely? Come get a companion!, Published : 2025-05-14T08:43:33.17
```

Unit Testing help displaying any unintentional operations, especially when registering and updating data within the files.

USER-DEFINED EXCEPTIONS

When implementing a `UserInterface` class, several user-defined `RuntimeException` subclasses are created in order to identify and resort specific issues such as misinputs, non-real option, incorrect writing format, and when function does not a specific object inside the system:

- `UserNotFoundException`
- `DeliveryNotFoundException`
- `EmailNotFoundException`
- `InvalidEmailException`
- `InvalidInputException`
- `InvalidNameException`
- `InvalidNumberOptionException`

Example: staffMenu()

```
} catch (InvalidNumberOptionException inoe) {
    System.out.println("Please select the following options.\n");
} catch (InputMismatchException ime) {
    System.out.println("Please remove any symbols and enter an Integer.\n");
} catch (NumberFormatException nfe) {
    System.out.println("Enter any of the following options\n");
}
}
```

Tests passed: 0 of 1 test

terfaceTest

nuTest()

```
[1] View delivery inbox
[2] Send mail
[3] Send parcel
[4] Process ongoing parcel
[5] View client list
[6] View Staff list
[7] Review next ticket
[8] Remove delivery from inbox
[9] Remove delivery from system
[10] View all the deliveries
[11] View all advertisements
[12] Exit

gtj
Enter any of the following options

Welcome to the staff Menu!

[1] View delivery inbox
[2] Send mail
```

In almost every scanner input, the input is checking for any possible exceptions before throwing its respective exception.

USER INTERFACE

When designing the project assignment, the `UserInterface` class is implemented to take an input of a user's response with the least possible inconsistency. By obligating the users to insert a number on the console, the interaction between the user interface and the user becomes more user friendly and reduces the likelihood of producing an issue in the post office's system.

Example: staffMenu()

```
Welcome to the staff Menu!

[1] View delivery inbox
[2] Send mail
[3] Send parcel
[4] Process ongoing parcel
[5] View client list
[6] View Staff list
[7] Review next ticket
[8] Remove delivery from inbox
[9] Remove delivery from system
[10] View all the deliveries
[11] View all advertisements
[12] Exit

3
Enter the item name :
ball
Detail :
|
```

PASSWORDS

After the first implementation of the `UserInterface` class, the security pass files for each `Client` and `Staff` classes is implemented with the intentions to create accounts with passwords by using `Map<>`. By mapping every `Client` or `Staff` to a password/ `String`, the user will be able to officially register a password to their account:

```
public static Map<Staff, String> staffSecurityPass = new TreeMap<>(new Staff.StaffComparator( (type: "") ));
public static Map<Client, String> clientSecurityPass = new TreeMap<>(new Client.ClientComparator( (type: "") ));

public PostOffice() {
    // ...
}

// ...

Tests passed: 0 of 1 test

UserInterfaceTest
  menuTest()
    [1] Register as a client
    [2] Register as a staff
    [3] Log in
    [4] Exit

    3
    Enter your email :
    Nathan@hotmail.com
    Enter your password :
    Pinotta
    Welcome to the staff Menu!

    [1] View delivery inbox
    [2] Send mail
    [3] Send parcel
    [4] Process ongoing parcel
    [5] View client list
    [6] View Staff list
    [7] Review next ticket
    [8] Remove delivery from inbox
    [9] Remove delivery from system
    [10] View all the deliveries
    [11] View all advertisements
    [12] Exit
```

CHALLENGES

UNIMPLEMENTATIONS

Because I am late on the schedule, multiple other possible implementations are not yet been added:

- Ensuring unique emails during registration of a Client or Staff to prevent other Client or Staff account from overlapping each other
- Making a manager menu or an user interface for the manager in order to properly access the manager's account in the post-office
- Creating a Refundable interface for the Parcel class to verify if the Parcel has pass over 30 days after receiving it
- Creating a Expirable interface for Mail & Advertisement classes to remove overdue mails
- Implementing an entire shopping catalogue with partnering produce companies if I had even more time

ISSUES

During the development of the project assignment, several conflicts hindered the progression and overall performance & efficiency of the project:

- Inconsistency and complexity of the implementation of the `exportData()` and `importData()` functions along with their inner methods.
- Complexity of the user interface and data management
- Unable to push new commits and changes to the Github repository because of a deleted file
- Unable to type on the IntelliJ console
- Unable to run on IntelliJ due to beta version of JDK 24
- Problem with managing data with List that are deemed "Immutable", which is why I changed all the `add()` and `remove()` method calls with stream and lambda expressions.
- Choosing the access modifiers adequately

LEARNING OUTCOME

After completing the project assignment, there are major improvements that can be made to facilitate the development of the application: my approach toward this project is unorganized and needs to spend more time on designing the project at the start. Also, simplifying a very complex part of the project is another way to potentially improve or learn more.

Possible improvements:

- Organize my work approach by adding the documentation and the unit testing after finishing implementing a function
- Organize adequately commits and pushes with specific comments instead of commit and push a large bulk of changes
- Prioritize the testing of the current function
- Be more specific when designing and developing the project at the start
- Avoid complex coding by simplifying