GIT Department of Computer Engineering
CSE 222/505 - Spring 2020
Homework # Report

HW1-Part2

Nevzat Seferoglu
171044024

1. SYSTEM REQUIREMENTS

Problem Description:

The automation system for a cargo company has users such as administrators, branch employees, transportation personnel and customers. Administrators manage the system by adding and removing branches, branch employees and transportation personnel. Branch employees can enter and remove information about the shipments sent from that branch, add and remove users (customers) to the system. The information of the sender and receiver are recorded for each shipment. When the package arrives at a branch or leaves the branch, its current status is entered into the system by the branch employee. When it is delivered, the transportation personnel make the update. The customer entering the system with the tracking number is only authorized to see the name and surname information of the sender and receiver and the current status of the cargo.

Version Extension Summary: The alteration has occurred at how data is kept. System uses ''Array List "java utility collection instead of proprietary list. There four functions, which belongs to array list itself, in this version.

Statements that is below explain each function and its usage in this version.

Functional and Non-functional requirements are located below.

Functional Requirement: There are users who can access the system and manipulate the system data. All entry that are sent to the system are 'String'.

➔ **Administrator**:

- **Must**, add an employee.
  - There are two main function that sustain this property. One of them takes **employee-id** as a **String** type, the other one takes **Employee abstract** class type that are illustrated in the class diagram.
    Note: These two different functions **may** call each other.
- **Must**, remove an employee.
  - There are two main function that sustain this property. One of them takes **employee-id** as a **String** type, the other one takes Employee abstract class type that are illustrated in the class diagram.
    Note: These two different functions **may** call each other.
- **Must**, add a branch.
  - There are two main function that sustain this property. One of them takes **branch-id** as a **String** type, the other one takes **Branch** class type that are illustrated in the class diagram.
    Note: These two different functions **may** call each other.

- **Must**, remove branch.
    - There are two main function that sustain this property. One of them takes **branch-id** as a **String** type, the other one takes **Branch** class type that are illustrated in the class diagram.

  Note: These two different functions **may** call each other.

- **Should**, change branch of specific employee.
    - There are two main function that sustain this property. One of them takes **employee-id** and **branch-id** as a String type, the other one takes **Employee** and **Branch** class type that are illustrated in the class diagram.

  Note: These two different functions **may** call each other.

- **Should**, list all branch that are currently exist.

➔ **<u>Branch Employee:</u>**

- **Must**, add a customer.
    - Takes a shipment as an argument. Also, shipment must be created for creating a new customer. Because all shipment has customer with ‘ has a relation ‘.
- **Must**, remove a customer.
    - There are two main function that sustain this property. One of them takes **shipment-tracking-number** as a **String** type, the other one takes Shipment class type that are illustrated in the class diagram.

      Note: These two different functions **may** call each other.
- **Must**, set current status of shipment.
    - There is only one function that takes two different arguments as parameters. One of the is **shipment-tracking-number** as a **String** type, the other one is shipment-status as an **Enumeration** type illustrated in the class diagram.
- **Must**, Edit shipment information.
    - There are several function and properties for to that it is indicated in Javadoc.

➔ **Transpiration Personnel:**

- **Must**, set the current status of shipment.

Note: There is restriction switching status of shipment for the Transportation Employee.

- **Should,** list shipments which is in-transit status.

➔ **Customer:**

- **Must,** display the shipment information.

- Takes a tracking-number as a **String**, display some tracking number information.

Non-Functional Requirement:

contains (Object object): The function that checks the existence of given object over the list relying on **equals (Object object)** of the object itself. Maintainers of system must write equals method if they want to use array list.

add (Object object): The function that adds a new element to the list. Depends on the collections implementation, adding **null** operation may not be allowed. But for this collection adding null to the system is **allowed**. There is **no restriction** on the usage of it.

remove (Object object): The function that removes an existing element from the list. There is no restriction on using of it.

get (int index): The function that returns a specific element from the list according to given index. But system does not use method directly. There are some enhanced for loop in the system that relies on this function inside itself (**not sure**).

2. USE CASE DIAGRAMS

Attached to zip file.

3. CLASS DIAGRAMS

Attached to zip file.

4. OTHER DIAGRAMS

5. PROBLEM SOLUTION APPROACH

The problem was to create cargo company system. When I come to the problem cargo company has to keep data about both customer and company structure itself. There are also some essential structures that belongs to company such as branch, employee, shipments. All those structures must involve in a uniqueness system. When I come back to problem, first thing that comes my mind was to creating ids-based system. Thanks to this system I could make a search easy. According to each data type that has some properties and skills, I need to create an interactive data communication system so that It save me not to think about code structure instead some algorithm. Algorithms and code structure should be used in a balance. In complex system, code structures are hard to design and at the same time, should be time saving also.

Exploiting class properties was an enough idea for implement system. But there is critical issue about where the actual data locate. I thought that Company has all branch, shipment, employee and customer. I did not put additional class for all sub employees and shipments etc. There are two enumeration structure for them.

- Shipment Status
- Employee Positions

These enumerates are collectively works with their own function. When employee ask for some operation it also has some position in structure itself. Thanks to these methods, when Employee created by the developer, it is easy to add a new function or new employee position to system structure.

Approach For New Version: I add some interface and Abstract Class , thanks to these structure , exploited the method of polymorphism .

Polymorphic Function: EmployeePositions getPosition(); // located In EmployeeWorkspace interface.

There is also down casting in main () test function.

System Side Effect: All methods and properties of unique employee such as admin, branch employee, transportation employee is represented in same structure which is Employee class.

Side effect is fixed for the new version of system. Each structure has own class and interface for the new version.

Summary: **In this project, I learn that even basic methods and variable can be more complicate, and design based. Evaluating both structure and real-case scenario could be tricky.**

6. TEST CASES

Attached to zip file.

7. RUNNING AND RESULTS

Attached to zip file.

Not: 'Test Cases' and 'Running and Results' are in the same .txt file in assignment.