Package Delivery System
Phase #3

Members: Elizabeth Dayton, Caroline Smith, Paul Poe, Collier Robinson, and Adam Bostwick

Table of Contents

Phase #1 Documentation Updates

The updates we made to our phase 1 documentation include some domain class changes and some design class changes. We got rid of the Flats class, because since flats do not have tracking numbers and cannot be tracked or intercepted, it was unnecessary. We also put all of the functionality related to the user manipulating the package (intercepting or tracking) into a single class called WebsiteBoundary instead of having them randomly separated into various domain classes.

The final update we made was combining the two controller classes into a single controller class, because we thought having separate controller classes was redundant.

Phase #2 Documentation Updates

Because we made phase #2 with our two use cases in mind, there weren't many conceptual updates, but there were quite a few technical updates that include changing the names of some of the previous functions and also breaking some functions into multiple functions for more clarity and usability. For example, in phase 2 we described pre- and post-conditions for a method called verifyInterception(), which was meant to be a method that confirms if a package can be intercepted or not using the arguments tracking number and interception type. In our implementation, we ended up breaking this method into four methods – two methods for each type of intercept – instead of taking an intercept type as an argument.

Nothing changed with our state diagrams except for a few of the method names.

Implementation

Follow the GitHub link to view the code base for our package delivery system:
https://github.com/CutestCoder/Package_Delivery_System_2

Documentation

a. Code components developed
- We developed code for 2 use cases: intercepting a package, which includes redirecting or holding a package, and tracking a package.
    - **Parcel**: A class containing attributes of a package including tracking ID, delivery address, return address, status, interception flags and a priority flag.
    - **Address**: A class containing the components of an address including house number, street name, zip code, city and state.
    - **WebsiteUI**: A class that is a substitute for a user interface.
        - Handles all interaction between the user and the website, including getting user input for a tracking number, address, and action.
    - **DB_Connection_Handler:** A class that handles connection to a MySQL database
        - Separating this functionality into a separate class allows for a reusable way of connecting to a database. If more than one class needs to connect to a specific database, they can achieve that by utilizing this class.
    - **WebsiteBoundary:** A boundary class that is the boundary between our system's database and the user.
        - It handles communication between the user and the system by using the user's input to handle all database calls and returns relevant data back to the user interface.

b. b. Setup and Execution instructions in detail (for evaluation purpose)
- Open code in your preferred IDE or the command prompt.
- Compile/Build the code.
    - If you're using the command prompt, you should compile using the command `$ g++ -std=c++11 your_file.cpp -o your_program`
- Run the code.
- Follow the instructions that appear on the command prompt
    - Select an action you want to perform
    - Depending on the action, you will enter in the appropriate input
    - For each action, you should be able to view the output of the system to see if the action was successful or not.

c. c. Approach taken to translate design to code.
- The approach we took was to study all the diagrams we created, and then write the code to implement those diagrams.
    - We looked at the class diagrams and made sure that every necessary function was implemented in the appropriate classes.
    - We looked at the sequence diagrams for our two use cases and made sure every scenario, including ones involving errors, was handled in the system and the interface.

        - We looked at the pre- and post- conditions of our operation contract, and figured out how to implement those as conditional clauses so that if a pre-condition is not met, it would return an error to the user.

d. d. Issues faced, and lessons learned.
    - The issues we faced while implementing our project include
        - Learning about relational database management and learning how to connect to a database via c++.
        - Figuring out how to best organize our classes so that our code is reusable and maintainable.
        - Learning how we turn system errors into understandable display messages so that the user knows what to do next.