Software Quality – ENGR 3980

Swift Ticket Project - Phase 4

Back End Rapid Prototype

Design Brief

Created by

**Daniel Smullen**

**Rayan Alfaheid**

**Jonathan Gillett**

28/03/2013

Document Revision 1.0

Contents

[Solution Architecture 4](#_Toc352276014)

[Overview 4](#_Toc352276015)

[Input and Output 4](#_Toc352276016)

[UML Representation 4](#_Toc352276017)

[Classes 6](#_Toc352276018)

[AddCredit Class 6](#_Toc352276019)

[Requirements Addressed 6](#_Toc352276020)

[AvailableTickets Class 6](#_Toc352276021)

[Requirements Addressed 6](#_Toc352276022)

[Buy Class 6](#_Toc352276023)

[Requirements Addressed 6](#_Toc352276024)

[Create Class 7](#_Toc352276025)

[Requirements Addressed 7](#_Toc352276026)

[CurrentUserAccounts Class 7](#_Toc352276027)

[Requirements Addressed 7](#_Toc352276028)

[DailyTransaction Class 7](#_Toc352276029)

[Requirements Addressed 7](#_Toc352276030)

[Delete Class 7](#_Toc352276031)

[Requirements Addressed 7](#_Toc352276032)

[Exception Classes 8](#_Toc352276033)

[Requirements Addressed 8](#_Toc352276034)

[Logout Class 8](#_Toc352276035)

[Requirements Addressed 8](#_Toc352276036)

[Refund Class 8](#_Toc352276037)

[Requirements Addressed 8](#_Toc352276038)

[Sell Class 8](#_Toc352276039)

[Requirements Addressed 9](#_Toc352276040)

[Ticket Class 9](#_Toc352276041)

[Requirements Addressed 9](#_Toc352276042)

[Transaction Interface 9](#_Toc352276043)

[Requirements Addressed 9](#_Toc352276044)

[User Class 9](#_Toc352276045)

[Requirements Addressed 9](#_Toc352276046)

[Validate Class 9](#_Toc352276047)

[Requirements Addressed 9](#_Toc352276048)

# Solution Architecture

Our design incorporates an object-oriented approach, decomposing all major transactions into objects which are bound to an interface that specifies their common interactive components. Many design aspects from the frontend were reused in the backend. The main difference is that the architecture was reworked slightly to reflect the state validation functionality of the backend, instead of the input validation functionality of the frontend. A detailed introspection into the design follows.

## Overview

The central aspects of the design include the Transaction interface, which all the transactions within the backend realize. Also included is the utilization of classes which extend the Exception class – these are used for failed constraints and fatal errors, the FailedConstraint and FatalError classes respectively. This aspect was chosen because all the transactions have main commonalities, particularly in terms of their collaboration with other objects, and the failed constraints/fatal errors are simply exceptions which require additional functionality that the standard Java Exception class does not provide. Transactions are components of the DailyTransaction container object, which also contains methods for interaction with Transactions. Users are components of the CurrentUserAccounts container object also.

The ExceptionCodes object contains methods and stores data with respect to errors that occur within the program. It is a component within the architecture to the FailedConstraint class.

### Input and Output

Inputs and outputs are handled by classes respective of each input and output aspect. Input from transactions is handled within a class designated for each input file within the program. Outputs to individual files are handled by the same classes’ methods dedicated to each output file – the daily transaction file, current user accounts file, and the available tickets file.

### UML Representation

An object model diagram, complete with methods and attributes is presented below. This diagram shows the interaction between the classes that make up the program. In this diagram, the interface realization relationship between the Transaction class and the various transactions in the program becomes evident. Also, the composition of transactions into the daily transaction file, as well as the composition of users within the current user accounts file is evident.

#### UML Diagram

#### C:\Users\100440203\Documents\GitHub\qa-project\design\BackEnd\BackEnd.png

# Classes

The following is a list of the classes within the project. Please refer to the Software Requirements Specification for specific requirement information.

## AddCredit Class

Contains the methods and attributes for the addcredit transaction. This means that the actual underlying functionality for adding credit to user accounts is handled by this object. As per the original design and architecture, this class is utilized whenever the addcredit transaction is performed.

### Requirements Addressed

The requirements met are regarding processing transactions for the username provided, credit amount to be added, and updating the values in the current user accounts file. Constraints relevant to the addition of credit with respect to overflow are also addressed.

12, 12.1, 12.1.1, 12.1.2, 12.1.3, 12.2

## AvailableTickets Class

This class contains the methods and attributes related to the available tickets file, and interacting with that file on disk. This class provides methods for writing the output to the file, and also validates and stores the values in memory while the available tickets are interacted with by other operations.

### Requirements Addressed

The requirements met are regarding processing the available tickets file, reading in the list of available tickets into memory, and ensuring that there is no corruption.

14, 14.1, 14.2, 14.2.1, 14.2.2, 14.3, 14.3.1, 14.3.2, 14.4, 14.4.1, 14.4.2, 14.5, 14.5.1, 14.5.2, 14.6, 14.7. 14.8, 14.9, 14.10, 21

## Buy Class

Contains the methods and attributes for the buy transaction. This means that the underlying functionality for completing the purchasing of a ticket (or tickets) is handled by this object. As per the original design and architecture, this class is utilized whenever the buy transaction is performed.

### Requirements Addressed

The requirements met are with respect to processing transactions containing the event title, seller username, and updating the values in the available tickets file, and current user accounts files relevant to the completed purchase. Constraints regarding purchases and credit underflow/overflow limits are addressed.

10, 10.1, 10.2, 10.2.1, 10.2.2, 10.3, 10.3.1, 10.3.2, 10.4, 10.4.1, 10.4.2, 10.5, 10.6, 10.6.1, 10.7, 10.8, 10.9

## Create Class

Contains the methods and attributes for the create transaction. This means that this object is dedicated to facilitating the process for creating users. Per the original design, this class is utilized whenever the create transaction is performed.

### Requirements Addressed

The requirements met are with respect to processing transactions containing the username and account type for the new user, and updating the current user accounts file respectively.

7, 7.1, 7.1.1, 7.1.2, 7.1.3, 7.2, 7.2.1, 7.3, 7.4

## CurrentUserAccounts Class

Contains the methods and attributes for interacting with the current user accounts file. This object is designed for handling interactions with the current user accounts file on disk, storing the entries therein in memory, and interacting with them.

### Requirements Addressed

The requirements met are with respect to processing the current user accounts file, reading in the list of user accounts into memory, and ensuring that there is no corruption.

13, 13.1, 13.2, 13.2.1, 13.2.2, 13.3, 13.4, 13.4.1, 13.4.2, 13.5, 13.6, 13.7, 13.8, 13.9, 13.10, 20

## DailyTransaction Class

Contains the methods and attributes related to interacting with the daily transaction file. This class provides methods for reading the daily transaction file, merging previous files, and storing those entries in memory for processing.

### Requirements Addressed

The requirements met are with respect to reading the daily transactions stored into memory from the files stored on disk. The daily transaction file stores a list of Transaction objects which handle the individual formatting of each type of daily transaction. These transactions are applied through executing the individual transactions using each respective class.

15, 16, 17

## Delete Class

Contains the methods and attributes for the delete transaction. This class is utilized in each instance that the delete transaction takes place. Methods are provided for deleting user accounts from the current user accounts file.

### Requirements Addressed

The requirements met are with respect to processing the transactions containing the username to delete, verifying the user exists and updating the current user accounts file accordingly.

8, 8.1, 8.2, 8.2.1, 8.2.2, 8.3, 8.4, 8.5,

## Exception Classes

The exception classes store the strings used during exceptions that are thrown within various other classes throughout the program. The exception classes provide facilities for mapping exception codes to diagnostic messages. The two exception classes are the FailedConstraint and FatalError classes. Both work by extending the built-in Java Exception class, and providing additional parameters in their constructor to support the additional data required for displaying diagnostic output.

FailedConstraint exceptions are generated by transactions in circumstances where back-end constraints have been violated. Diagnostic information includes the type of error that has occurred, and the transaction that caused the constraint to be violated.

FatalError exceptions are generated by transactions or file operations which result in an illegal state of execution, and result in the termination of the back-end.

### Requirements Addressed

The exceptions classes are used by all of the classes within the project, including each transaction, in order to throw exceptions in the event that a constraint is violated or a fatal error occurs. The exception classes are used by every other class to meet all constraint validation requirements.

## Logout Class

Contains the methods and attributes for the logout transaction. This class is utilized when a user logged out and the transaction is logged in the daily transaction file.

### Requirements Addressed

The requirements met are with respect to tracking the amount of credit added per session. Logging in and logging out are operations related to the front-end. Because of this, the class has minimal functionality.

6, 6.1

## Refund Class

Contains the methods and attributes for the refund transaction. This class is used in each instance that the refund transaction is logged in the daily transaction file, and contains helper functions for processing refund amounts, and making changes to the available tickets and account credit for users.

### Requirements Addressed

The requirements met are with respect to processing the buyer and seller usernames, verifying the accounts exist, processing the credit amount to transfer, updating the values for the credit and tickets in the current user accounts file and available tickets file respectively.

11, 11.1, 11.2, 11.2.1, 11.3, 11.3.1, 11.4, 11.4.1, 11.5, 11.5.1, 11.6, 11.7

## Sell Class

Contains the methods and attributes for the sell transaction. This class is used for processing sales, and contains functions used for adjusting ticket volumes and account credits based on sales.

### Requirements Addressed

The requirements met are with respect to processing user input for the event title, ticket price, number of tickets for sale, and updating the respective values for credit and tickets in the available tickets file.

9, 9.1, 9.2, 9.2.1, 9.2.2, 9.3, 9.4, 9.4.1, 9.4.2, 9.5, 9.5.1, 9.5.2, 9.6, 9.7

## Ticket Class

Contains methods and attributes for data related to event tickets. Each entry in the available tickets file is stored as a ticket object.

### Requirements Addressed

The Ticket class is used indirectly by the AvailableTickets class to interact with the actual event listings, which are stored in the available tickets file. This is to fulfill the requirements also addressed by the AvailableTickets class.

## Transaction Interface

This class guarantees two methods for interacting with the transaction strings associated with transaction entries for each transaction, and executing the transactions themselves.

### Requirements Addressed

The transaction interface does not directly address any requirements, but is an architectural choice which is designed to ensure that all transactions within the back-end behave similarly, and provisions for polymorphism between the transaction classes.

## User Class

Contains methods and attributes for data related to users. Each user is stored within the current user accounts file, and thus the CurrentUserAccounts collection. This class also provides methods for interacting with user attributes.

### Requirements Addressed

The User class is used indirectly by the CurrentUserAccounts class to process the current user accounts file and fulfill the requirements met by the CurrentUserAccounts class.

## Validate Class

This class contains methods for validating input from the various files interacted with by the program. These methods ensure that input and output is compliant with requirements, and the files are not corrupted. FatalError exceptions can be thrown as a result of failed validations.

### Requirements Addressed

The Validate class is used indirectly by nearly every class to perform basic input validation in order to meet all of the data and input validation requirements.