Ticket System Design Document

By Nihil Corp.

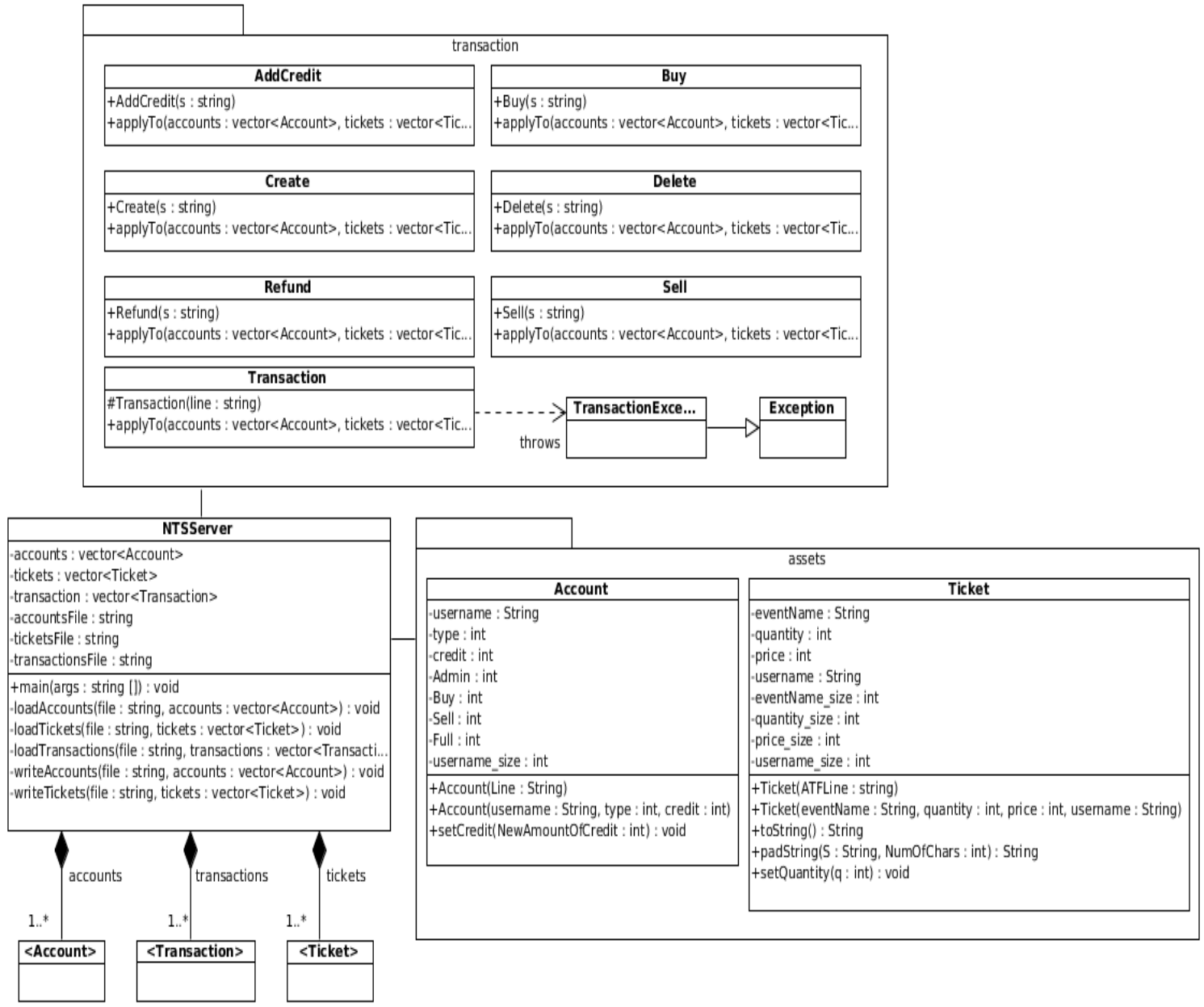
Wesley Unwin	100370215
Kalev Kalda Sikes	100425828
Bryce Benn	100395647
Ridhwaan Shakeel	100425724

1.0 Overview

The following document details the design of the Nihil Ticket System back-end prototype. In this document is a class diagram representing the object oriented structure of the ticket system, as well as descriptions explaining class interaction and intentions of each methods and variables in the various java classes of the ticket system.

The back end system fulfils the role of processing the transaction files from the various front end systems, collected into a merged transaction file, and acts as an overnight batch processor that performs the transaction operations on the system's user accounts and available tickets Upon completion, the back end outputs, an updated new users file and new tickets file.

2.0 Class Diagram



The NTSServer class contains the main method, the entry point to the entire application. Account and Ticket classes are packaged in assets, and each class that represents each six different transaction (extend from the abstract Transaction class) is packaged in the package Transaction.

(*Please see nts_server.png and nts_server.vpp for a better view of the UML class diagram)

3.0 Class Specifications

3.1 The NTSServer Class

The NTSServer (Nihil Ticket System Server class) class defined in the NTSServer.java source file, is the main class of the back end of the ticket system and contains the main method, defining the entry point of the application. This class resides in the default package.

It is important to note the local variables of the ***main method define several key, application wide variables***, that are passed to various objects of the system as needed for further processing. The following list introduces these significant variables:

3.1.1 Main Method Variables:

- **accounts : vector<Account>**
  This vector contains the currently loaded set of Account objects representing the currently available sset of user accounts in the system. These will be updated and modified throughout processing of the transactions from the merged transaction file.
- **tickets : vector<Ticket>**
  This vector contains the currently loaded set of Ticket objects, representing all tickets currently available for sale in the system. These will also be updated and modified throughout processing of the transactions.
- **transaction : vector<Transaction>**
  The transaction vector holds a list of all transactions issued by the front end clients. These transactions will be applied to (see applyTo(...) in the Transaction class) the system's vector of account and ticket classes.
- **accountsFile : string**
  This local string variable stores the directory path and filename of the current user accounts .cua file being used by the system, and is used in loading from or writing user accounts data to the file.
- **ticketsFile : string**
  This local string variable stores the directory path and filename of the current available tickets file (.atf) being used by the system, and is used in loading from or writing ticket data to the file.
- **transactionsFile : string**
  this local string variable (in the ticket system main class) stores the directory path and filename of the daily transaction .dtf file, and is used in loading from or writing transaction data to the file

3.1.2 NTSServer *Class Methods:*

Function Signature	Description of Intention
void main(String[] args)	The main function of the back end system class has processing and maintenance operations. It basically receives a set of arguments and vectors uses to try to load accounts, tickets and transactions batch data and write back to the system files
void loadAccounts(String file, Vector<Account> accounts)	The loadAccounts method is intended for opening and reading each line of the user accounts file into a vector based on a given filename string as a means of getting user accounts data
void loadTickets(String file, Vector<Ticket> tickets)	The loadTickets method attempts to open and read each line of the available tickets file into a vector based on a given filename string as a means of getting available tickets data
void loadTransactions(String file, Vector<Transaction> transactions)	The loadTransactions method opens the daily transaction file based on a filename string and reads each line into a vector in order to get a list of transactions data
void writeAccounts(String file, Vector<Account> accounts)	This method is used to write the list of user accounts data in the accounts vector to the specified current user accounts file
void writeTickets(String file, Vector<Ticket> tickets)	This method is used to write the list of user accounts data in the tickets vector to the specified available tickets file

3.2 The Account Class

An Account object represents a ticket system user, with a unique username, an account with a specified account type, amount of credit, and username. The Account class constructor works with a single line string parameter input received from each line in the current user accounts (cua) file.

3.2.1 Account *Class Variables:*

Variable Declaration:	Description & Purpose:
public String username;	String identifying the unique user of the account.
public int type;	The account type of the user, dictating the permissions and ability of the user account. This will be set to one of the constants defined below (Admin, Buy, Sell, or Full)
public int credit;	Amount of credit currently owned by the user, expressed as an integer amount of CENTS.
public static final int Admin = 0;	Constant representing the Admin account type.
public static final int Buy = 1;	Constant representing the Buy Standard account type.
public static final int Sell = 2;	Constant representing the Sell Standard account type.
public static final int Full = 3;	Constant representing the Full Standard account type.
public static final int username_size = 15;	Size of the username field (from a UAC file) in characters.

3.2.2 Account *Class Methods:*

Function Signature	Description of Intention
public Account(String Line) throws DataFormatException	Constructs and sets up an account object using a line of text from the current user accounts file.

public Account(String username, int type, int credit)	Constructs a new account object with the specified parameters. This is used by the Create transaction.

3.3 The Ticket Class

Ticket objects represent groups of tickets to some event, from some given user (the seller) that will be made available for sale.

3.3.1 Ticket *Class Variables:*

Variable Declaration:	Description & Purpose:
public String eventName;	String containing the event name to which the tickets of this ticket object are being sold to.
public int quantity;	The number of tickets being sold. This will kept in the range of 0 to 999.
public int price;	The price per ticket, expressed in CENTS.
public String username;	String identifying the user who is selling these tickets (the seller).
public static final int eventName_size = 19; public static final int quantity_size = 3; public static final int price_size = 6; public static final int username_size = 15;	These constants identify the lengths, in characters of the various fields in the line of text that originated from an available tickets file, used to construct the ticket.

3.3.2 Ticket *Class Methods:*

Function Signature	Description of Intention
public Ticket(String ATFLine) throws DataFormatException	Constructs a new Ticket object, from the given line of text, from an available tickets file. This method throws a DataFormatException if the line of text does not

	conform to the required format.
public Ticket(String eventName, int quantity, int price, String username)	Constructs a ticket object with the specified parameters, representing a group of 'quantity' number of tickets to some event, being sold from the specified seller (username)

3.4 The Transaction Class

The transaction class serves as an abstract class for all the specific transaction subclasses. This class is used for both structural purpose, and to enforce the required commonality between all concrete transaction classes. There are no variables currently defined in this class.

3.4.1 Transaction *Class Methods:*

Function Signature	Description of Intention
public abstract void applyTo (Vector<Account> accounts, Vector<Ticket> tickets) throws TransactionException;	Abstract method which will be implemented by the transaction sub-classes, defining how the transaction will be applied to the given set of user accounts and tickets passed as vectors. Transaction-specific error checking will also be performed to ensure the transaction is valid. Furthermore the system will be checked afterwards to ensure that all constraints are met.

3.5 The AddCredit Class

The AddCredit Class is a subclass of the transaction class. An instance of this class represents a particular AddCredit transaction to be applied against the systems current user accounts and available tickets.

3.5.1 AddCredit *Class Variables:*

Variable Declaration:	Description & Purpose:
public static final int code = 6	Code that identifies this transaction type, used by this server.
public static final int username_size = 15	Code that keeps track of the size of the username's length
String username	Code that grabs the username from the string that's passed to the method
int credit	Code that grabs the credit amount that is passed to the method

3.5.2 AddCredit *Class Methods:*

Function Signature	Description of Intention
public AddCredit(String s) throws DataFormatException	Constructs an AddCredit object, representing an AddCredit transaction to be applied, by getting the necessary parameters from the given line of text (s) from the merged transaction file. Note that the transaction code (2 digits/characters) and the space immediately after will have been stripped from s, before it is passed to this method. This method will throw a DataFormatException if the given string (s) does not conform to the required format.

public void applyTo(Vector<Account> accounts, Vector<Ticket> tickets) throws TransactionException	Applies the necessary operations to carry out this transaction to the given vectors containing the system's current set of user accounts and available tickets. Error checking will be performed to ensure the transaction is valid prior its application, and that the system will be in a valid state (meets back end constraints) upon complete application of the transaction.

3.6 The Buy Class

The Buy Class is a subclass of the transaction class. An instance of this class represents a particular Buy transaction to be applied against the systems current user accounts and available tickets.

3.6.1 Buy *Class Variables:*

Variable Declaration:	Description & Purpose:
public static final int code = 3;	Code that identifies this transaction type, used by the server to identify the transaction type.
public String eventName;	String containing event name of tickets
public String seller;	String containing user name of the seller to attempt to obtain the tickets from.
public int numTickets;	Number of tickets to purchase. (1-999)
public static final int eventName_size = 19; public static final int quantity_size = 3; public static final int price_size = 6; public static final int username_size = 15	String lengths of the various fields in the line of text from the ATF.

3.6.2 Buy *Class Methods:*

Function Signature	Description of Intention
public Buy(String s) throws DataFormatException	Constructs a Buy object, representing an Buy transaction to be applied, by getting the necessary parameters from the given line of text (s) from the merged transaction file. Note that the transaction code (2 digits/characters) and the space immediately after will have been stripped from s, before it is passed to this method.

	This method will throw a DataFormatException if the given string (s) does not conform to the required format.
public void Buy(Vector<Account> accounts, Vector<Ticket> tickets) throws TransactionException	Applies the necessary operations to carry out this transaction to the given vectors containing the system's current set of user accounts and available tickets. Error checking will be performed to ensure the transaction is valid prior its application, and that the system will be in a valid state (meets back end constraints) upon complete application of the transaction.

3.7 The Create Class

The Create Class is a subclass of the transaction class. An instance of this class represents a particular Create transaction to be applied against the systems current user accounts and available tickets.

3.7.1 Create *Class Variables:*

Variable Declaration:	Description & Purpose:
public static final int code = 1;	This integer denotes the integer code of the Create transaction and is useful in class operations
public static final int username_size = 15; public static final int credit_size = 6; public static final int usertype_size = 2;	These integer variables define the field size values in the create transaction line format and are used while reading create transaction lines and to extract substrings
public String username;	The username variable part of the Create transaction object stores the Create transaction parameter value of the username of new account to be created
public int type;	The type variable part of the Create transaction object stores the parameter value of the usertype index for the specified user to be created
public int credit;	The credit variable part of the Create transaction object stores the Create transaction parameter value of initial account balance in integer cents for the user account to be created

3.7.2 Create *Class Methods:*

Function Signature	Description of Intention

public Create(String s) throws DataFormatException	Constructs a Create object, representing an Create transaction to be applied, by getting the necessary parameters from the given line of text (s) from the merged transaction file. Note that the transaction code (2 digits/characters) and the space immediately after will have been stripped from s, before it is passed to this method. This method will throw a DataFormatException if the given string (s) does not conform to the required format.
public void Create(Vector<Account> accounts, Vector<Ticket> tickets) throws TransactionException	Applies the necessary operations to carry out this transaction to the given vectors containing the system's current set of user accounts and available tickets. Error checking will be performed to ensure the transaction is valid prior its application, and that the system will be in a valid state (meets back end constraints) upon complete application of the transaction.

3.8 The Delete Class

The Delete Class is a subclass of the transaction class. An instance of this class represents a particular Delete transaction to be applied against the systems current user accounts and available tickets.

3.8.1 Delete *Class Variables:*

Variable Declaration:	Description & Purpose:
public static final int code = 2;	This integer denotes the integer code of the Delete transaction and is useful in class operations
public static final int username_size = 15; public static final int credit_size = 6; public static final int usertype_size = 2;	These integer variables define the field size values in the create transaction line format and are used while reading create transaction lines and to extract substrings
public String username;	The username variable is one of the required arguments of the Delete transaction object. It stores the string parameter value of the username of the account to be deleted
public int type;	The type variable is one of the required arguments of the Delete transaction object. It stores the usertype index for the specified user to be deleted
public int credit;	The credit variable part of a required argument the Create transaction object the holds the account balance value in integer cents of the user to be deleted

3.8.2 Delete *Class Methods:*

Function Signature	Description of Intention

public Delete(String s) throws DataFormatException	Constructs an AddCredit object, representing an AddCredit transaction to be applied, by getting the necessary parameters from the given line of text (s) from the merged transaction file. Note that the transaction code (2 digits/characters) and the space immediately after will have been stripped from s, before it is passed to this method. This method will throw a DataFormatException if the given string (s) does not conform to the required format.
public void applyTo(Vector<Account> accounts, Vector<Ticket> tickets) throws TransactionException	The applyTo operation first builds a new account class object with extracted transaction line fields as arguments, uses the username to find the index of the account in the accounts vector list. If the index is found, the account is removed at the found index and the next accounts are shifted accordingly. Error checking will be performed to ensure the delete transaction is formatted and its field attributes are valid prior

3.9 The Refund Class

The Refund Class is a subclass of the transaction class. An instance of this class represents a particular Refund transaction to be applied against the systems current user accounts and available tickets.

3.9.1 Refund *Class Variables:*

Variable Declaration:	Description & Purpose:
public static final int code = 5	Code that identifies this transaction type, used by this server.
public static final int username_size = 15	Code that keeps track of the size of the username's length
String buyerUsername	Code that grabs the username of the buyer from the string that is past to it
String sellerUsername	Code that grabs the username of the seller from the string that is past to it
int refund	Code that grabs the refund amount from the string that is past to it

3.9.2 Refund *Class Methods:*

Function Signature	Description of Intention
public AddCredit(String s) throws DataFormatException	Constructs an Refund object, representing an Refund transaction to be applied, by getting the necessary parameters from the given line of text (s) from the merged transaction file. Note that the transaction code (2 digits/characters) and the space immediately after will have been stripped from s, before it is passed to this method.

	This method will throw a DataFormatException if the given string (s) does not conform to the required format.
public void applyTo(Vector<Account> accounts, Vector<Ticket> tickets) throws TransactionException	Applies the necessary operations to carry out this transaction to the given vectors containing the system's current set of user accounts and available tickets. Error checking will be performed to ensure the transaction is valid prior its application, and that the system will be in a valid state (meets back end constraints) upon complete application of the transaction.

3.10 The Sell Class

The Sell Class is a subclass of the transaction class. An instance of this class represents a particular Sell transaction to be applied against the systems current user accounts and available tickets.

3.10.1 Sell *Class Variables:*

Variable Declaration:	Description & Purpose:
public static final int code = 4;	Code that identifies the transaction type of this Sell object.
public String eventName;	String containing the event name of the tickets being sold.
public String seller;	String containing the username of seller.
public int numTickets;	Number of tickets being sold (1-999)
public int ticketPrice;	Price per ticket, expressed in cents.
public static final int eventName_size = 19; public static final int quantity_size = 3; public static final int price_size = 6; public static final int username_size = 15;	String lengths for the various fields within the line of text from the ATF.

3.10.2 Sell *Class Methods:*

Function Signature	Description of Intention
public AddCredit(String s) throws DataFormatException	Constructs an Sell object, representing an Sell transaction to be applied, by getting the necessary parameters from the given line of text (s) from the merged transaction file. Note that the transaction code (2 digits/characters) and the space immediately after will have been stripped

	from s, before it is passed to this method. This method will throw a DataFormatException if the given string (s) does not conform to the required format.
public void applyTo(Vector<Account> accounts, Vector<Ticket> tickets) throws TransactionException	Applies the necessary operations to carry out this transaction to the given vectors containing the system's current set of user accounts and available tickets. Error checking will be performed to ensure the transaction is valid prior its application, and that the system will be in a valid state (meets back end constraints) upon complete application of the transaction.

3.11 TransactionException Class

This class is a subclass of the exception class, which is used to represent an exception caused when a transaction is unable to be applied, due to invalid parameters, or semantic errors. This errors will be caught by the server, handled, and outputted to the user.