JAV745 Winter 2020: Project 2 (v. 1.0) - (19%)

Dr. Eden Burton
School of ICT, Seneca College of Applied Arts and Technology
Winter 2020

Due Friday April 10, 2020 - 11:30 pm

Instructions

Please read the instructions carefully and follow the naming conventions specified for each question. Solutions must be submitted in the Blackboard Dropbox created for project 2.

The deliverables will be placed in two packages named ***jav745.client* and *jav745.server*** respectively. The submission itself shall be in a jar files for each process (called ***jav745p2client.jar* and *jav745p2server.jar*)** which contains both source (*.java) and bytecode (*class) files.

Your solution should be well documented using the JavaDoc utility to describe both your interface and your solution design. Ensure that your read the requirements and deliverables carefully. Note that the deadline is strictly enforced. The system tracks the exact time that submissions are uploaded. 10% will be deducted for each late day.

Your task is to build concert ticket sales processing system. User will have the opportunity to place orders for concert tickets. The system shall have client and server components. The requirements are as follows.

Server

The solution must support multiple venues and multiple concerts. Each venue will have a unique seating plan. For simplicity, we will not track individual seats, but types of seats. Each seat type will have a certain number of seats allocated to it.

For example, the "Toronto Concert Hall" may have 100 floor seats, 500 level 1 seats, and 3000 level 2 seats while the "Sony Centre" may have 1000 gold seats and 7000 silver seats.

Each concert will be held on a particular date and at a particular venue. A concert booked at a particular venue can (but does not have to) use all the seats in a venue. A concert promoter may decide to only use 50% of the seats in the chosen venue. Each seat type is assigned a single price. The number of seats sold shall be tracked by the system.

Information about venues, concerts, pricing and initial account balances must be uploaded to the server via a text file loaded at program startup.

The server accepts tickets orders and processes them in the order received. It shall be able to process requests from multiple clients concurrently. The result of each transaction to be logged in a file, successful transactions should indicate a client identifier, current time and order details. Failed transactions should client identifier, current time and the reason of the failure.

Client

A user's ticket order will be encapsulated in a request to the server process. It shall contain the concert, venue and date of the tickets desired. Also included will be the number of each class of seat desired. Each request will attempt to obtain tickets for only one concert date.

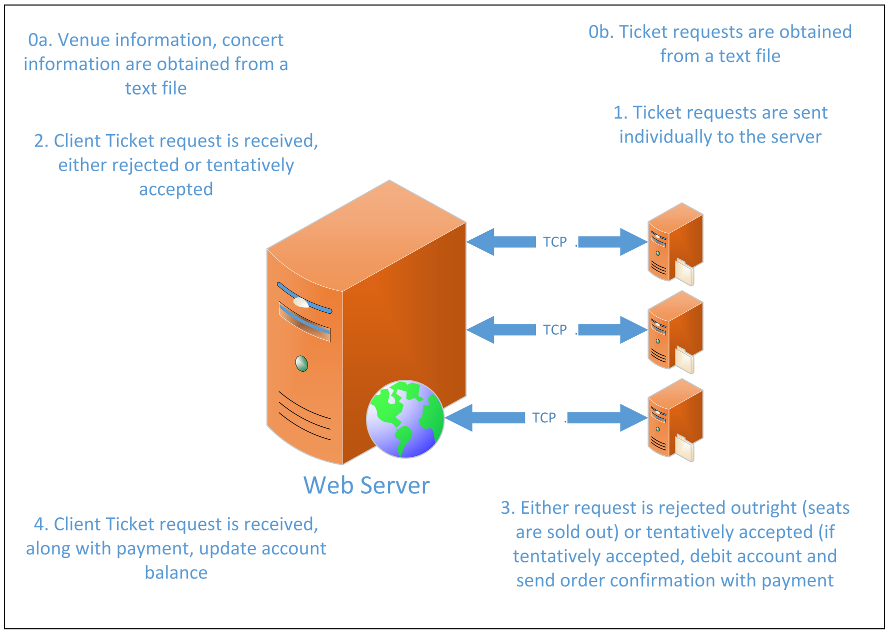
0a. Venue information, concert information are obtained from a text file
0b. Ticket requests are obtained from a text file
1. Ticket requests are sent individually to the server
2. Client Ticket request is received, either rejected or tentatively accepted
TCP
TCP
TCP
Web Server
3. Either request is rejected outright (seats are sold out) or tentatively accepted (if tentatively accepted, debit account and send order confirmation with payment
4. Client Ticket request is received, along with payment, update account balance

Figure 1: Concert Purchasing Application Architecture

An example of an order... *"Drake Live, Madison Square Garden, 2 Gold tickets, 5 Silver tickets"*

Multiple requests can be grouped together in a single file. After the file is loaded into the program, ticket requests are sent to the server one at a time. The client application will process all files in a configurable directory. There is a single account (initialized at program startup) which holds money required to pay for the requests. This account is debited by the appropriate amount when a tentatively confirmed order is transmitted to the server. An order is abandoned if there is not enough money available to pay for it. The result of each transaction to be logged in a file, successful transactions should indicate a current time and order details. Failed transactions should note the current time and the reason of the failure.

Protocol

Upon receiving a client request to purchase a quantity of tickets for a particular concert, the server checks to see if such tickets are available. If the quantity of tickets is not available, then the server response indicates this. Otherwise (if there is availablity), the available tickets are removed from the pool and reserved for the client pending payment. The user then sends a subsequent request with payment. The server deposits the payment into a payment account and a response is sent to the user confirming ownership of the tickets. If payment is not received in 60 seconds, then the order is cancelled.

The way that this information is encoded the messages between machines is a design decision that you are to make.

Deliverables

- an executable jar for the web server. The command `java -jar jav745p2server.jar` <port number> <input file(s)> <starting balance> should start the application.
- an executable jar for the user application. The command `java -jar jav745p2client.jar` <server address> <server port> <configuration file> <starting balance> should start the application. Alternatively, you can use a configuration file.
- a document which explains ...
  - instructions on how to run your program

  - describe the structure of both your user and server input files so someone else could build these files without your assistance
  - a detailed description of your client-server protocol

- samples of user and server input files

Note that the jar files submitted must and contain both source (.java) and bytecode (*class) files. Ensure that you test check this before you submit.*

Marking Criteria

- functionality ...Does the system actually work? Does it satisfy the requirements? Can it run using submitted information and instructions?
  1. *web server.* Does it successfully listen on a configurable port when the command specified above runs? Can it accept requests? Does it generate well formed responses?
  2. *client.* Does your application recover from invalid/missing file information? Are suitable error messages displayed or written to a log file when problems occur?
- does your documentation provide enough detail so that someone can run/test your solution without assistance? Do the internal comments describe the system in enough detail that another programmer can understand it? Does running the JavaDocs utility provide meaningful documentation for developers?
- were submission instructions followed?