UNIVERSITY OF WESTMINSTER

INFORMATICS INSTITUTE OF TECHNOLOGY

University of Westminster

Object Oriented Programming

5COSC019C.1

Student Name : K.K.C.S.S.Bandara

UoW ID : w2052748

IIT ID : 20221007

Acknowledgement

I would like to express my gratitude to Mr. Gugsi for his assistance and encouragement during the Real-Time Event Ticketing System development process. His guidance could help us overcome a lot of the obstacles we encountered. I also want to use this opportunity to express my gratitude to our module team for their hard work, which made the project a success. Your assistance and support are much appreciated. I appreciate that you provide a welcoming learning environment that fosters teamwork.

Abstract

This extensive software program uses the producer-consumer pattern to replicate a dynamic ticketing environment based on sophisticated object-oriented programming (OOP) principles. Resource management and data integrity in a multi-threaded world are ensured by vendor and customer checks and balances on both ends of the process, if such a system is in place. Users can choose system settings such as the maximum capacity, release rate, and total tickets using the JavaFX-powered application. In order to provide easy operation and great safety from race circumstances, all of its duties are shared utilising synchronisation mechanisms. Additionally, it has a system activity record and error handling mechanism for stability. A realistic simulation of real-world ticketing issues is integrated into an intuitive interface that shows real-time ticket availability and system status.

Table of Contents

Problem Statement.

The goal of this project was to create a real-time event ticketing system that would enable customers to set up and model ticket purchases using a user-friendly JavaFX interface. They let users enter information like the total amount of tickets, the speed at which they are released, the speed at which clients may purchase tickets, the maximum quantity they can purchase, and the number of tickets they wish to purchase. Both our maximum number of tickets which must exceed the entire number of available tickets and any negative input value will be examined andrefused.

Users can also save their configurations and run simulations at a later time. The system will then display the ticket sales process, including how merchants release tickets and how customers attempt to purchase them in real time. Users can ask to load prior results or start new configurations, but the interface will walk them through the process because they can only interact with our interface and not the simulation setup. When considered as a whole, the system aims to be an easy and efficient way to sell event tickets.

Test Cases

Test Case ID	Scenario	Test Steps	Expected Result	Outcome
GUI				
1	Handling ticket purchases	1. Configure the customer retrieval rate 2. Send multiple customer ticket purchase requests concurrently. 3.Monitor ticket availability	No race conditions, correct ticket count after all requests are processed.	Pass
2	Handling the adding of tickets	1. Configure the vendor ticket release rate 2. Send multiple vendor ticket adding requests concurrently. 3. Monitor ticket availability	No race conditions, correct remaining ticket count after all ticket-adding requests.	Pass
3	Reaching the maximum capacity	1. Send vendor ticket adding requests until ticket pool limit is reached. 2. Monitor system response when limit is reached.	Ticket adding should stop, message should inform vendor the limit has been reached	Pass
4	Reaching the maximum capacity of the event	1. Send vendor and customer requests until the total tickets released by vendors reach the max ticket capacity. 2. Monitor system response.	Ticket adding should stop, vendors cannot add more tickets once the max ticket capacity is reached.	Pass
5	Real-time update of the available ticket count	1. Send several vendor and customer requests. 2. Monitor if available tickets count updates in real-time without refreshing.	Available ticket count should change in real-time without needing a refresh.	Pass

6	Input Validation	1. Insert a negative number when creating an event, adding tickets, or purchasing tickets. 2. Monitor system response	The system should display a customized error message when invalid input is entered.	Pass
7	Thread synchronization	1. Send vendor ticket adding requests. 2. Send customer ticket purchase requests at the same time. 3. Monitor the logs	No race conditions or deadlocks. The system should handle concurrent requests from both vendors and customers correctly.	Pass
8	Error handling	1. Simulate a database failure. 2. Send a ticket purchase or ticket adding request. 3. Monitor system's response	System should not crash, should display a customized error message instead.	Pass

CLI				
11	Starting the ticketing process	1. Enter the start command	Vendor and customer threads should be started, and the ticketing process should be running..	Pass
12	Trying to start when the process is already started.	1. Enter "Start" when the process is already running.	A message should display saying the ticketing process is already running.	Pass
13	Stopping the ticket process	1. Enter "stop" to stop the ticketing process..	The ticketing process should stop, and a message should display saying the system has stopped.	Pass
14	Exiting the system	1. Enter "Stop" to stop the ticketing process. 2. Enter "No" when asked to configure again.	The system should exit with a message displayed.	Pass

15	Input validation	1. Enter negative values or invalid input.	A customized error message should be shown, and the user should be prompted to enter the correct value.	Pass
16	Checking for proper thread synchronization	1. Configure the system. 2. Start the ticketing process. 3. Monitor the system logs.	There should be no race conditions. Vendors and customers should be able to add and purchase tickets concurrently.	Pass

Class Diagram

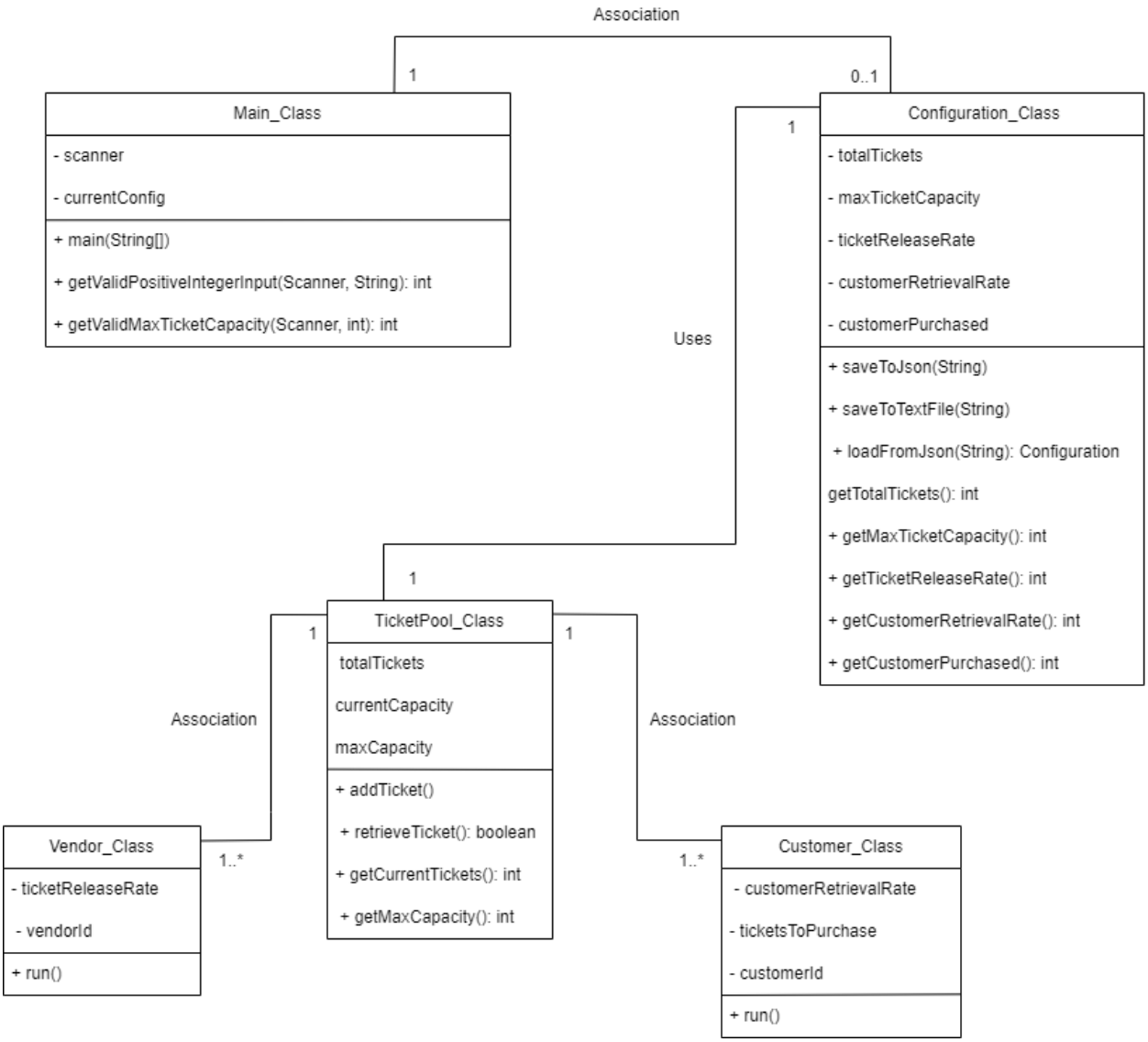

Figure 1 Class Diagram

Sequance Diagram

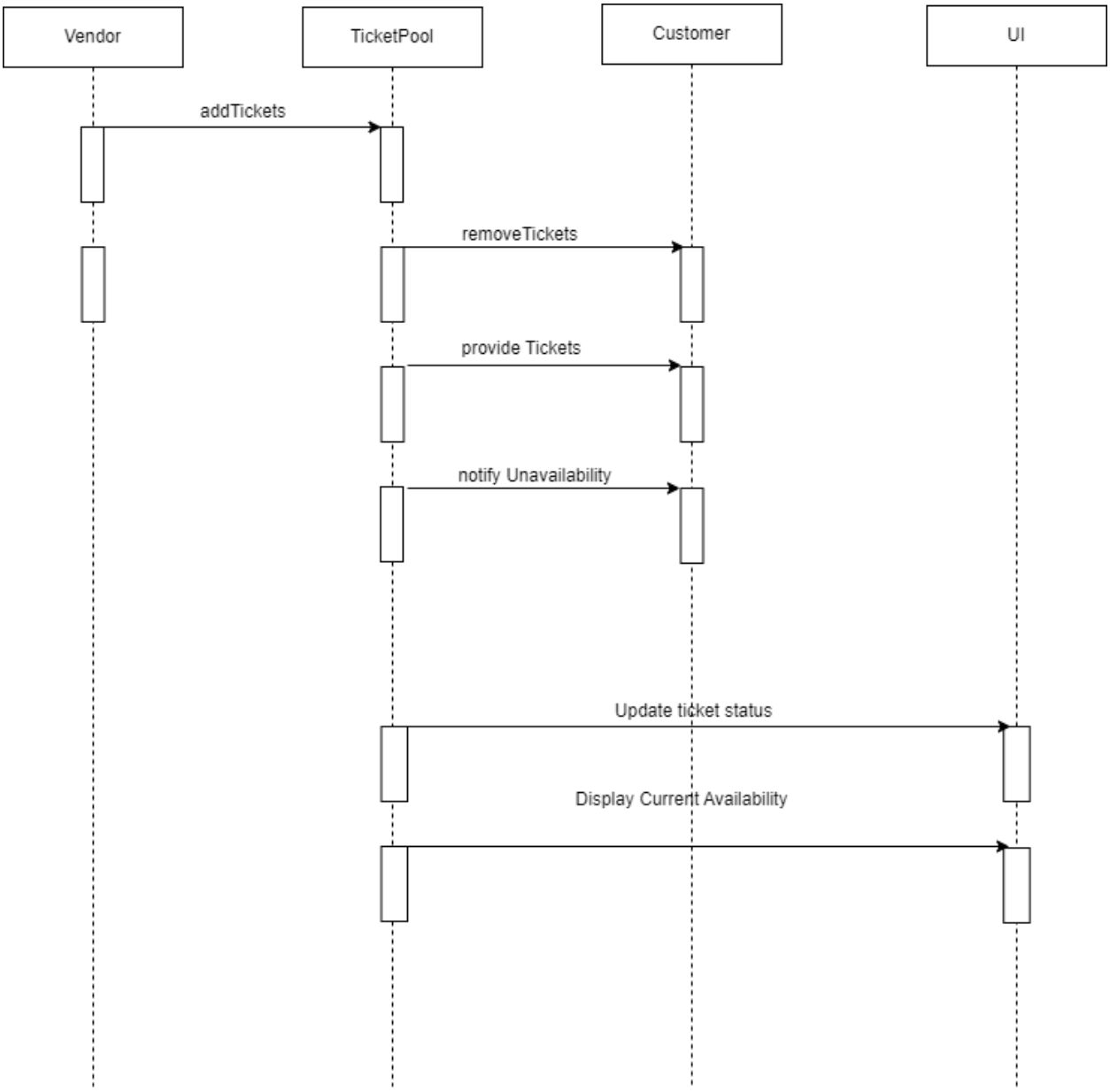

Figure 2 Sequance Diagram