This README file explains how to use the TicketService Application

---- BUILD AND RUN APPLICATION ON THE COMMAND LINE ----

* Firstly, download and add maven to classpath
* Go to the TicketService location in the command line

**C:\{classpath\*}\TicketService>**

* TicketService should have the maven's pom file. The pom file has all the dependencies and build configurations set up. Just run the following commond

**C:\{classpath\*}\TicketService>mvn clean install**

* The above command builds successfully. In the target directory a executeable jar file with name "TicketService-0.0.1" is created.
* To execute the jar file give the following command along with command line variable {number of customer threads to be run}

**C:\{classpath\*}\TicketService\target>javac -jar "TicketService-0.0.1.jar 10**

* The above command creates 10 customer threads to serve the customers

---- BUILD AND RUN JUNIT TEST CASES ON THE COMMAND LINE ----

* Go to the TicketService location in the command line

**C:\{classpath\*}\TicketService>**

* To build the Junit give the following commands

**C:\{classpath\*}\TicketService>mvn test**

* Run the test class, in this project TicketServiceTest is the test class

**C:\{classpath\*}\TicketService>mvn -Dtest=TicketServiceTest test**

* All the test cases should run successfully with Failures: 0

--- Assumptions made ---

* To find and hold best seats, the minLevel and maxLevel parameters are integer values representing the Venue Levels. Orchestra – 1, Main – 2, BalconyOne – 3, BalconyTwo – 4
* SeatHold object returned, will expire in 60 seconds if it isn’t reserved.
* Cleaning up of expired thread is done by a daemon thread for every 20 seconds.

--- Classes summary ---

* **MainService:** The starting point of execution with main method. It creates fixed number of customer threads and CacheCleanUpThread.
* **CustomerThread:** An instance of the customer.
* **CustomerService:** It provides options to every customer thread to view, hold and reserve seats.
* **TicketServiceImpl:** An implementation of TicketService, serves the available ticket count, holds and reserves tickets.
* **VenueLevels:** **Orchestra, Main, BalconyOne, BalconyTwo** are model classes representing the venue levels and provides seats count, seats to be held and reserved.
* **SeatHold:** model object representing held seats by a particular customer. It has a field to expire the seats held.
* **Seat:** model object to uniquely represent a seat in a particular Venue Level.
* **CacheCleanUpThread:** Daemon thread cleans up expired held seats for every 20 seconds.
* **SeatHoldCache:** Maintains record of the cached SeatHolds.
* **VenueLevelCache:** Maintains details about VenueLevels.

--- Concurrency problems and locks ---

* Every customer thread has to provide input about the service from console/command prompt, here concurrent access to common resource namely Standard Input (stdin) may lead to data ambiguity, hence implemented lock on stdin, where only one thread has hold on stdin.
* Every customer thread access common resources like Venue Levels and SeatHoldCache for data manipulation, to prevent data inconsistency, locks are used.