Read Me - Data Migration Project

Background

During week 5 of the Sparta global course the main points were:

- input and output streams within java
- Concurrency
- Establishing connections to external MySQL servers
- Queries on MySQL servers from Java
- DAO's (Data Access Objects)
- DTO's (Data Transfer Objects)

This project was aimed at utilising the skills that were developed this week within the course. Initially I was provided with a CSV file of employees that needed to be entered into a database.

Requirements

The program needed to take the CSV file and read the values and store them in a useful format within the java application. It then needed to send these entries to a local database, as well as checking and logging failed entries for the client to action on. Once this had been done in a single threaded application multiple threads should be introduced to speed up the process of transferring the entries across to the database.

User Guide

Upon starting the application, the user is asked for an input to decide the file that is to be copied across to the database. Once this is done the program reads the selected csv using a buffered reader and creates a list of the rows of the CSV File. The program then creates the employee table and creates employee DTOs for each of the entries contained within the CSV.

The program then creates a list checking through the additions to this list one at a time to ensure that there are no duplicate primary keys contained within the CSV. To reduce the number of connections the program needs to make to MySQL the employee DTO's are then placed into batches of 1000 before being passed to a thread to be inputted into the database through an instance of the employee DAO.

Once the program has finished uploading the entries to the database the program prints the failed entries into a CSV file called failed entries. After this is complete the program then prints the time taken in seconds to: read, check, package, upload, and output to the CSV file.

Performance Testing

After completing the single threaded and multi-threaded versions of the application I carried out performance testing of the application to further demonstrate the reason to use multi-threading within an application such as this.

Below is a table that details the results of the performance testing where the small test was 10000 entries with some entries with duplicate employee IDs and the Large test consisted of 65499 distinct employees.

	Small	Large
Single Thread	91.9 Seconds	307.4 Seconds
Multi Thread	3.46 Seconds	13.5 Seconds

As can be seen from the data presented in the table above the multi-threaded version of the application was considerably faster than the single threaded version in both scenarios (nearly 30 times faster in the small and 23 times faster in the large). This performance benefit is because in the multi-threaded version the next thread can start preparing its data to be sent off and start the connection while the previous thread is still uploading as opposed to the single threaded version which is entirely sequential.