55-500213 Algorithms and Data Structures II Assignment

Task 1: ADS Implementations: Telephone Banking System

Due in	Thursday, 14th January 2021, at 15:00
Submission Method	Online Submission via Blackboard
Marks available	100 (40% of the module assessment)

This assignment aims to assess your practical understanding of fundamental algorithms and data structures through a specific task – operating a telephone banking system.

A Java skeleton structure is provided for this assignment. In the skeleton structure, frontend functions such as file I/O and user interactions have been fully developed. Your task is to design, develop and test the requested underlying algorithms and data structures for the telephone banking system. You also need to explain the time complexity of those algorithms by using Big-O notation.

Following learning outcomes are assessed in this task:

- Characterise algorithms and describe their attributes using appropriate metrics;
- Design novel algorithms;
- Construct and manipulate the complex data structures required of many algorithms.

The Java files in the IntelliJ project provide the skeleton code for each class you need to implement. It is important to follow this specification as the functional correctness and method of implementations will be checked.

You will need to download the skeleton from Blackboard at "**Assessment**" → "**Task 1 ADS2 Implementations**" → "**ADS2_Task1_TelephoneBanking.zip**".

During the implementation, you must observe the following requirements:

1. While you can add extra methods and classes to help your task, the provided classes are not allowed to be deleted. Failure to preserve this minimum specification will result in a loss of marks.
2. You may use or manipulate any code already written during tutorial sessions. You need to understand and be able to defend and edit any code you submit. The code must be compatible with the skeleton structure.
3. The use of the Java Collections Classes is prohibited. You must implement all data structures and algorithms from base types, although the use of Strings is allowed. Please ask if you are not sure about whether you can use certain built-in Java classes.
4. This assignment is an individual piece of work, and your submission must be in the form of a working IntelliJ project. You may be asked to give a walkthrough of your code before a mark is awarded. During the walkthrough, you will be asked to demonstrate your understanding of the code and what it does. You will be notified by email if you need to have a walkthrough.
5. In-module retrieval is available for this assignment.

1. Introduction

This telephone banking system has two functions:

- **Queueing customers**
  Using the first-come-first-serve rule, each customer needs to wait in a queue before getting banking services. Create a queue data structure for this function. You need design and develop the underlying data structure from scratch to supporting the functionality of queues.
- **Providing banking services for customers.**

The system reads customer bank record from a CSV file (i.e. user name and current balance) during the system initialisation. You need to implement a suitable data structure to hold those records.

There are five different banking services in the system, which are “open an account”, “close the account”, “check balance”, “save money” and “withdraw money”. The requests are randomly generated. Each customer is associated with only one task. You are expected to design and develop some effective algorithms for banking services.

You are free to choose any underlying data structures for those algorithms. However, it would be best if you showed excellent awareness of the trade-off between time and space.

The basic system flow is driven by the Finite-State Machine. Figure 1 illustrates the states and transition conditions of the system.

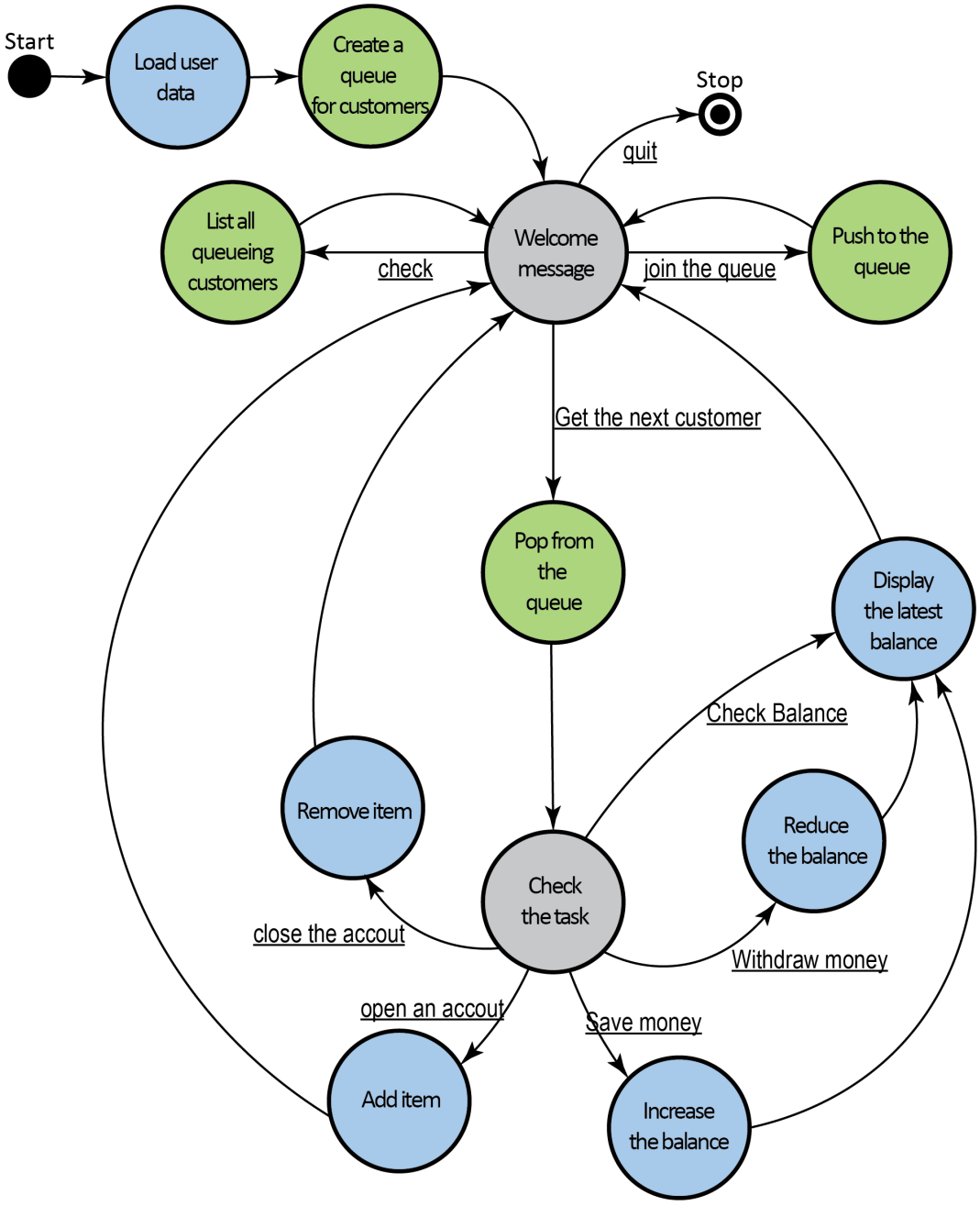

Figure 1. The state diagram of the telephone banking system.

In the diagram, the functions of **"Grey"** states are fully developed as parts of the skeleton structure (in the main `Main.java`).

The **"Green"** states contain the methods from `CustomerQueue` Class. You need to develop a structure to queue the customers.

The **"Blue"** states contain the methods from `CustomerData` Class. You need to develop a suitable underlying data structure for keeping user records and providing service for each customer.

The skeleton code is executable, but all the functions are just placeholders. You will see messages such as "`This function is not implemented`" in the placeholder. Here is where you can add your own implementation to this system.

After the development, the system should be able to load the bank record and interact with each "virtual customer". For example, you can queue a group of customers, check their requests, input their bank account, provide services, and get next customer *et al.* by choosing different options from the menu listed in the console.

During this process, the "virtual customers" are automatically created by `CustomerRequest.newRequest()` method. The method associates only one random request for each customer. The method also uses "`amountToChange`" to record a random amount of money to update the account balance in case of "open an account", or "save/withdraw money".

The bank record is a two-column data table containing customers bank account IDs and balances. An example data file includes 100 user information, named as "`BankUserDataset100.csv`", has been packed with the IntelliJ Project. In the database, the ID is a string containing two uppercase characters followed by five digital numbers such as "AA12345" and "XY11238". The balances are floating-point numbers in the range of £100.0 – £1000.0.

After the development, you may want to change to load a more extensive datasheet, "`BankUserDataset10K.csv`", which contains 10,000 user information for the system testing and profiling.

2. Grading Criteria

The work will be graded using the criteria listed below. Please also refer to the University's Generic Grade-based Assessment Structure as the general grading guidance. (See Appendix)

2.1 Design suitable data structures and algorithms (20/100)

You need to design suitable underlying algorithms and data structures in `CustomerQueue` and `CustomerData` classes. There are basic functions required from the skeleton code, which are listed in the table below:

CustomerQueue	Check if the queue is empty
	Print out all the queueing customers and their requests
	Add a new customer to the queue
	Get the customer from the queue
CustomerData	Load the data from CSV file to your chosen underlying data structure
	Add a new data point for "open an account" request
	Delete a data point for "close the account" request
	Check if an ID has been used before opening a new account
	Update the account balance
	Show the latest balance

You need to explain how you design the system in a “ADS2 Task 1 Report” and submit the report together with the code.

To get a higher grade, you should be able to:

1. **Show great awareness of time-space-trade-off**
2. **Choose suitable underlying data structure for queueing requests and storing the customer data**
3. **Design highly effective algorithms for the required functionalities.**
4. **Understanding the advantage and disadvantage of using array-based/linked-list-based data structures**
5. **Understanding the advantage and disadvantage of using iteration and recursion**

2.2 Construct and manipulate the designed algorithms and data structures (50/100)

Implement your designed algorithms and data structures to the skeleton code.

You may need to create additional methods and helper functions for the implementation. However, all the data structures and algorithms must be built from basics without using Java Collections Classes. If you are not sure whether you can use certain inbuilt Java classes or functions, please ask.

You are also highly recommended to test your code. Although there is no such thing called “bug-free” code, you still need to maintain the quality for your code. Once the quality level is defined, you need to test your code until that level before release. When you develop each function, it is suggested to break the tasks down into manageable and testable units, and to use IntelliJ debug tool, unit test tool, and console output to locate and correct code errors.

To get a higher grade, you should be able to:

1. **Implement the proposed algorithms and data structure correctly**
2. **Create robust code which does not crush due to the run-time errors**
3. **Guard the function inputs and outputs**
4. **Show a great understanding of how to write effective code in Java Programming Language**

If there are aspects of the implementations that you are unable to code, then you should leave the stub function in place and comment out the code that is causing issues to ensure the IntelliJ project compiles. The commented code will be reviewed for a portion of the marks.

2.3 Creativity (30/100)

You are encouraged to do something creative beyond the given specification. You are highly recommended to carry out some improvements for the underlying algorithms and data structures which allow the system to be more effective.

The level of extra work required to obtain full marks for the creativity should be in proportion to the rest of the assignment. If you do need to explain what you have created, you should submit this in a separate document (do not fold it into other reports – keep it separate to make it very clear what you have done).

3. Submission Process

Your assignment should be submitted electronically through the module's Blackboard site as a single ZIP file that contains all your source code and reports.

The submitted ZIP file must contain all files that allow the IntelliJ project to be opened (and run) on a campus computer (including their directory structure). Check your upload to ensure you have submitted the correct files successfully as issues will not be considered after the deadline.

Appendix: GRADE-BASED ASSESSMENT

What is it?

Marking is often subjective. In evaluating code style or the functionality of a piece of work there is no single right answer and no single wrong answer. Rather, a piece of work sites on a continuum from perfect to awful. Most academics grading work know this and know that the difference between 7/10 and 6/10 is to some degree a matter of subjective judgement.

Grade-based assessment breaks the link between the quality of each aspect of an assignment submission and a simple number. Instead work is allocated a letter grade that is later converted programmatically into a number using a spreadsheet.

How it works

Each assessment task is marked against a number of criteria that may cover aspects of the work such as coding style, presentation or software architecture. Each criterion is allocated a number of marks, so, for example, 15 marks may be available for the software architecture. Rather than assume that there is an objective difference between an architecture that is worth 11/15 and one that is worth 12/15, grade-based assessment allocates a broad letter grade to the architecture. This grade is converted to a mark through a simple arithmetic substitution. The overall mark for the assignment is the sum of the marks given to each of the criteria.

The system has grades A down to F which broadly mirror the classifications given to SHU degrees. An F grade represents a borderline fail. The highest marks (aka “First”) are split in two with the B grade covering the range 71 to 80 and the A grade covering 81 to 100.

Each grade can be modified by a +/- (eg A-, C+). The grade A+ implies that the work is as near perfect as could be expected from an undergraduate and is given only very rarely.

Grade Descriptors

The grade descriptions in the following table are generic and indicative; the university has provided grade descriptors for each level of study. These are available on Blackboard and should be used alongside this document.

Grade	Descriptor
A FIRST (Excellent)	● Exceptional work of the highest quality, demonstrating excellent knowledge and understanding, analysis, organisation, accuracy, relevance, presentation and appropriate skills. ● Where relevant the work may achieve or be close to publishable standard. ● It is difficult to suggest ways that the work could be improved/extended without introducing concepts from higher levels of study. ● There is extensive evidence of the independent investigation, learning and thought.
B FIRST (Excellent)	● Very high quality work demonstrating excellent knowledge and understanding, analysis, organisation, accuracy, relevance, presentation and appropriate skills. ● Work which may extend existing debates or interpretations. ● The work as presented is difficult to fault, but there are some obvious areas where it could be extended/improved. ● There is ample evidence of the independent investigation, learning and thought.
C UPPER SECOND (Very good)	● High quality work demonstrating good knowledge and understanding, analysis, organisation, accuracy, relevance, presentation and appropriate skills. ● The work as presented has some very minor errors, and there are some obvious areas where it could be extended/improved. ● There is some limited evidence of independent investigation and learning.
D LOWER SECOND (Good)	● Competent work, demonstrating reasonable knowledge and understanding. ● Some analysis, organisation, accuracy, relevance, presentation and appropriate skills are shown. ● The work as presented has some minor errors, but is limited in scope. ● Does not significantly extend work previously presented.
E THIRD (Sufficient)	● Work of limited quality. ● Demonstrates some relevant knowledge and understanding.
F FAIL (Insufficient)	● Work does not meet the standards required for this stage of an Honours degree. ● There may be evidence of some basic understanding of relevant concepts and techniques but this does not meet the level that was taught in class.
G FAIL (Insufficient)	● Some attempt has been made to tackle the assessment. ● The work has little or no merit.
Z	● Work of no merit or work was not submitted.