

A dark blue vertical bar runs down the left side of the slide. A blue arrow points to the right from this bar, containing the date.

11/28/2016

# Shared Corporate Bank Account Problem

CS3103 Group Project

Several thin, curved lines in shades of blue and grey originate from the bottom left and sweep upwards and to the right.

Robert Cinca, Lorenz Friedrich  
CITY UNIVERSITY OF HONG KONG

## Table of Contents

<b>Overview of Project .....</b>	<b>3</b>
<b>Our Simulation .....</b>	<b>3</b>
<b>Difficulties Encountered.....</b>	<b>4</b>
Collaboration.....	4
Efficient implementation of synchronization.....	4
Time Management.....	4
<b>What was Learnt .....</b>	<b>5</b>
<b>Results .....</b>	<b>5</b>
<b>Individual Contributions .....</b>	<b>5</b>
Robert Cinca .....	5
Lorenz Friedrich .....	6
<b>Acknowledgments .....</b>	<b>6</b>
<b>Appendix.....</b>	<b>7</b>

## Overview of Project

This year, the CS3103 Operating Systems group project involved creating a Shared Corporate Bank Account. The idea behind this problem is that multiple employees from different banks need to access their respective corporate accounts to check the balance, withdraw or deposit money. The problem is complicated by the fact that the employees cannot do this task on their own: they need bank staff to conduct the process.

There are multiple things happening at the same time but with certain restrictions: only one employee can perform withdraw or deposit on a corporate account at the same time, however multiple employees from the same company can read their corporate account (if no withdrawal/deposit is taking place). Additionally, there are only a handful of bank staff that are able to perform the task. Thus the company employees may need to wait for a bank staff to become available and the bank staff may need to wait on other bank staff to finish with a corporate account before having access to it.

Consequently, the main objective of the object is how to coordinate multiple threads together in order to achieve a relatively fast and efficient way for the employees to perform their required task. Our implementation will be discussed in the 'Our Simulation' section.

The Summary Report will then discuss the following sections:

- Difficulties encountered
- What was learnt
- Results
- Individual group member's responsibility and efforts listed in detail.

## Our Simulation

We decided to implement the project in Java with the use of Eclipse as the IDE (Integrated Development Environment). We chose Java as we were both competent with the language. Moreover, the Java library provides a vast range of built-in tools to control threads and simultaneous actions, namely through the use of concurrency utilities.

We included the following concurrent tools: cyclic barriers, priority blocking queues, and re-entrant read-write locks. While we didn't explicitly employ semaphores on their own, some of the utilities mentioned beforehand incorporate them, as well as other concurrency control mechanisms. A graphic of the cyclic barrier implementation is shown on the last page in the Appendix of this document. In the design brief, we will explain how we used these tools to handle concurrency in more detail.

## Difficulties Encountered

### Collaboration

Our initial difficulty was to find a way to collaborate on the project remotely in an easy and efficient manner. This was especially important given our circumstances: our schedules did not really match, meaning that we could work on the project at different times.

Thus we decided to use GitHub, a git repository that allows us to share and collaborate during the project. There were several reasons why we decided to use GitHub:

- The Git repository is safe from file loss, as the project is stored online.
- It is simple to revert back to previous versions of the code. There is an easy user interface that displays previous versions and who made modifications.
- Conversely, it is also simple to keep up with the most up-to-date version.
- It is easy to access the Git repository from other devices, one does not have to be on a specific device.
- The use of the branch feature permits us to code different parts of the application separately and try out new ideas without damaging the working code.
- GitHub is widely used so there is extensive support available online if issues do arise.

Our GitHub repository can be found here: <https://github.com/robertcinca/concurrencyproject>

We also used Slack as a way to communicate and share files/ideas in a free manner.

### Efficient implementation of synchronization

Another problem we faced at the start involved an efficient implementation of synchronization. This was because we had never used multiple threads in a program of this length and complexity before.

To find a solution, we broke the problem into smaller parts, drawing out possible layouts of the program and then looking online on StackOverflow and the Java Docs for possible ways to implement our solution. As a result, we found that our implementation was easiest through the use of cyclic barriers, priority blocking queues, and re-entrant read-write locks.

### Time Management

Another issue we faced was implementing an efficient way of using our time wisely so that we do not have to do too much work on any particular day. This was made more difficult by our busy and different schedules. Thus online communication and code sharing for remote collaboration was essential. To ensure that tasks do not build up too close to the final deadline we set interim deadlines throughout the project. This was also used as a way to keep an eye on the level of progress.

## What was Learnt

The first thing we learnt was the importance of having a structured approach to a project of this complexity and length of time. We learnt the importance of having good communication and collaboration tools as a way of working together on a project remotely. We also realized how important the use of intermediary deadlines were as a way of controlling the pace of our project progression.

We also implemented certain Software Development tools we have learnt at university. For instance, before even coding anything, we separated the requirements into functional and non-functional ones, using a MoSCoW-style (Must-have, Should-have, Could-have, Would-have) sorting system and other project management methodologies. We also created UML and Class diagrams to aid us with the visualization of the project. We have included some of these graphics in our Design Report.

In terms of programming, we learnt how to use threads and concurrency tools as a way to control the flow of a program that deals with simultaneous actions. To facilitate this, it was important to implement a logical structure, using classes and methods to achieve this.

## Results

The outcome of our project was a fully working solution to the Shared Corporate Bank account problem. Our program starts with a GUI interface that lets the user select which file they want to run. Once the file is selected the rest of the program is executed, using cyclic barriers to control the flow of the project. The results of the program are printed in the interface so that the user knows what the program has done.

## Individual Contributions

Although we worked together as a team, we split tasks between each other, taking into account our strengths and weaknesses. This was based on an online test we conducted called the Clifton Strengths Finder test (<http://www.strengthsfinder.com/home.aspx>). Our individual strengths and task allocation are shown below:

### Robert Cinca

After taking the Clifton Strengths Finder test, I uncovered that my top 5 skills are: achiever, competition, learner, analytical and input. Thus my primary roles for this project were:

- Team Leader and Project Manager – including task allocation
- Design and implementation of GUI interface
- Summary report
- Repository lead – creating GitHub repository and managing it

## Lorenz Friedrich

My top strengths are as follows: Futuristic, Relator, Deliberative, Adaptability and Maximizer.

Thus my primary roles for this project were:

- Design and implementation of concurrency and threading
- Design report
- Requirements Collecting – in order to find a solution to Bank problem
- Program Testing – to ensure program works

## Acknowledgments

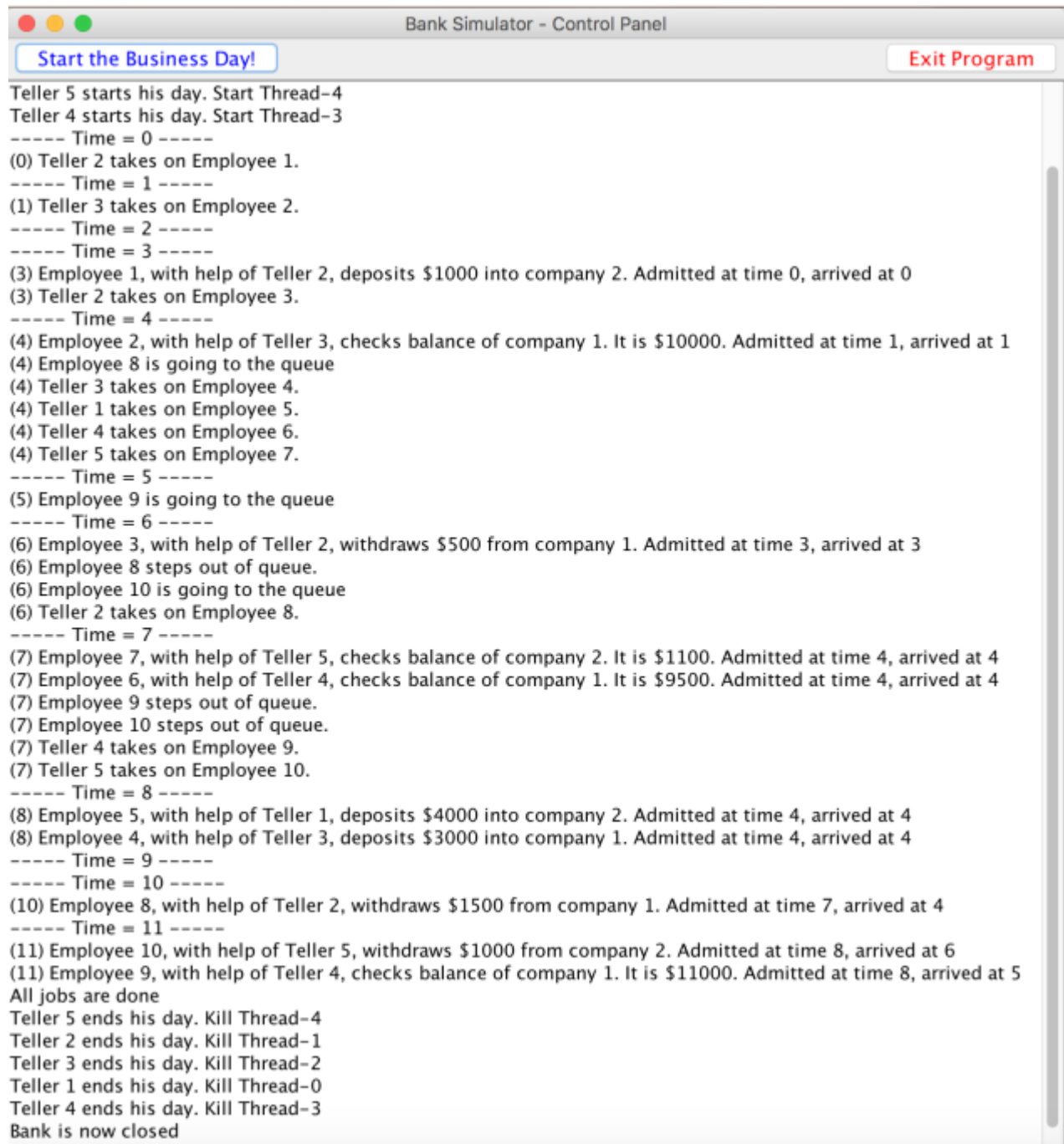
We would like to acknowledge the use and help of the following resources, without which our project would have been harder to implement:

1. GitHub (<https://github.com/>): this helped us for collaborating and storing the project files.
2. StackOverflow ([stackoverflow.com](https://stackoverflow.com/)): could be considered Computer Science's holy grail, this website helped us with errors/issues in our code that we could not figure out on our own.
3. Slack (<https://slack.com/>): this program was used for us to communicate on the project.
4. Javadocs (<https://docs.oracle.com/javase/7/docs/api/>): the Java Documentation that lists everything that can be done, from classes to methods to parameters etc.

## Appendix

Some screenshots from the project:

Figure 1: the figure below shows how a program run would display output in the interface:



```
Bank Simulator - Control Panel
Start the Business Day! Exit Program

Teller 5 starts his day. Start Thread-4
Teller 4 starts his day. Start Thread-3
----- Time = 0 -----
(0) Teller 2 takes on Employee 1.
----- Time = 1 -----
(1) Teller 3 takes on Employee 2.
----- Time = 2 -----
----- Time = 3 -----
(3) Employee 1, with help of Teller 2, deposits $1000 into company 2. Admitted at time 0, arrived at 0
(3) Teller 2 takes on Employee 3.
----- Time = 4 -----
(4) Employee 2, with help of Teller 3, checks balance of company 1. It is $10000. Admitted at time 1, arrived at 1
(4) Employee 8 is going to the queue
(4) Teller 3 takes on Employee 4.
(4) Teller 1 takes on Employee 5.
(4) Teller 4 takes on Employee 6.
(4) Teller 5 takes on Employee 7.
----- Time = 5 -----
(5) Employee 9 is going to the queue
----- Time = 6 -----
(6) Employee 3, with help of Teller 2, withdraws $500 from company 1. Admitted at time 3, arrived at 3
(6) Employee 8 steps out of queue.
(6) Employee 10 is going to the queue
(6) Teller 2 takes on Employee 8.
----- Time = 7 -----
(7) Employee 7, with help of Teller 5, checks balance of company 2. It is $1100. Admitted at time 4, arrived at 4
(7) Employee 6, with help of Teller 4, checks balance of company 1. It is $9500. Admitted at time 4, arrived at 4
(7) Employee 9 steps out of queue.
(7) Employee 10 steps out of queue.
(7) Teller 4 takes on Employee 9.
(7) Teller 5 takes on Employee 10.
----- Time = 8 -----
(8) Employee 5, with help of Teller 1, deposits $4000 into company 2. Admitted at time 4, arrived at 4
(8) Employee 4, with help of Teller 3, deposits $3000 into company 1. Admitted at time 4, arrived at 4
----- Time = 9 -----
----- Time = 10 -----
(10) Employee 8, with help of Teller 2, withdraws $1500 from company 1. Admitted at time 7, arrived at 4
----- Time = 11 -----
(11) Employee 10, with help of Teller 5, withdraws $1000 from company 2. Admitted at time 8, arrived at 6
(11) Employee 9, with help of Teller 4, checks balance of company 1. It is $11000. Admitted at time 8, arrived at 5
All jobs are done
Teller 5 ends his day. Kill Thread-4
Teller 2 ends his day. Kill Thread-1
Teller 3 ends his day. Kill Thread-2
Teller 1 ends his day. Kill Thread-0
Teller 4 ends his day. Kill Thread-3
Bank is now closed
```

Figure 2: this shows how the program would print out in the console if the user chooses to do so. The interface displays a message informing user that the program is printing to console:

```

Main.java  FileScanner.java  BankFrame.java

1 package v1;
2
3 import java.awt.EventQueue;
4
5 /**
6  * Read Documentation for Execution Details.
7  * For proper execution, this project has to contain:
8  * - src folder with v1 package; should contain 8 classes
9  * - resources folder with 3 configs: named config1.txt, config2.txt, config3.txt
10
11 */
12
13 public class Main {
14     public static void main(String[] args) {
15         // TODO: Add your code here
16     }
17 }

```

Problems Javadoc Declaration Search Console

Main [Java Application] /Library/Java/JavaVirtualMachines/jdk1.8.0\_40.jdk/Contents/Home/bin/java (28 Nov 2016 04:34:19)

Entering manual configuration mode...reverting to console use  
Please use console to manually enter a file number.  
Which manual configuration should be run? (If in doubt, use 1, 2 or 3!)

2

File exists!  
Do you wish to print results in Console or Frame?  
(Enter 'console' or 'frame')

console

PRINTING TO CONSOLE

Bank is now open!

Teller 1 starts his day. Start Thread-0  
Teller 2 starts his day. Start Thread-1  
Teller 3 starts his day. Start Thread-2  
Teller 4 starts his day. Start Thread-3  
Teller 5 starts his day. Start Thread-4  
Teller 6 starts his day. Start Thread-5  
Teller 7 starts his day. Start Thread-6  
Teller 9 starts his day. Start Thread-8  
Teller 8 starts his day. Start Thread-7  
Teller 10 starts his day. Start Thread-9

----- Time = 0 -----

(0) Teller 1 takes on Employee 1.

----- Time = 1 -----

----- Time = 2 -----

(2) Teller 6 takes on Employee 2.

----- Time = 3 -----

(3) Employee 1, with help of Teller 1, withdraws \$2200 from company 2. Admitted at time 0, arrived at 0

(3) Teller 1 takes on Employee 3.

----- Time = 4 -----

(4) Teller 2 takes on Employee 4.  
(4) Teller 4 takes on Employee 5.  
(4) Teller 3 takes on Employee 6.

----- Time = 5 -----

(5) Employee 2, with help of Teller 6, checks balance of company 3. It is \$210000000. Admitted at time 2, arrived at 2

Bank Simulator - Control Panel

Start the Business Day! Exit Program

PRINTING TO CONSOLE  
Check console for program execution



## CS3103 Operating Systems Group Project Summary Report

Figure 3: Screenshot from our online repository on GitHub:

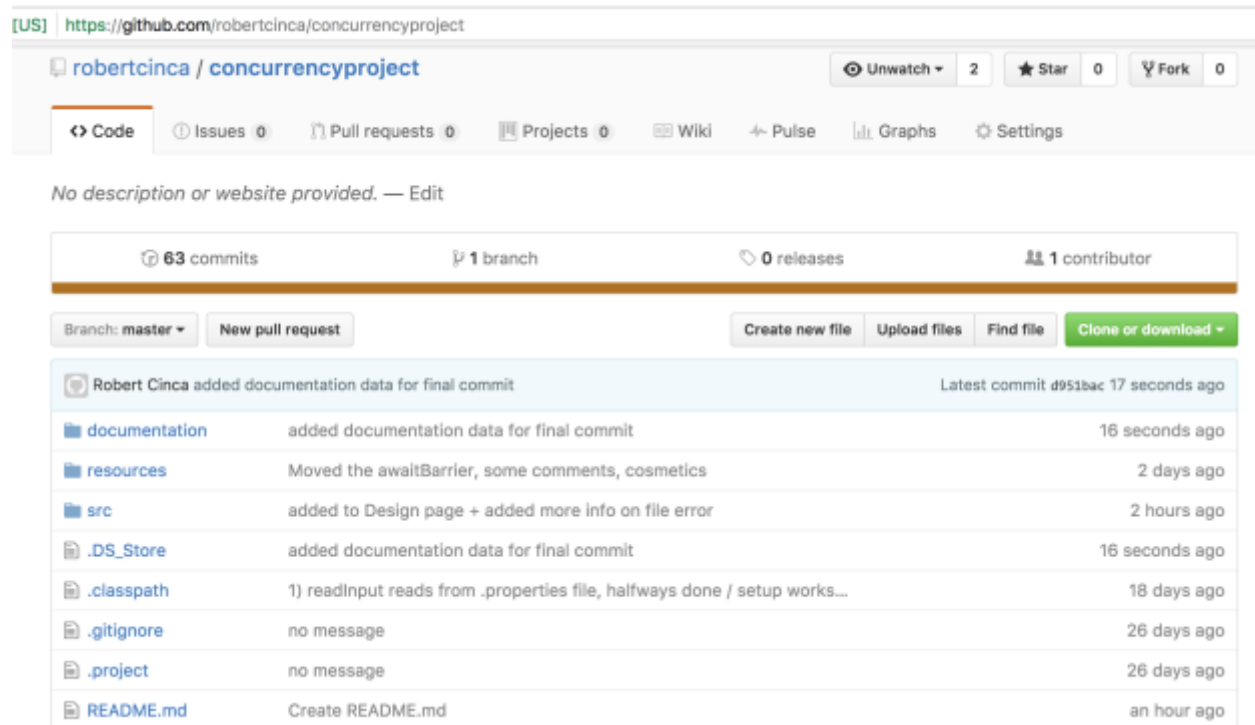


Figure 4: Eclipse IDE and Project structure seen in left bar:

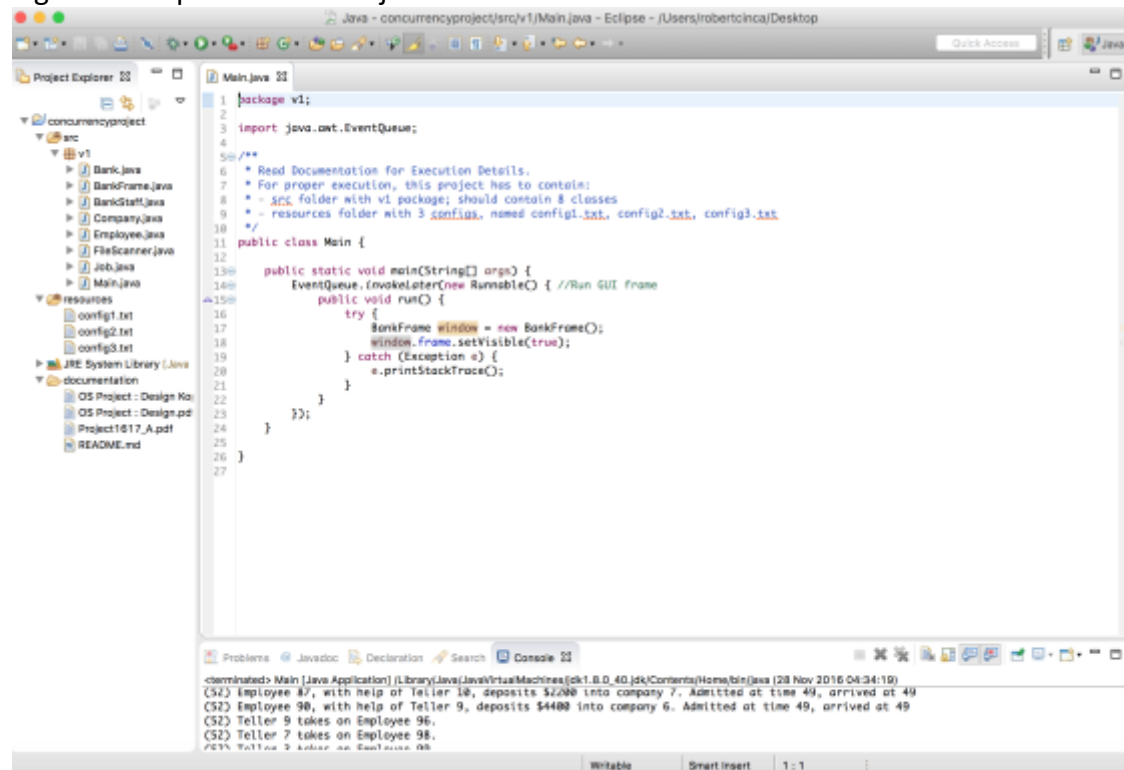


Figure 5: Overview of how barriers were used to handle threads:

