**Design Programming Project 2**

Brenda Garcia

CSIS 3810 - Operating Systems Concepts

Dr. Gregory Simco

November 17th, 2021

Table of Contents

[Methodology 3](#_Toc87988094)

[Design 3](#_Toc87988095)

[Scope/Overview 3](#_Toc87988096)

[Data Design 4](#_Toc87988097)

[Data Structures 4](#_Toc87988098)

[Semaphores 4](#_Toc87988099)

[Thread Sleep 5](#_Toc87988100)

[Producer Threads 5](#_Toc87988101)

[Consumer Threads 6](#_Toc87988102)

[Architectural Design 8](#_Toc87988103)

[Initialization 8](#_Toc87988104)

[Bank Account Threads 8](#_Toc87988105)

[Termination 9](#_Toc87988106)

[Class and Object Design/Modules 9](#_Toc87988107)

[1. bank.java 9](#_Toc87988108)

[2. Consumer.java 10](#_Toc87988109)

[3. Producer.java 12](#_Toc87988110)

[4. BankAccount.java 14](#_Toc87988111)

[5. CreditCard.java 15](#_Toc87988112)

[Diagram 16](#_Toc87988113)

[Pseudocode 17](#_Toc87988114)

[Interface Design 18](#_Toc87988115)

[Test Provisions 19](#_Toc87988116)

# Methodology

Concurrency is a property of systems in which the processes of a computation are carried out simultaneously and can interact with each other. Synchronization is used to cause 2 or more processes to work sequentially in such a way that they do not interfere with each other, and they can get a job done. In this paper, primitive semaphores (acquire and release) were used along with a virtual timeline created through the use of for loops and thread built-in functions with the primary aim of developing an automated bank account management system in JAVA capable of running multiple consumer and producer threads. Five classes are part of the solution to the system: the main class which is called “bank”, and 4 other classes that connect with the main class to initialize the system and provide the user with meaningful information about their bank account.

# Design

## Scope/Overview

The Java project executed had the goal of designing and developing a program that would simulate the operation of a bank account management system. The program runs a of bank account that holds a debit card and a credit card with a limit of $1500, and it simulates a one-year period of time for a college student (Fall semester, Winter semester, and Summer semester). Transactions made by producers (which add money to the debit card) and made by consumers (which deduct money from the debit card or add to the outstanding balance of the credit card) take place. The system requires there to be a producer thread (Financial Aid) that will add money to the bank account prior to the beginning of each semester. Right after, the cost of tuition, books and fees is deducted from the bank account on the beginning of each term, plus dorm and food fees are deducted at the beginning of every month, and there are 5 months in the created hypothetical system. These bills are considered required consumer threads. In addition to the system requirements, other producer and consumer threads are to be implemented in order to simulate the different financial decisions that a typical college student could make, plus an event thread is included at the end of the academic year as an additional feature. To avoid concurrency problems, the system is synchronized only using the semaphores primitives (acquire and release). To make the system automated, a virtual timeline is created using for loops, and a JAVA built-in function for threads called “sleep()”. These tools enable the different threads to iterate the number of times that they should appear and in a specific order.

## Data Design

### Data Structures

An array list of threads is used in the bank class to store different consumer and producer threads. This kind of array allows for flexibility in sizing and implements the Thread interface. The array list is then used within a for loop to run each thread using a JAVA method called “join()” which basically waits for all threads to “die” before executing an event thread and printing the final status on the bank account.

### Semaphores

Semaphores are a type data structure that is being used for synchronization between the producer and consumer threads. In specific, they are used in the producer and the consumer classes, as well as in the bank account and credit card classes. The value of the semaphore is the number of units of the resource that are available (in the project, a binary semaphore is used with an initial value of 1 since there is only one resource). To enter a critical zone, a thread must acquire the right of access using the “acquire()”, and when exiting it frees it using the “release()” in JAVA. Semaphores are very important in the system as it cannot afford to have race conditions that mess up the amount deposited and deducted, and then result in a wrong message output.

1. Acquire: this primitive is used to block a resource.
2. Release: this primitive is used to release the blocked resource, allowing another thread to use it.

### Thread Sleep

To put a thread to sleep for a certain time in milliseconds, the sleep() method is used. What this method does, is that it pauses execution on a thread for a specific period of time, enabling other threads to run on the system. In the program, putting a thread to sleep allows for the creation of a timeline which enables threads to happen in the order that they should. This JAVA built-in function also helps set a pace for the system, simulating the passing of weeks, months, and academic terms. In the case of the current system, it was built to have each academic term be 6000 milliseconds long. This means that within that time span, the other threads that happen each semester can be arranged to sleep for a certain amount of time before executing so that they take place in their assigned spot in the timeline.

### Producer Threads

The system has 4 producer threads. These threads are communicating with the Producer class which is in charge of the way the thread executes, and money is deposited into the bank account.

1. Financial Aid: this thread iterates three times symbolizing the way in which each semester financial aid gets deposited. The amount assigned to the student’s account each month is equivalent to the amount paid in college expenses (tuition, books, student fees, dorm, and food) being $24310.0.
2. Job: this thread happens 2 times each month of each semester. In the made-up bank system, the semester has 5 months, and each of those months, a paycheck is deposited two times (representing the way employees typically get paid every two weeks each month). The paycheck of $972.0 gets deposited into the bank account.
3. Birthday Gift: this thread happens one time in the program in the second month of the first academic term (September). It represents a birthday gift of $100 sent by the user’s mom for their birthday. This money is then deposited into the bank account.
4. Found Money: this thread happens one time in the program in the third month of the last academic term being Summer. It represents a $50 bill that was found on the street by the user, and then was deposited into the bank account.

### Consumer Threads

The system has 7 producer threads, with one of the threads (Lottery) being both a consumer and a possible producer thread. These threads are communicating with the Consumer class which is in charge of the way the thread executes, and money is withdrawn from the bank account (in the case of the Lottery thread, it also handles the way money won from playing the lottery is deposited).

1. Tuition: this thread works the same number of times as the Financial Aid thread. It iterates 3 items symbolizing the way in which each semester tuition gets withdrawn after financial aid gets deposited. The tuition cost of attending the university, being $15380.0, is withdrawn from the student’s bank account.
2. Books: just like tuition, the fee of $1240.0 paid by the student for books is withdrawn from the bank account every semester.
3. Student Fees: just like tuition, the fees of $500.0 paid by the student for attending the university are withdrawn from the bank account every semester.
4. Dorm: this thread happens each month of the semester (5 times). It symbolizes the way a student has to pay their assigned dorm fee every first of each month of the term. The amount of $1178.0 gets withdrawn from the bank account.
5. Food Bill: this thread happens each month of the semester (5 times). It symbolizes the way a student has to pay their assigned food plan for 21 meals per week every first of each month of the term. The amount of $ 260.0 gets withdrawn from the bank account.
6. Laptop: this thread happens one time in the program in the third month of the second semester. It is a necessity for any college student to have a laptop/computer/tablet in order to do their schoolwork. Therefore, the student in the system buys a $1300 laptop using their credit card. This amount gets added to the credit card as an outstanding balance.
7. Lottery\*: this thread happens one time in the program right at the end of the academic year. This is an event thread that asks the user whether they want to buy a lottery ticket with their credit card. If the user says yes, the price of the ticket, being $30, gets added to the credit card as an outstanding balance, and the amount won gets calculated using Math.ramdom(). If the prize is not $0, the user gets the prize money deposited to their bank account. This thread works as both a consumer and a producer thread in the way that if the user decides to play the lottery, the price gets added to the credit card (consumer), and the amount won gets deposited (producer).

## Architectural Design

### Initialization

The system starts with the bank account at $0 as no activity has happened. An array list is created and instances of each producer and consumer threads are made. The instances of each class hold the name of the thread, the assigned account (whether it is the debit or credit account), and the amount of money that each thread will withdraw or deposit. This information is then passed down to the producer and consumer classes respectively in order for them to run the threads. Each thread is then started with the “start()” JAVA method which begins execution of the thread.

### Bank Account Threads

After each thread begins execution, the JAVA Virtual Machine calls the “run()” method of the thread. The Producer and Consumer classes hold a “run()” method that specifies the way the thread should execute. If the thread is a producer then the Producer class is called. Within this class the corresponding attributes from the constructor are assigned and on the start of the thread, the “run()” method is executed by the system. The first thing done by this function is that it uses a semaphore to acquire the resource before entering the critical section. Then, depending on the kind of thread, different actions are taken. Finally, before the thread has finished running, the resource is released. If the thread is a consumer, then the Consumer class is called, and a similar process to that of the producers takes place. Since tuition should get withdrawn after financial aid, a “sleep()” is used to simulate the passing of time before the tuition thread is to take place. No matter what kind of thread is running, as long as it uses the debit card, each one will call either the “credit” or the “withdraw” function from the BankAccount class. Each of these functions use a semaphore to acquire the resource, and then release it after going through the critical section which is either after the money of the debit card is increased (due to a producer), or the money is decreased (due to a consumer). If the thread happens to be using the credit card, then the CreditCard class will be called by each thread, and the “useCreditCard()” method will be implemented to make the payment of the transaction by adding to the outstanding balance of the credit card only if it has not gone past the limit provided for the card.

### Termination

The system waits for all threads to “die” by using the “join()” method, however, as they are running, the user is displayed important messages that depict the status of the bank account. After they are all done executing, a final thread prompts the user to choose whether they want to run the Lottery thread which represents them buying a ticket and playing the lottery. If the user says yes, the thread executes, adds the transaction to the credit card, and if a prize was won, it is deposited to the bank account. To end the system, an exit message is shown to the user, displaying the final account balance, the final outstanding balance of the credit card, as well as the available credit left for the user to spend with the card.

## Class and Object Design/Modules

#### 1. bank.java

This class serves the functionality of a “main”. It makes different producer and consumer objects, which are instances of the Producer and Consumer classes. This class is in charge of running the system and going through all the steps necessary for all threads to start and run successfully.

The following data members are used in the class:

* balance : double
  + this variable holds the money that the bank account starts with. Initially, it starts with zero dollars.
* outstandingBalance : double
  + this variable holds the money to pay (outstanding balance) that the credit card starts with. Initially, it starts at zero dollars.

The following methods are implemented in the class:

* main(String[])
  + The “main” is the entry point of the program. It holds each producer and consumer object, runs them, and prints a final message about the status of the bank account.

#### 2. Consumer.java

This class is implemented for each consumer thread.

The following data members are used in the class:

* amount : double
  + this variable indicates the amount of money that the consumer thread is to withdraw from the bank account.
* name : String
  + this variable indicates the name of the consumer thread, which will then be used to determine the way it should run.
* bankAccount : BankAccount
  + this variable uses the BankAccount data type which allows the Consumer class to implement functions from this class like the withdraw() and credit().
* creditAccount : CreditCard
  + this variable uses the CreditCard data type which allows the Consumer class to implement functions from this class like the useCreditCard() and payCreditCard().
* semaphore : Semaphore
  + this variable is a semaphore that is used to synchronize the threads in the Consumer class.

The following constructors are used in the class:

* Consumer(String name, BankAccount bankAccount, double amount)
  + this is a constructor for the amount of money being withdrawn from bank account by a thread.
* Consumer(String name, BankAccount bankAccount, CreditCard creditAccount,double amount)
  + this is a constructor for the amount of money being charged to the credit card and maybe deposited in the bank account (lottery thread).

The following methods are implemented in the class:

* run()
  + this method executes the threads and implements the acquire and release actions for the semaphore.
* CollegeExpenses()
  + this function will deduct tuition, books, and fees at the start of each semester by using a for loop that iterates three times. This method is called within the run() function; however, it was implemented as a separate function for readability purposes.
* DormAndFood()
  + this function will deduct dorm and food bills every month for each semester by using a nested for loop (one that iterates five times for each of the 5 months, and another that iterates three times for each of the academic terms). This method is called within the run() function; however, it was implemented as a separate function for readability purposes.
* Lottery()
  + this function will charge the credit card with the lottery ticket price and if the user wins something by playing the lottery, the money will get deposited in his/her account.

#### 3. Producer.java

This class is implemented for each producer thread.

The following data members are used in the class:

* amount : double
  + this variable indicates the amount of money that the producer thread is to deposit into the bank account.
* producer : String
  + this variable indicates the name of the producer thread, which will then be used to determine the way it should run.
* bankAccount : BankAccount
  + this variable uses the BankAccount data type which allows the Producer class to implement functions from this class like the withdraw() and credit().
* semaphore : Semaphore
  + this variable is of type semaphore, and it is used to synchronize the threads in the Producer class.

The following constructor is used in the class:

* Producer(String, BankAccount, double)
  + this is a constructor for the amount of money being deposited into the bank account by a thread.

The following methods are implemented in the class:

* run()
  + this method executes the threads and implements the acquire and release actions for the semaphore.
* FinancialAid()
  + this function will deposit the money given as financial aid prior to the beginning of each semester, by using a for loop that iterates three times. This method is called within the run() function; however, it was implemented as a separate function for readability purposes.
* Job()
  + this function will deposit the money given by the job two times each month for each semester by using a nested for loop (one that iterates two times for the two weeks that user gets paid, one that iterates five times for each of the 5 months, and another that iterates three times for each of the academic terms). This method is called within the run() function; however, it was implemented as a separate function for readability purposes.

#### 4. BankAccount.java

This class acts as a debit card, and it is in charge of actually carrying out the different transactions involved when a consumer or producer uses the card.

The following data members are used in the class:

* balance : double
  + this variable keeps track of the balance in the debit card
* semaphore : Semaphore
  + this variable is of type semaphore, and it is used in the withdraw and credit methods to synchronize the threads in the class.

The following constructor is used in the class:

* BankAccount(double)
  + this constructor is later used in the main to initialize the system.

The following methods are implemented in the class:

* getBalance()
  + this getter returns the amount of money in the bank account.
* setBalance(double)
  + this setter is used to set the amount of money in the bank account after the different transactions have been made.
* credit(double)
  + this method is in charge of updating the current balance to reflect that a deposit has been made.
* withdraw(double)
  + this method is in charge of updating the current balance to reflect that a withdrawal has been made.

#### 5. CreditCard.java

This class acts as a credit card, and it is in charge of actually carrying out the different transactions involved when a consumer uses the card.

The following data members are used in the class:

* outstandingBalance : double
  + this variable keeps track of the outstanding balance in the credit card.
* creditLimit : double
  + this variables is declared as a final, and it defines the credit limit of the card.
* semaphore : Semaphore
  + this variable is of type semaphore, and it is used in the useCreditCard and payCreditCard methods to synchronize the threads in the class.

The following constructor is used in the class:

* CreditCard(double)
  + this constructor is later used in the main to initialize the system.

The following methods are implemented in the class:

* getOutstandingBalance()
  + this getter returns the outstanding balance in the bank account.
* setOutstandingBalance(double)
  + this setter is used to set outstanding balance in the bank account after the different transactions have been made.
* availableCredit()
  + this method returns the credit left to spend by the college student.
* useCreditCard(double)
  + this method is in charge of updating the current outstanding balance to reflect that the credit card has been used to pay for something.
* payCreditCard(double)
  + this method is not implemented in the system but is still included for later implementations. It takes care of updating the current outstanding balance to reflect that debit card has been used to pay some, or all of the credit card’s outstanding balance.

### Diagram

Big picture diagram of the system’s design

### Pseudocode

The following displays the pseudocode for key parts of the system:

Consumer{

run(){

acquire(consumer); //acquire semaphore

sleep(1000); //sleep to simulate the way in which financial aid gets deposited first

if the consumer is equals to “Tuition” or to “Books” or to “Student Fees"{

CollegeExpenses(); //deduct tuition, books, and fees at the start of each semester

}

else if the consumer is equals to “Dorm” or to “Food Bill”{

DormAndFood(); // deduct dorm and food bills at the start of each month

}

else if the consumer is equals to “Laptop”{

sleep(8625); //simulates the passing of time until the month when the laptop is bought arrives

print about the money that will be charged;

userCreditCard(amount); //charge the credit card by calling the method from the CreditCard class

}

else if the consumer is equals to “Lottery”{

print about the money that will be charged;

userCreditCard(amount); //deduct lottery ticket price

Lottery(); // if user wins, the money will get deposited in his/her account.

}

release(consumer); //release semaphore

}

}

Producer{

run(){

acquire(producer); //acquire semaphore

if the producer is equals to “Financial Aid”{

FinancialAid(); //deposit financial aid before the start of each semester

}

else if the producer is equals to “Job”{

sleep(1500); //paycheck does not get deposited until after the first of each month

Job(); // deposit money from job paycheck two times a month

}

else if the producer is equals to “Birthday Gift”{

sleep(2625); //simulates the passing of time until the month the birthday months arrives

print about the money that will be deposited;

credit(amount); //deposit money into the bank account

}

else if the producer is equals to “Found Money”{

sleep(15625); //simulates the passing of time until the month when the money is found arrives

print about the money that will be deposited;

credit (amount); //deduct lottery ticket price

}

release(producer); //release semaphore

}

}

BankAccount{

credit(amount) { //deposit money

acquire(thread); //acquire semaphore

temp = getBalance(); //get the balance of the account

temp += amount; //add up the balance with the amount being deposited

setBalance(temp); //set the balance of the account

release(thread); //release semaphore

}

withdraw(amount) { //withdraw money

acquire(thread); //acquire semaphore

if balance is less than the amount being deducted{

print “Not enough funds”

return; //exit

}

temp = getBalance(); //get the balance of the account

temp -= amount; //subtract the balance with the amount being withdrawn

setBalance(temp); //set the balance of the account

release(thread); //release semaphore

}

}

# Interface Design

The zip file called “Bank System” should empty its contents into one subdirectory. This subdirectory contains a “src” folder that holds the five JAVA classes. The user is to open up the command prompt and from within the “src/” directory, compile the program by writing “javac \*.java”. Right after, five “.class” files will be created for each JAVA class. To run the program, the user should type “java bank”. The command line should show different messages that are generated as each transaction is made in the bank account. Academic terms are neatly organized and separated, and the months of the academic year are labeled. Additionally, if an error were to occur, a message indicating the error would be listed. Overall, within the command prompt, the user will find important information about the bank account manager, like when a producer deposits money, when a consumer withdraws money, when the event thread prompts the user to enter a Y for yes and N for no, as well as when the program is done running.

# Test Provisions

Several testing techniques were used to make sure that the system works as it should.

* Expected messages: print statements were added throughout the program to facilitate the testing of the system. Since a bank system works by adding and subtracting numbers, the calculations of what each semester should yield as the bank account balance were done manually to later corroborate them with the program’s output.
* Potential errors: Try and catch statements were implemented throughout the system, especially for the semaphore and Thread functions. Input validation was used to make sure the user’s input would be read and executed correctly.
* Thread synchronization: It was also important to make sure that there were no thread conditions, or deadlocks that were altering the program’s output. This was tested through the use of print statements to verify the program was working correctly.