|  |  |  |  |
| --- | --- | --- | --- |
|  | 5/1/2015 |  | |
| |  |  | | --- | --- | |  |  | |  |  | |  |  | |  |  | | |  | |
| SSE 554 Project 4  *Encrypting the Bank* | | | |
|  |  | | |  |  |  | | --- | --- | --- | |  |  |  | |  |  |  | |  |  |  | |  |  |  | |  |  |  | |  |  |  | |
|  |  | | Daryl Ebanks & Josh Deremer |



SSE 554 Project 4

Encrypting the Bank

# Introduction

This project is the final installment of the banking concept initially created by Daryl in his first project. Since the inception of the program, multithreading, client-server interaction, and database storage have all been added to the original basic bank program. In this last portion, we will add encryption to the data being transmitted. The data will be encrypted before being sent from the client program to the server program and vice versa. Each direction will have its own encryption method, one created by Josh and the other created by Daryl. Once again, we will begin by discussing some key topics before presenting the integration of encryption into the basic bank program.

Table of Contents

[Introduction 1](#_Toc418291004)

[Encryption 3](#_Toc418291005)

[Symmetric Ciphers 3](#_Toc418291006)

[Public Key Ciphers 4](#_Toc418291007)

[Testing 5](#_Toc418291008)

[Description of Source 5](#_Toc418291009)

[Client to Server Encryption 5](#_Toc418291010)

[Server to Client Encryption 7](#_Toc418291011)

[Code in Action 8](#_Toc418291012)

[DVCS 9](#_Toc418291013)

[Test Code 10](#_Toc418291014)

[Account Test 10](#_Toc418291015)

[Bank Test 12](#_Toc418291016)

[Database Test 14](#_Toc418291017)

[Encryption Test 16](#_Toc418291018)

[Source Code 17](#_Toc418291019)

[Account 17](#_Toc418291020)

[Bank 18](#_Toc418291021)

[BankClient 22](#_Toc418291022)

[BankGUI 25](#_Toc418291023)

[Bank\_Main 32](#_Toc418291024)

[BankServer 33](#_Toc418291025)

[CheckingAccount 34](#_Toc418291026)

[ClientThread 35](#_Toc418291027)

[Database 39](#_Toc418291028)

[SavingsAccount 42](#_Toc418291029)

[Client Encryptor 43](#_Toc418291030)

[Server Decrypter 45](#_Toc418291031)

[Encryption Utility 47](#_Toc418291032)

# https://blog.spideroak.com/wp-uploads/encryption.jpgEncryption

Encryption is a very important aspect of security with digital information. In many situations, it is undesirable and even dangerous to have one’s transmitted data available for the world to read as it is traveling from its source to destination. When information is encrypted, it is not visible to anyone who might try to read it and requires a matching key in order to decrypt and make understandable.

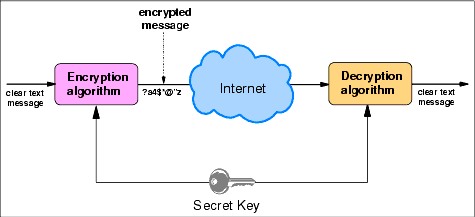
The encryption process involves using a cipher to encode and decode data. Ciphers can be personally created or randomly generated using the utilities available in the Java API. Throughout the years, more and more algorithms have been created to generate stronger and stronger ciphers in order to deter the continuing advancement of cyber thieving technology and computer processing capabilities. Because of this, there is a large number of cipher algorithms available, however, half or more are already outdated and can be easily cracked.

Most ciphers will generally require a cipher key to go along with them. This key can be used to ensure a particular behavior out of the cipher. Any changes in this key will result in changes from the output of the cipher. It is important when encrypting and decrypting data, the same key or matching keys are used, otherwise the data cannot be decrypted properly.

In this project, we make use of two different types of ciphers: symmetric ciphers and public key ciphers.

## Symmetric Ciphers

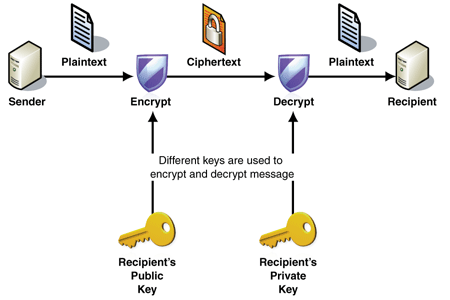
A symmetric cipher is a type of cipher that uses the exact same key for both encryption and decryption. This is a very common encryption method that has a wide variety of cipher algorithms created following this model. The process for a symmetric cipher involves first creating a secret key to be used with both encryption and decryption of data. Then, the user begins data transfer by accepting a plaintext message as the input and the encrypting the message by running it through the encryption algorithm which is governed by the secret key. The encrypted message then travels along the public path where the encryption is required to keep it secure, most often the internet. Once the message reaches its destination, it is run through the decryption algorithm, which is governed by the same secret key as before, and emerges as the original plaintext message. This process can be seen in the image below.



This encryption method is very economical in terms of processing resource usage, however, it requires that the secret key be known by both the sender and receiver of the message. This may be perfectly fine or could pose a serious problem depending on the application. Because of this, other ciphers were created, particularly the public key cipher.

## Public Key Ciphers

The issue mentioned earlier with the symmetric encryption cipher where both the sender and receiver need to know the secret key is known as key distribution. In order for one user to send a message to another user, the secondary user must have the same key the first user uses to encrypt the message. However, the first user cannot just send that new key through an unsecure channel, otherwise the purpose of encrypting the method in the first place is defeated.



This is where public key cryptography comes in. With a public key cipher, the users can have a key pair consisting of a public key and a matching private key. The public key can be access and used by anyone to encrypt a message and send it to the user with the private key. Only the private key can then decrypt the message. Therefore, one user can simply publish their public key anywhere. The second user can grab the public key and use it to encrypt their message. Then, the second user sends the encrypted message to the first user who decrypts the message using the matching private key.

However, as fantastic as that may sound, there are still drawbacks to using public key ciphers. First and foremost is that public key algorithms are much slower than symmetric key algorithms. Because of this, the transfer of large quantities of data is not conceivable for public key ciphers. In order to find a feasible use for ciphers, we combine the two.

# Testing

The testing for this project was very simple. The objective was to create an encryption scheme that could encrypt and decrypt text. The resulting unit test simply creates a string, encrypts it, compares it to the original, decrypts it and compares it again. A lot of time was spent attempting to devise a method of generating a symmetric key on separate machines without explicit communication. When these methods failed, or were too obvious to provide adequate security, they were abandoned for the public/symmetric method described in the book.

The methods passed the test.

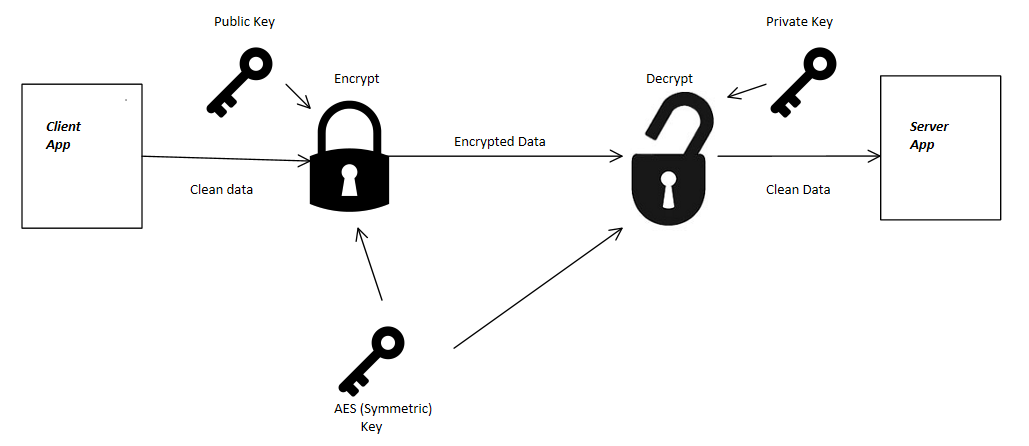
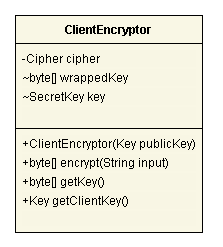




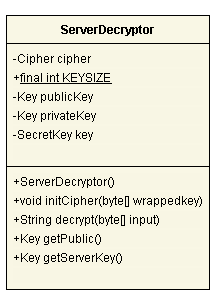
# Description of Source

## Client to Server Encryption

The client to server encryption method makes use of both the symmetric cipher and the public key cipher. By using the public key cipher first, the server can create a key pair and send the pubic key out to the client. The client then creates a symmetric key and uses the public key to encrypt the symmetric key and send the encrypted key back to the server. Next, the server unencrypts the symmetric key. Now, both the server and the client have a matching symmetric key and can send and receive data using the symmetric cipher method. Using this method, the expensive public key encryption is only applied to a small amount of data.



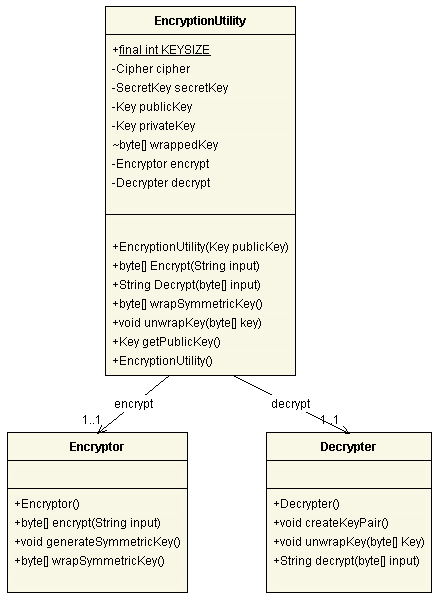
The symmetric cipher was encrypted using the Advanced Encryption Standard (AES) algorithm. A key for the cipher was generated using the SecureRandom generator from the Java API. The public key cipher made use of the RSA algorithm which was invented by Rivest, Shamir and Adleman. This cipher also generated a random key pair using the SecureRandom generator.

Two classes were created to perform the cryptography. The server side class, ServerDecryptor, keeps the private key for the public key cipher and only decrypts messages from the client side application. The client side class, ClientEncryptor, gets the public key for the public key cipher and only encrypts messages to send to the server application. The encryption and decryption methods convert the byte arrays to strings and vice versa to send and receive the information.

The ClientEncryptor was initiated in the client application and encrypts every message before sending it out to the server. Meanwhile, the ServerDecryptor is initiated in the server application and is used to decrypt every message coming from the client.

## Server to Client Encryption

The server to client encryption method also uses the same method of encryption; a symmetric encryption key wrapped by a public key. The main difference is the implementation. The server to client encryption code is implemented as a single class that handles both encryption and decryption.

This Encryption Utility class requires the same steps be taken, but some of the steps are bound together for simplicity of use. Each new Encryption Utility can only be used for either encryption or decryption, but it can be used for either.

The class has two helper classes that control separate aspects of the encryption process. The Encryptor and Decrypter, as they are named function as you might expect. These classes are used to ensure that a utility created for encrypting cannot easily be used for decrypting and visa-versa.

The constructor determines how each utility can be used. The Encrypt, Decrypt, wrap Semmetrical Key and unwrap Key methods invoke the methods of the helper classes.

The helper classes themselves contain the logic behind the encryption process. Each of the classes contains the necessary methods for its part of the process.

The diagram describes the relationship between each of the members of the Encryption Utility class.

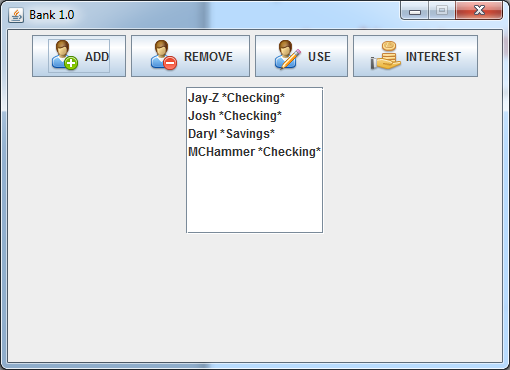
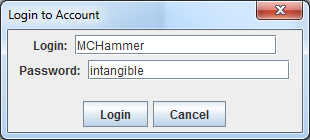
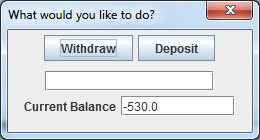
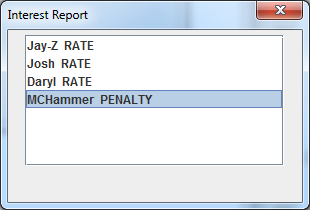
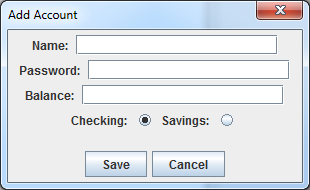
There are small changes in the client and server that were made to accommodate the use of encryption. This is one such example:



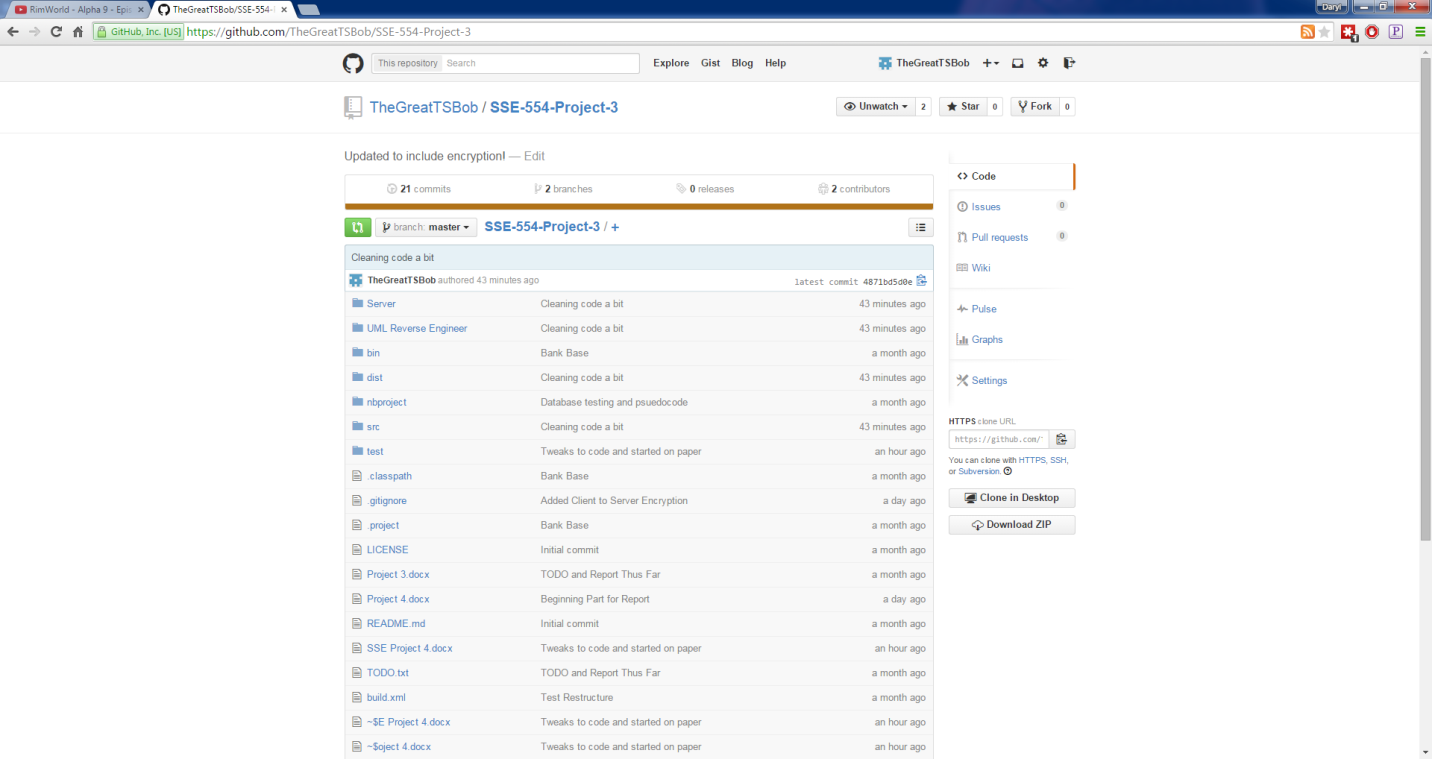
In this example the Encryption Utility encryptor is used to encrypt the message before it is sent across the network.

# Code in Action

The display and functionality of the code are identical to the previous version. Not much to see here.



# DVCS



The DVCS was instrumental in the collaboration once again. The ability to make changes to code and synchronize these changes with a group makes group work much more feasible and simple.

# Test Code

## E:\Newsbin Downloads\Documents\SSE 554 Project3\accounttest\accounttest-page-001.jpgAccount Test



## E:\Newsbin Downloads\Documents\SSE 554 Project3\banktest\banktest-page-001.jpgBank Test



## Database TestE:\Newsbin Downloads\Documents\SSE 554 Project3\databasetest\databasetest-page-001.jpg

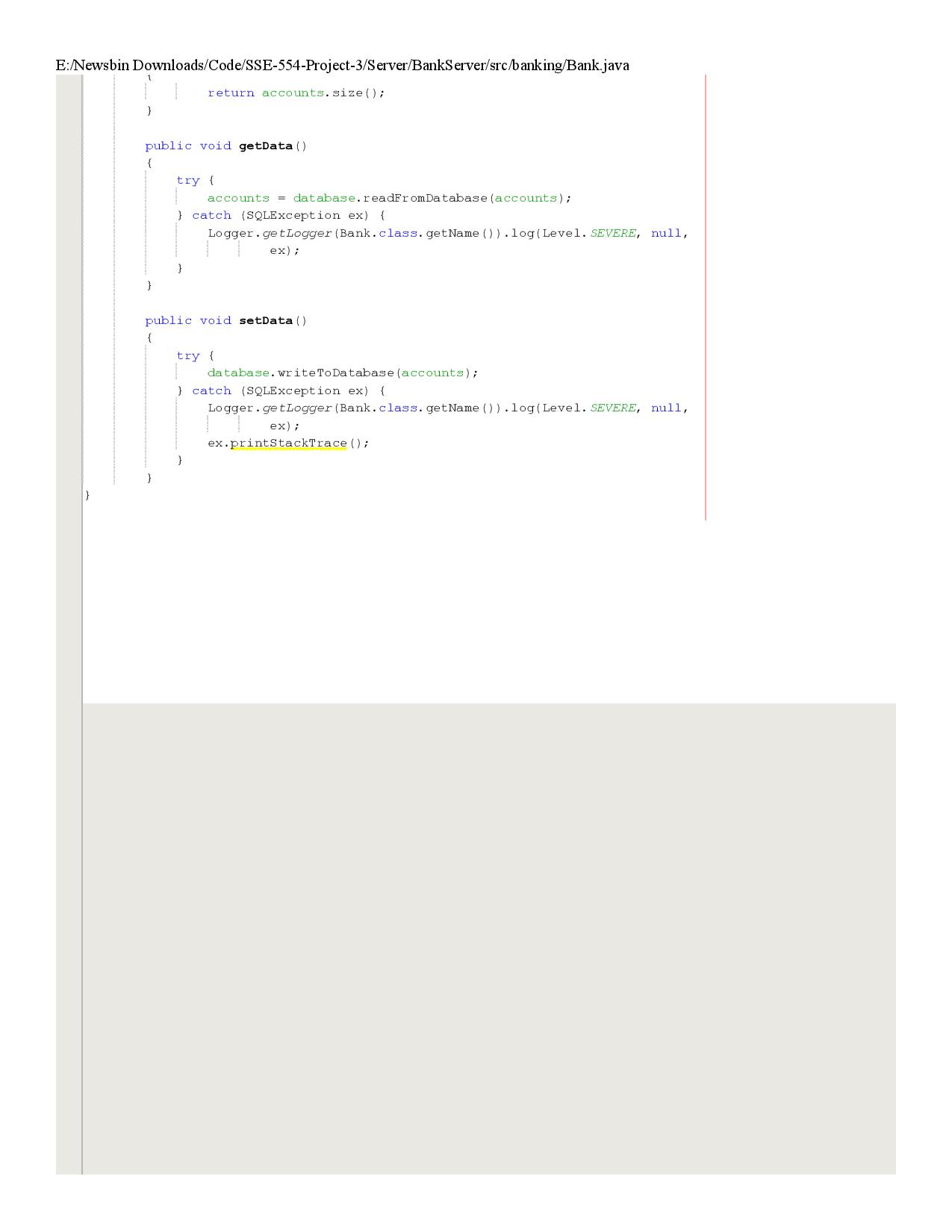


## E:\Newsbin Downloads\Documents\SSE 554 Project4\EncryptionTest\EncryptionTest-page-001.jpgEncryption Test

# Source CodeE:\Newsbin Downloads\Documents\SSE 554 Project3\Account\Account-page-001.jpg

## Account

## E:\Newsbin Downloads\Documents\SSE 554 Project3\Bank\Bank-page-001.jpgBank

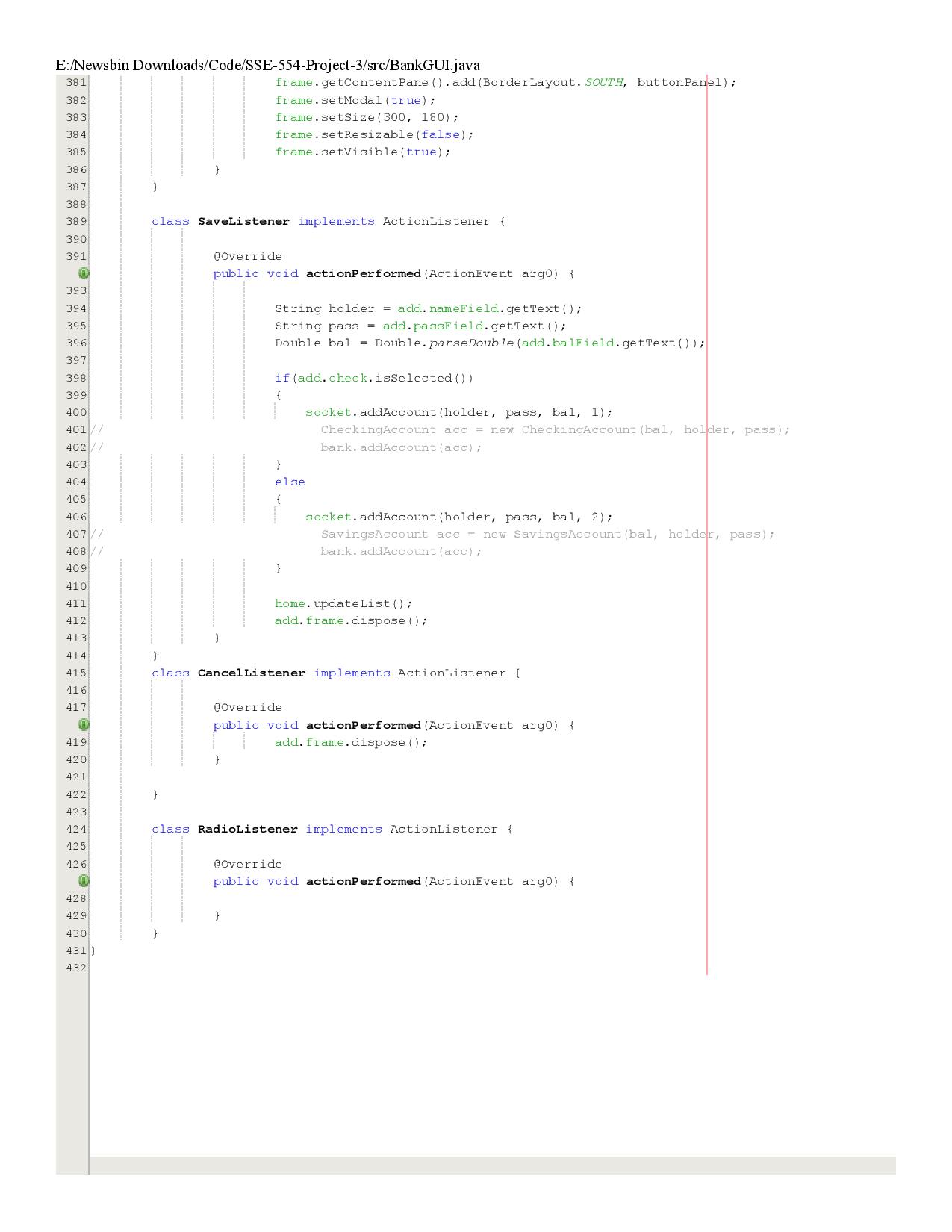
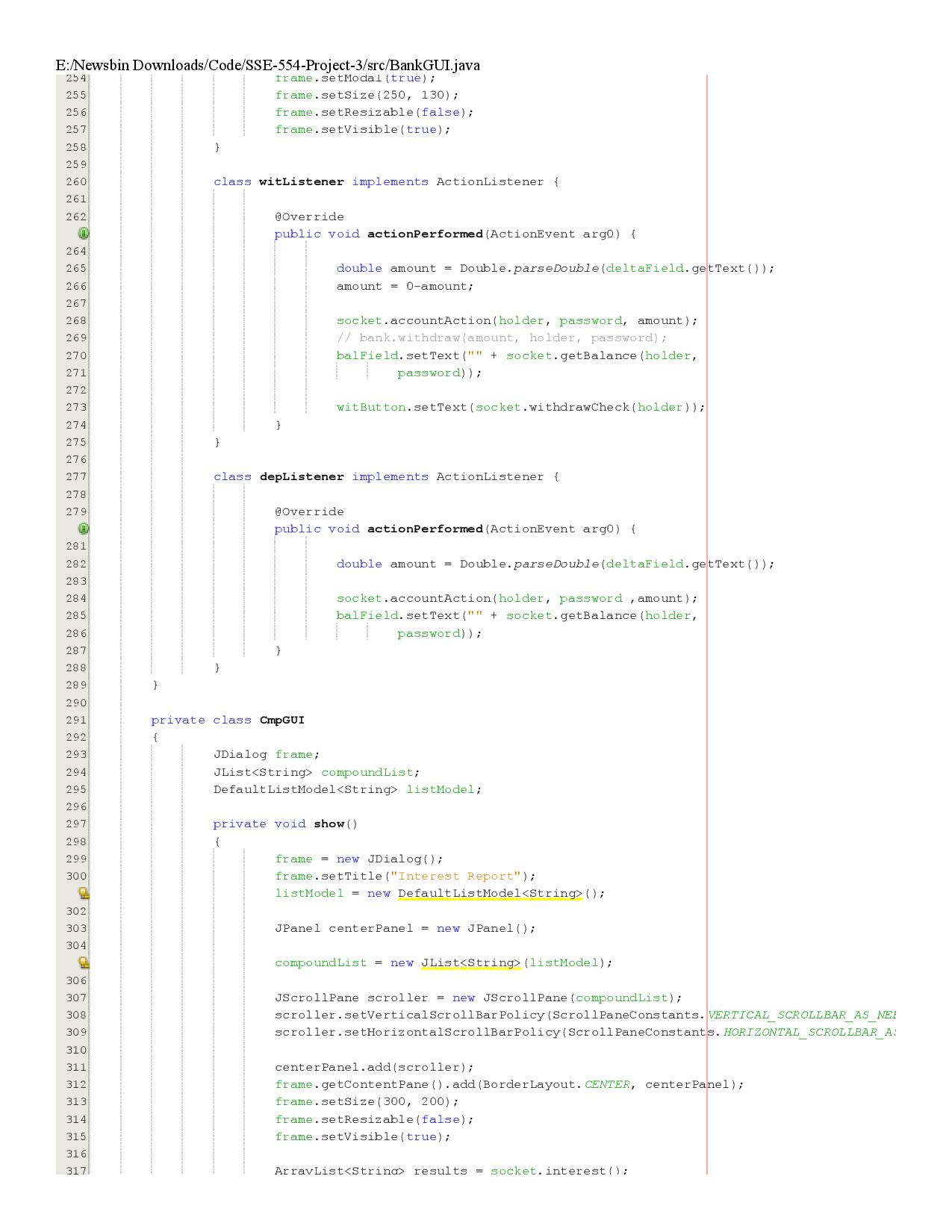


## E:\Newsbin Downloads\Documents\SSE 554 Project4\BankClient\BankClient-page-001.jpgBankClient



## BankGUI





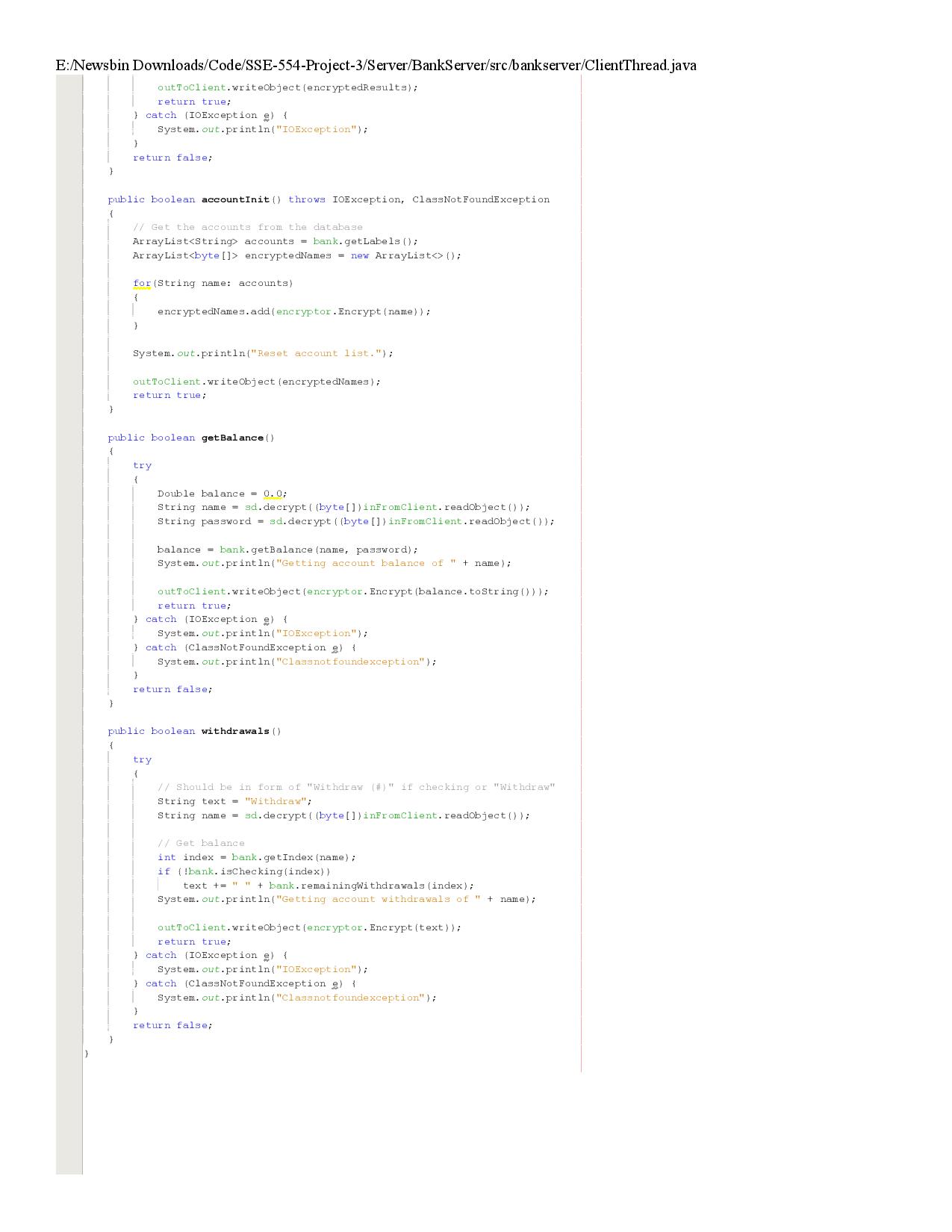
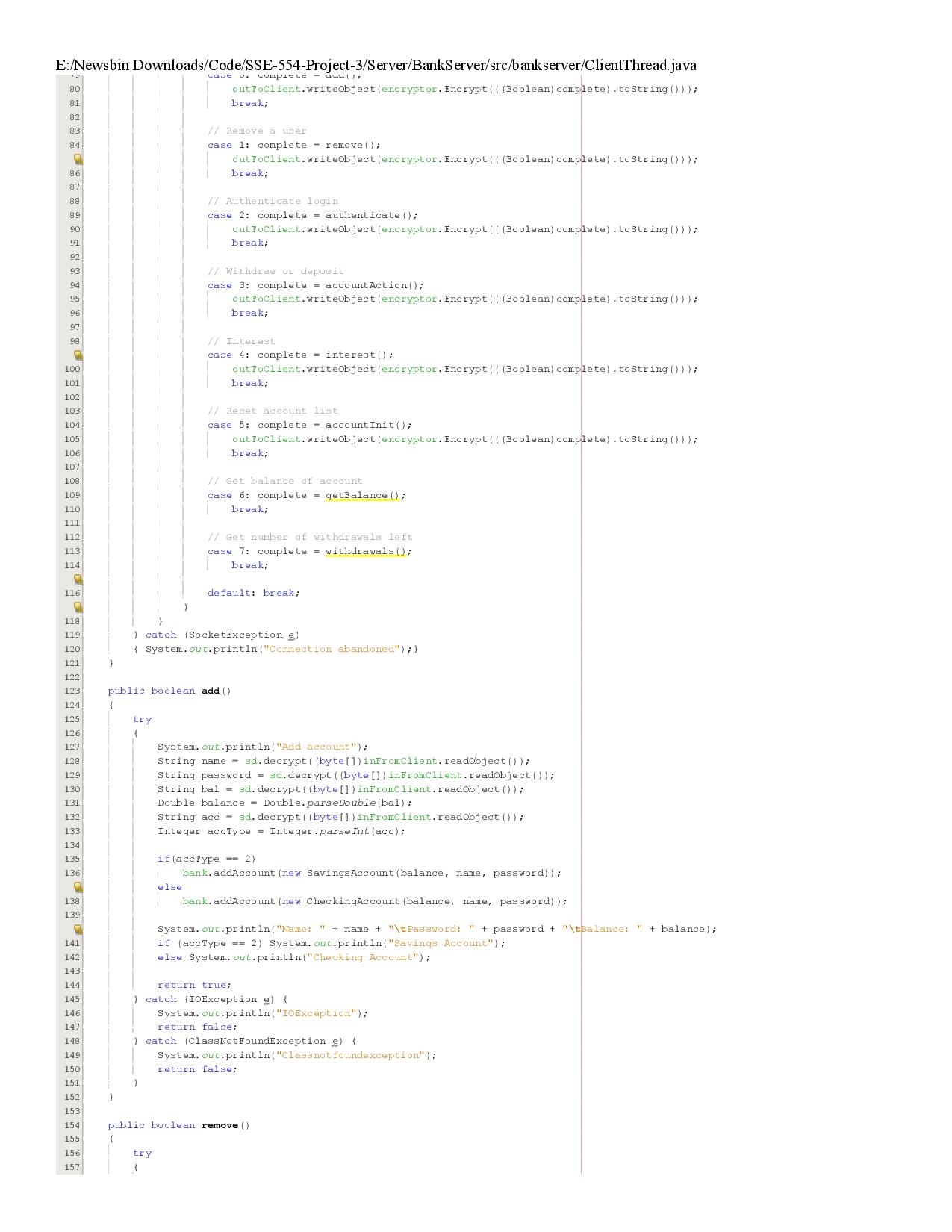
## E:\Newsbin Downloads\Documents\SSE 554 Project3\BankMain\BankMain-page-001.jpgBank\_Main

## BankServer



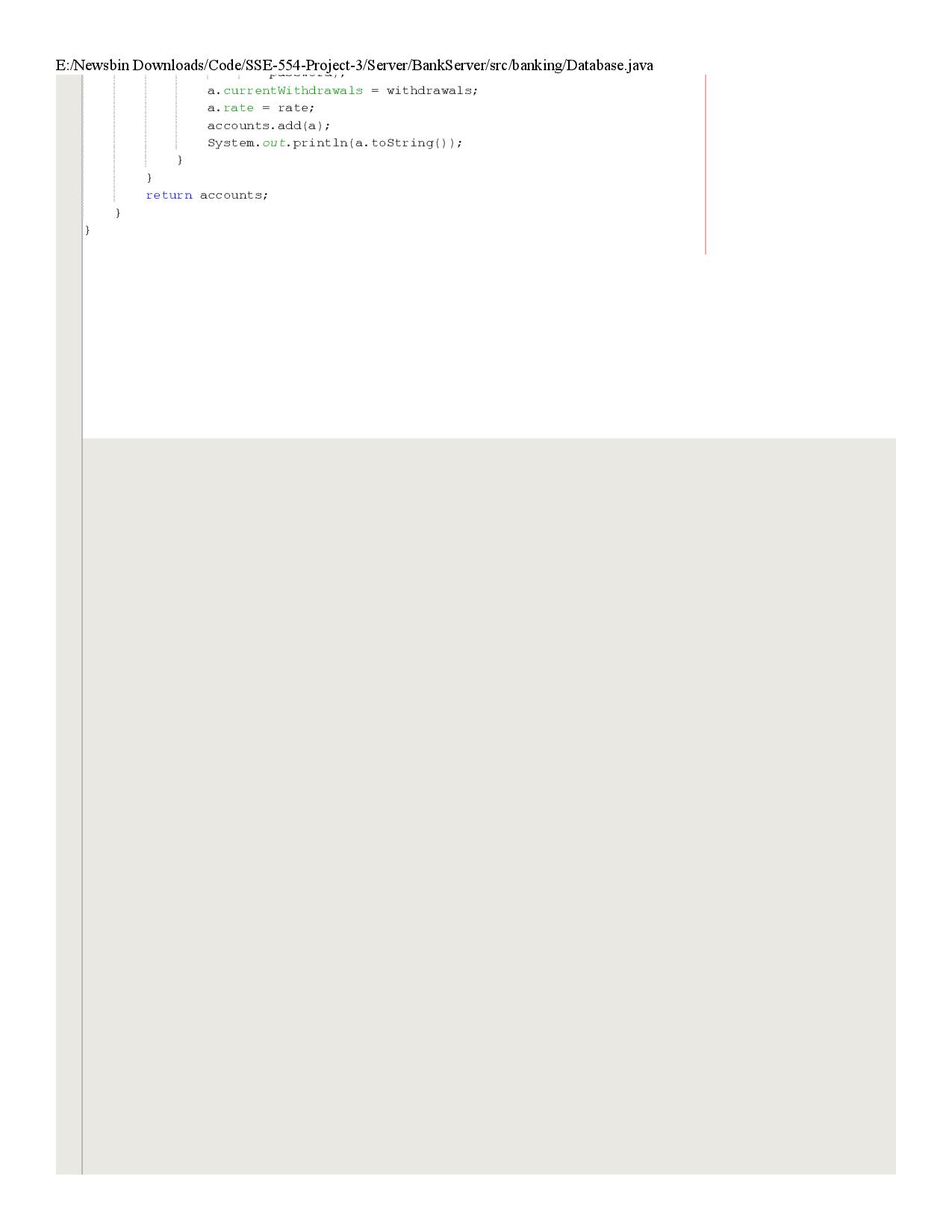
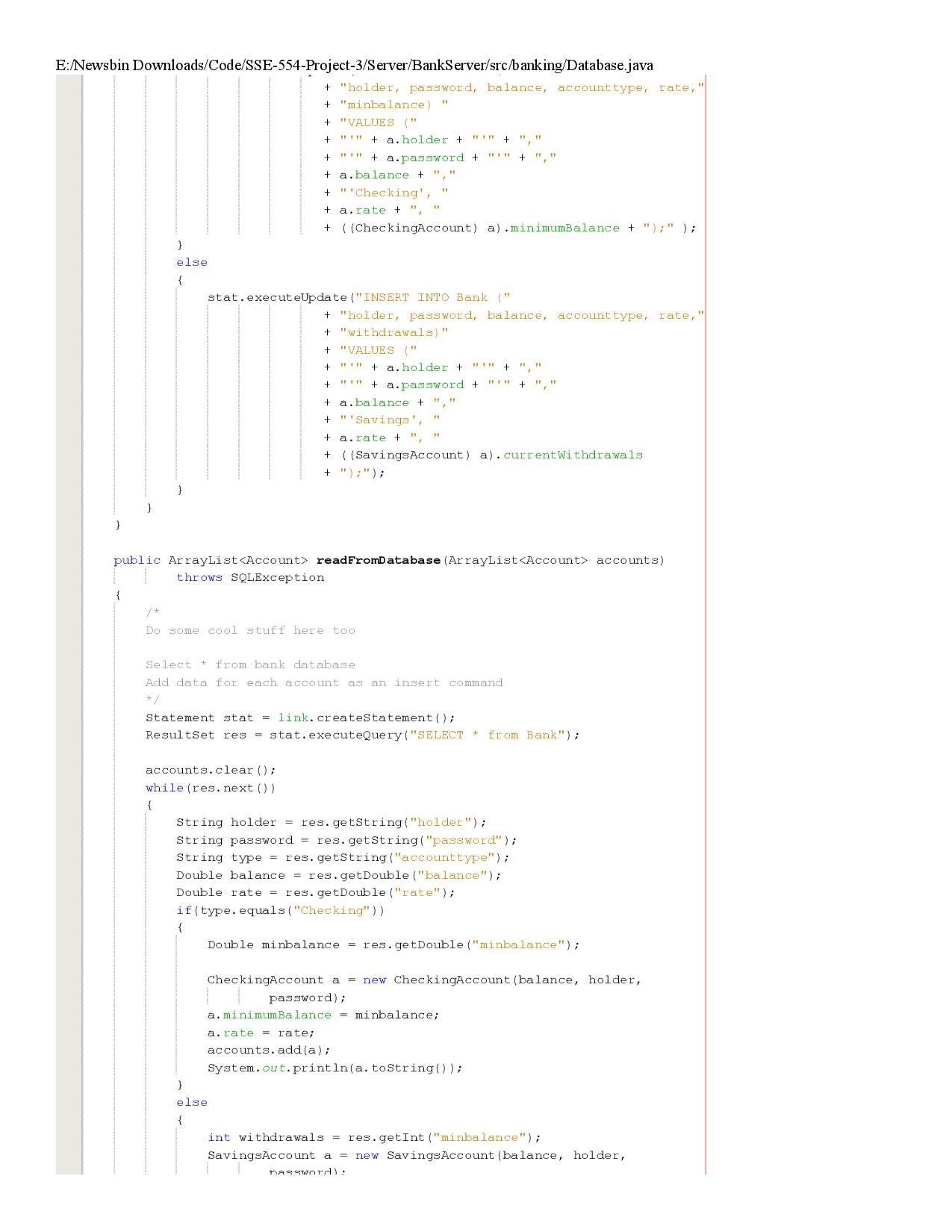
## E:\Newsbin Downloads\Documents\SSE 554 Project3\CheckingAccount\CheckingAccount-page-001.jpgCheckingAccount

## E:\Newsbin Downloads\Documents\SSE 554 Project4\ClientThread\ClientThread-page-001.jpgClientThread



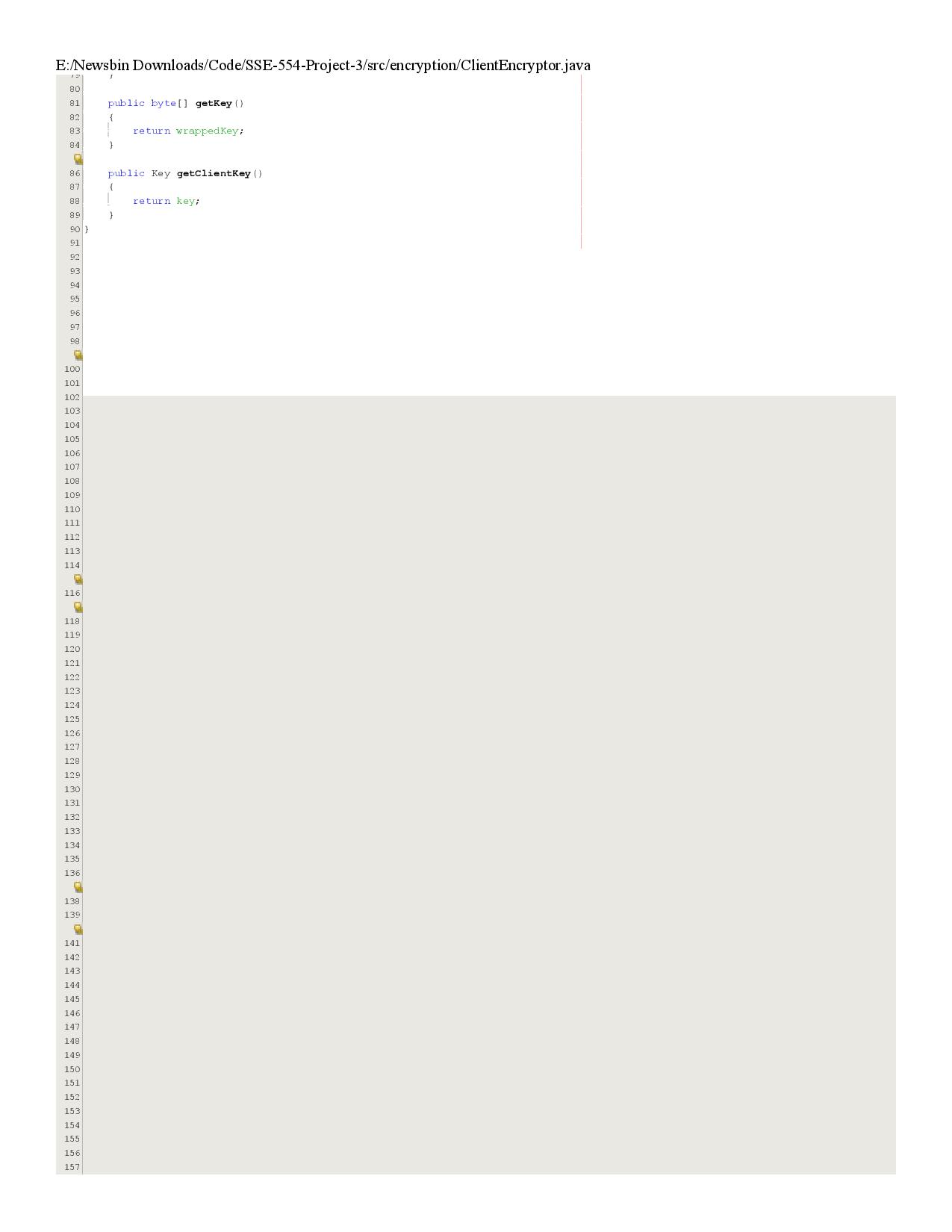
## Database



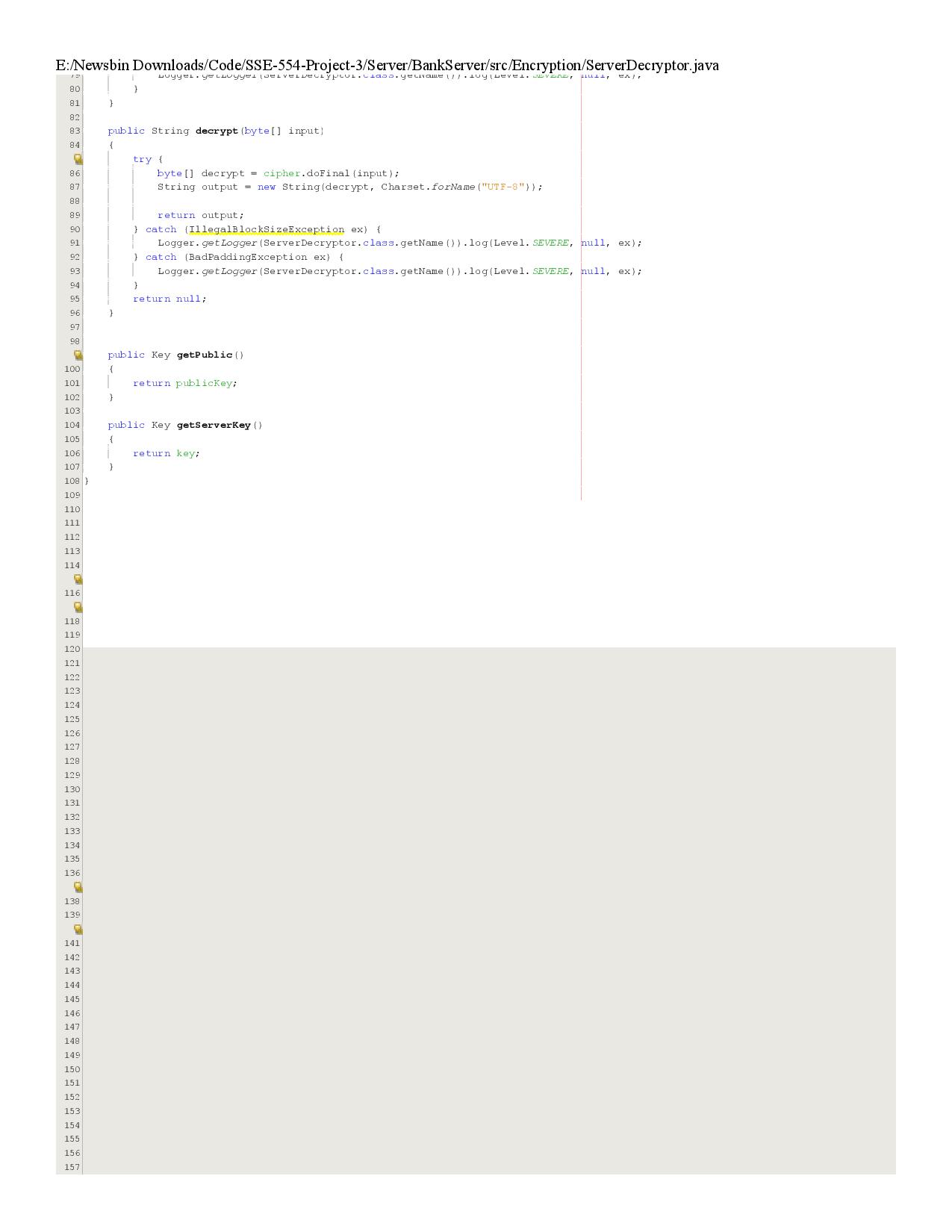


## E:\Newsbin Downloads\Documents\SSE 554 Project3\SavingsAccount\SavingsAccount-page-001.jpgSavingsAccount

## E:\Newsbin Downloads\Documents\SSE 554 Project4\ClientEncrypter\ClientEncrypter-page-001.jpgClient Encryptor



## E:\Newsbin Downloads\Documents\SSE 554 Project4\ServerDecrypter\ServerDecrypter-page-001.jpgServer Decrypter



## E:\Newsbin Downloads\Documents\SSE 554 Project4\EncryptionUtility\EncryptionUtility-page-001.jpgEncryption Utility

