REPORT

SEHH2242

Object Oriented Programming
Group Assignment

ATM Prototype Program

Group Member list:

21117516A Li Wing Sum 102C
21030030A Wan Hoi Nam 102D
21150285A Fung Pui Kiu 102D
21106575A Ng Ka Ki 102B
21107664A Fung Cheuk Hin 102C
21137816A Wong Hiu Chun 102B

Google Drive :
https://drive.google.com/file/d/1U_t7CbkWkWUDRlRucrRrAfJNGY1nBHM7/view?usp=sharing

YouTube :
https://youtu.be/LQUZUTbeUxQ

CONTENTS

PROGRAM CODE

ACCOUNT.JAVA

```java
// Account.java
// Represents a bank account

public class Account
{
    private int accountNumber; // account number
    private int pin; // PIN for authentication
    private double availableBalance; // funds available for withdrawal
    private double totalBalance; // funds available + pending deposits

    // Account constructor initializes attributes
    public Account( int theAccountNumber, int thePIN,
    double theAvailableBalance, double theTotalBalance )
    {
        accountNumber = theAccountNumber;
        pin = thePIN;
        availableBalance = theAvailableBalance;
        totalBalance = theTotalBalance;
    } // end Account constructor

    // determines whether a user-specified PIN matches PIN in Account
    public boolean validatePIN( int userPIN )
    {
        if ( userPIN == pin )
            return true;
        else
            return false;
    } // end method validatePIN

    // returns available balance
    public double getAvailableBalance()
    {
        return availableBalance;
    } // end getAvailableBalance
```

```
    // returns the total balance
    public double getTotalBalance()
    {
        return totalBalance;
    } // end method getTotalBalance

    // credits an amount to the account
    public void credit( double amount )
    {
        totalBalance += amount; // add to total balance
    } // end method credit

    //  add Amount to Transfer Account
    public void transferAdd( double amount )
    {
        availableBalance += amount; // add to available balance
        totalBalance += amount; // add to total balance
    } // end method transferAdd

    // debits an amount from the account
    public void debit( double amount )
    {
        availableBalance -= amount; // subtract from available balance
        totalBalance -= amount; // subtract from total balance
    } // end method debit

    // returns account number
    public int getAccountNumber()
    {
        return accountNumber;
    } // end method getAccountNumber

    public String toString()
    {
        return String.format( "%s\nAvailableBalance: %s\nTotal Balance:
%s\n",
            "Account Type: Account",
            getAvailableBalance(),
```

```
            getTotalBalance());
    } // end method toString

} // end class Account
```

ACCOUNTLOGINSCREEN.JAVA

```
import javax.swing.JTextField;
import javax.swing.JButton;
import java.awt.event.ActionListener;
import javax.swing.JPanel;
import java.awt.event.ActionEvent;


public class accountLoginScreen extends MainScreen implements
ActionListener{
    private static JTextField userText;



    public accountLoginScreen(String getFileName) {
        super(getFileName);
    }

    public void screenCenter(JButton[] centerButton, JPanel panel){
        userText = getTextField("",true,(screenWidth-309)/2, 300, 605/2,
32) ;
        panel.add (userText);
    }//end of screenCenter

    public void waiting() {
        while(waitingEnter) {
            try
            {
                Thread.sleep(1);
            }
            catch (InterruptedException ie)
            {
                ie.printStackTrace();
```

```
            }
        } //end while
    }// end method waiting

    public void actionPerformed( ActionEvent event )
    {
        switch (event.getActionCommand())
        {
            case "0" :

userText.setText(userText.getText().concat( event.getActionCommand() ));
                break;

            case "00" :

userText.setText(userText.getText().concat( event.getActionCommand() ));
                break;

            case "." :

userText.setText(userText.getText().concat( event.getActionCommand() ));
                break;

            case "7" :

userText.setText(userText.getText().concat( event.getActionCommand() ));
                break;

            case "8" :

userText.setText(userText.getText().concat( event.getActionCommand() ));
                break;

            case "9" :

userText.setText(userText.getText().concat( event.getActionCommand() ));
                break;
```

```
            case "6" :
userText.setText(userText.getText().concat( event.getActionCommand() ));
                break;
            case "5" :
userText.setText(userText.getText().concat( event.getActionCommand() ));
                break;
            case "4" :
userText.setText(userText.getText().concat( event.getActionCommand() ));
                break;
            case "3" :
userText.setText(userText.getText().concat( event.getActionCommand() ));
                break;
            case "2" :
userText.setText(userText.getText().concat( event.getActionCommand() ));
                break;
            case "1" :
userText.setText(userText.getText().concat( event.getActionCommand() ));
                break;
            case "Enter":
                try {
                    setAccount(Integer.parseInt(userText.getText()));
                    waitingEnter = false;
                    dispose();
                } catch(NumberFormatException e){
```

```
                }
                break;

            case "Change" :
                userText.setText("");
                break;

            case "Cancel" :
                userText.setText("");
                break;
        }
    } // end method actionPerformed
} //end class accountLoginScreen
```

ATM.JAVA

```
// ATM.java
// Represents an automated teller machine


public class ATM
{
    private boolean userAuthenticated; // whether user is authenticated
    private BankDatabase bankDatabase;
    private CashDispenser cashDispenser;
    private MainScreen getMainScreen;


    // no-argument ATM constructor initializes instance variables
    public ATM()
    {
        userAuthenticated = false; // user is not authenticated to start
        getMainScreen = new MainScreen();
        bankDatabase = getMainScreen.getBankDatabase();
        cashDispenser = getMainScreen.getCashDispenser();

    } // end no-argument ATM constructor
```

```
    // start ATM
    public void run()
    {
        // welcome and authenticate user; perform transactions
        while ( true )
        {
            // loop while user is not yet authenticated
           while ( !userAuthenticated )
            {
                ShowMessageScreen welcomeMessage = new
ShowMessageScreen("welcomeATM.png",false,3,"empty");
                welcomeMessage.setVisible(true);
                welcomeMessage.waiting();
                authenticateUser(); // authenticate user
            } // end while

            performTransactions(); // user is now authenticated
            userAuthenticated = false; // reset before next ATM session
        } // end while
    } // end method run

    // attempts to authenticate user against database
    private void authenticateUser()
    {
        accountLoginScreen loginAccount = new
accountLoginScreen("UserID.png");
        loginAccount.setVisible(true);
        loginAccount.waiting();

        int accountNumber = loginAccount.getAccount();
        loginAccount = null;

        passwordLoginScreen loginPassword = new
passwordLoginScreen("Password.png");
        loginPassword.setVisible(true);
        loginPassword.waiting();
        int pin = loginPassword.getPassword();
```

```
        loginPassword = null;

        // set userAuthenticated to boolean value returned by database
        userAuthenticated =
        bankDatabase.authenticateUser( accountNumber, pin );

        // check whether authentication succeeded
        if ( userAuthenticated )
        {
            getMainScreen.setAccount(accountNumber); // save user's
account #

        } // end if
        else {
            ShowMessageScreen showMessage = new
ShowMessageScreen("WrongLogin.png",false, 3, "empty");
            showMessage.setVisible(true);
            showMessage.waiting();
        }
    } // end method authenticateUser

    // display the main menu and perform transactions
    private void performTransactions()
    {
        boolean userExited = false; // user has not chosen to exit
        MainMenuScreen mainMenu = new MainMenuScreen("MainMenu.png");
        mainMenu.setVisible(true);
        while(!userExited) {
            try
            {
                Thread.sleep(1);
            }
            catch (InterruptedException ie)
            {
                ie.printStackTrace();
            }
            if(getMainScreen.getAccount() == 0) {
                userExited = true;
```

```
            }
         }
      } // end method performTransactions
} // end class ATM
```

ATMCASESTUDY.JAVA

```
// ATMCaseStudy.java
// Driver program for the ATM case study

public class ATMCaseStudy
{
   // main method creates and runs the ATM
   public static void main( String[] args )
   {
      ATM theATM = new ATM();
      theATM.run();
   } // end main
} // end class ATMCaseStudy
```

BALANCEINQUIRY.JAVA

```
// BalanceInquiry.java
// Represents a balance inquiry ATM transaction

public class BalanceInquiry extends Transaction
{
   // BalanceInquiry constructor
   public BalanceInquiry( int userAccountNumber, Screen atmScreen,
      BankDatabase atmBankDatabase )
   {
      super( userAccountNumber, atmScreen, atmBankDatabase );
   } // end BalanceInquiry constructor

   // performs the transaction
   public void execute()
```

```
   {
      // get references to bank database and screen
      BankDatabase bankDatabase = getBankDatabase();
      Screen screen = getScreen();

      // get the available balance for the account involved
      double availableBalance =
         bankDatabase.getAvailableBalance( getAccountNumber() );

      // get the total balance for the account involved
      double totalBalance =
         bankDatabase.getTotalBalance( getAccountNumber() );

      // display the balance information on the screen
      screen.displayMessageLine( "\nBalance Information:" );
      screen.displayMessage( " - Available balance: " );
      screen.displayDollarAmount( availableBalance );
      screen.displayMessage( "\n - Total balance:     " );
      screen.displayDollarAmount( totalBalance );
      screen.displayMessageLine( "" );
   } // end method execute
} // end class BalanceInquiry
```

BANKDATABASE.JAVA

```
// BankDatabase.java
// Represents the bank account information database

public class BankDatabase
{
    private Account accounts[]; // array of Accounts

    // no-argument BankDatabase constructor initializes accounts
    public BankDatabase()
    {
        accounts = new Account[ 5 ]; //  accounts for testing
        accounts[ 0 ] = new Account( 12345, 54321, 1000.0, 1200.0 );
```

```
        accounts[ 1 ] = new Account( 98765, 56789, 200.0, 200.0 );
        accounts[ 2 ] = new SavingAccount(55555, 55555, 2000.0, 1000.0);
        accounts[ 3 ] = new ChequeAccount(66666, 66666, 4000.0, 2000.0);
        accounts[ 4 ] = new Account(1, 1, 14000.0, 14000.0);
    } // end no-argument BankDatabase constructor

    // retrieve Account object containing specified account number
    private Account getAccount( int accountNumber )
    {
        // loop through accounts searching for matching account number
        for ( Account currentAccount : accounts )
        {
            // return current account if match found
            if ( currentAccount.getAccountNumber() == accountNumber )
                return currentAccount;
        } // end for

        return null; // if no matching account was found, return null
    } // end method getAccount

    public Boolean checkAccountExist(int accountNumber)
    {
        if(getAccount(accountNumber) != null) {
            return true;
        } else {
            return false;
        }
    }// end method checkAccountExist

    // determine whether user-specified account number and PIN match
    // those of an account in the database
    public boolean authenticateUser( int userAccountNumber, int userPIN )
    {
        // attempt to retrieve the account with the account number
        Account userAccount = getAccount( userAccountNumber );

        // if account exists, return result of Account method validatePIN
        if ( userAccount != null )
```

```
            return userAccount.validatePIN( userPIN );
        else
            return false; // account number not found, so return false
    } // end method authenticateUser

    // return available balance of Account with specified account number
    public double getAvailableBalance( int userAccountNumber )
    {
        return getAccount( userAccountNumber ).getAvailableBalance();
    } // end method getAvailableBalance

    // return total balance of Account with specified account number
    public double getTotalBalance( int userAccountNumber )
    {
        return getAccount( userAccountNumber ).getTotalBalance();
    } // end method getTotalBalance

    // credit an amount to Account with specified account number
    public void credit( int userAccountNumber, double amount )
    {
        getAccount( userAccountNumber ).credit( amount );
    } // end method credit

    // add Amount to Transfer Account
    public void transferAdd( int userAccountNumber, double amount )
    {
        getAccount( userAccountNumber ).transferAdd( amount );
    } // end method add Amount to Transfer Account

    // debit an amount from of Account with specified account number
    public void debit( int userAccountNumber, double amount )
    {
        getAccount( userAccountNumber ).debit( amount );
    } // end method debit
} // end class BankDatabase
```

CASHDISPENSER.JAVA

```java
// CashDispenser.java
// Represents the cash dispenser of the ATM

public class CashDispenser
{
   // the default initial number of bills in the cash dispenser
   private final static int INITIAL_COUNT = 1;
   private int count; // number of $20 bills remaining

   // no-argument CashDispenser constructor initializes count to default
   public CashDispenser()
   {
      count = INITIAL_COUNT; // set count attribute to default
   } // end CashDispenser constructor

   // simulates dispensing of specified amount of cash
   public void dispenseCash( int amount )
   {
      int billsRequired = amount / 20; // number of $20 bills required
      count -= billsRequired; // update the count of bills
   } // end method dispenseCash

   // indicates whether cash dispenser can dispense desired amount
   public boolean isSufficientCashAvailable( int amount )
   {
      int billsRequired = amount / 20; // number of $20 bills required

      if ( count >= billsRequired  )
         return true; // enough bills available
      else
         return false; // not enough bills available
   } // end method isSufficientCashAvailable
} // end class CashDispenser
```

CHEQUEACCOUNT.JAVA

```
// ChequeAccount.java
// Represents a bank cheque account
public class ChequeAccount extends Account
{
    // instance variables
    private double eachChequeLimit;

    // Cheque Account constructor initializes attributes
    public ChequeAccount(int theAccountNumber, int thePIN,
    double theAvailableBalance, double theTotalBalance)
    {
        super( theAccountNumber, thePIN, theAvailableBalance,
theTotalBalance);
        // initialise instance variables
        setEachChequeLimit(10000.0);
    } // end ChequeAccount constructor

    //set Each Cheque Limit
    public void setEachChequeLimit(double limitValue)
    {
        eachChequeLimit = limitValue;
    } //end method setEachChequeLimit

    //return double Each Cheque Limit
    public double getEachChequeLimit()
    {
        return eachChequeLimit;
    } //end method getEachChequeLimit

    //override toString method
    @Override
    public String toString()
    {
        return String.format("%s\nAvailableBalance: %s\nTotal Balance:
%s\nEach Cheque Limit:%s\n",
            "Account type: Cheque Account",
```

```
            getAvailableBalance(),
            getTotalBalance(),
            getEachChequeLimit());
    }
} //end public class ChequeAccount
```

CONFIRMATION.JAVA

```
import javax.swing.JTextField;
import javax.swing.JPanel;
import javax.swing.JButton;
import java.awt.event.ActionEvent;


public class Confirmation extends MainScreen {
    private JTextField confirmUserID;
    private JTextField confirmAmount;
    private int transferAccount;
    private double amount;
    //Confirmation constructor initializes attributes
    public Confirmation(String getFileName,int userID,double tranAmount) {
        super(getFileName);
        confirmUserID.setText(Integer.toString(userID));
        confirmAmount.setText(Double.toString(tranAmount));
        transferAccount = userID;
        amount = tranAmount;
    }// end Confirmation constructor

    public void screenCenter(JButton[] centerButton, JPanel panel){

        confirmUserID = getTextField("",false,(screenWidth)/2+45, 167,
240, 32);
        panel.add (confirmUserID);

        confirmAmount = getTextField("",false,(screenWidth)/2+45, 265,
240, 32);
        panel.add (confirmAmount);
```

```
    }//end method screenCenter

    private void execute() {
        double availableBalance; // amount available for withdrawal
        if( transferAccount == getAccount()) {
            new ShowMessageScreen("AccountOrBalanceError.png",false,3); //
Cannot transfer to same account
            dispose();
        } else {
            if(bankDatabase.checkAccountExist(transferAccount)) {
                // get available balance of account involved
                availableBalance =
                bankDatabase.getAvailableBalance( getAccount() );

                // check whether the user has enough money in the account
                if ( amount <= availableBalance && amount > 0)
                {
                    // update the account involved to reflect withdrawal
                    bankDatabase.debit( getAccount(), amount );
                    bankDatabase.transferAdd( transferAccount, amount );

                    new
ShowMessageScreen("TransferSuccessful.png",false,3); //Successfully
transferred

                dispose();
                } // end if
                else // not enough money available in user's account
                {
                new
ShowMessageScreen("AccountOrBalanceError.png",false,3); //Insufficient
funds in your account.
                dispose();
                } // end else
            } else {
                //transfer account not exist
```

```
                new
ShowMessageScreen("AccountOrBalanceError.png",false,3); //Transfer account
does not exist. Please try again.
                dispose();
            } //end else
        } //end else
    } //end method execute

    public void actionPerformed( ActionEvent event )
    {
        switch (event.getActionCommand())
        {
            case "Enter" :
                execute();
                break;
            case "Change" :
                new Thread(() -> new TransferFunction()).start();
                dispose();
                break;
            case "Cancel":
                new MainMenuScreen("MainMenu.png");
                dispose();
                break;
        }
    } // end method actionPerformed
} // end class Confirmation
```

CONFIRMWITHDRAWALSCREEN.JAVA

```
import javax.swing.JTextField;
import javax.swing.JButton;
import java.awt.event.ActionListener;
import javax.swing.JPanel;
import java.awt.event.ActionEvent;
```

```
public class confirmWithdrawalScreen extends MainScreen implements
ActionListener{
    private JTextField withdrawlAmountText;
    private int withdrawlAmount;

    //confirmWithdrawalScreen constructor initializes attributes
    public confirmWithdrawalScreen(String getFileName, int
withdrawalAmount) {
        super(getFileName);
        this.withdrawlAmount = withdrawalAmount;
        withdrawlAmountText.setText(Integer.toString(withdrawlAmount));
    } // end confirmWithdrawalScreen constructor

    public void screenCenter(JButton[] centerButton, JPanel panel){
        withdrawlAmountText = getTextField("",true,(screenWidth-309)/2,
300, 605/2, 32) ;
        panel.add (withdrawlAmountText);
    } // end method screenCenter

    public void waiting() {
        while(waitingEnter) {
            try
            {
            Thread.sleep(1);
            }
            catch (InterruptedException ie)
            {
                ie.printStackTrace();
            }
        }
    }// end method waiting

    public void actionPerformed( ActionEvent event )
    {

        switch (event.getActionCommand())
        {
            case "0" :
```

```
withdrawlAmountText.setText(withdrawlAmountText.getText().concat( event.ge
tActionCommand() ));
                break;

            case "00" :

withdrawlAmountText.setText(withdrawlAmountText.getText().concat( event.ge
tActionCommand() ));
                break;

            case "." :

withdrawlAmountText.setText(withdrawlAmountText.getText().concat( event.ge
tActionCommand() ));
                break;

            case "7" :

withdrawlAmountText.setText(withdrawlAmountText.getText().concat( event.ge
tActionCommand() ));
                break;

            case "8" :

withdrawlAmountText.setText(withdrawlAmountText.getText().concat( event.ge
tActionCommand() ));
                break;

            case "9" :

withdrawlAmountText.setText(withdrawlAmountText.getText().concat( event.ge
tActionCommand() ));
                break;

            case "6" :
```

```
withdrawlAmountText.setText(withdrawlAmountText.getText().concat( event.ge
tActionCommand() ));
                break;

            case "5" :

withdrawlAmountText.setText(withdrawlAmountText.getText().concat( event.ge
tActionCommand() ));
                break;

            case "4" :

withdrawlAmountText.setText(withdrawlAmountText.getText().concat( event.ge
tActionCommand() ));
                break;

            case "3" :

withdrawlAmountText.setText(withdrawlAmountText.getText().concat( event.ge
tActionCommand() ));
                break;

            case "2" :

withdrawlAmountText.setText(withdrawlAmountText.getText().concat( event.ge
tActionCommand() ));
                break;

            case "1" :
withdrawlAmountText.setText(withdrawlAmountText.getText().concat( event.ge
tActionCommand() ));
                break;

            case "Enter":

                try {
                    // update the account involved to reflect withdrawal
                    bankDatabase.debit( getAccount(), withdrawlAmount );
```

```
                    cashDispenser.dispenseCash( withdrawlAmount ); //
dispense cash

                    new ShowMessageScreen("takeCard.png",false,3,
"withdrawlComplete1"); //Please take your cash now.
                    dispose();

                } catch(NumberFormatException e){

                }
                 break;

            case "Change" :
                withdrawlAmountText.setText("");
                break;

            case "Cancel" :
                new MainMenuScreen("MainMenu.png");
                dispose();
                break;
        }
    } // end method actionPerformed
} //end class confirmWithdrawalScreen
```

CUSTOMWITHDRAWALSCREEN.JAVA

```
import java.awt.event.ActionListener;
import javax.swing.JTextField;
import javax.swing.JButton;
import javax.swing.JPanel;
import java.awt.event.ActionEvent;

public class CustomWithdrawalScreen extends MainScreen implements
ActionListener{
    private static JTextField amountText;
    private boolean waitingEnter = true;
```

```
    public CustomWithdrawalScreen(String getFileName) {
        super(getFileName);
    }

    public String getAmount() {
        return amountText.getText();
    }

    public void execute()
    {
         int withdrawlAmount=0;
        double availableBalance; // amount available for withdrawal
        // get available balance of account involved
        availableBalance =
        bankDatabase.getAvailableBalance( getAccount() );
        try{
      withdrawlAmount = Integer.parseInt(getAmount());
    }
catch (NumberFormatException e) {
    }
        if(withdrawlAmount % 100 == 0) {
            // check whether the user has enough money in the account
            if ( withdrawlAmount <= availableBalance && withdrawlAmount >
0)
            {
            // check whether the cash dispenser has enough money
                if
( cashDispenser.isSufficientCashAvailable( withdrawlAmount ) )
                {
                new
confirmWithdrawalScreen("ConfirmWithdrawal.png",withdrawlAmount);
//Insufficient cash available in the ATM.
                    dispose();
                } // end if
                else {// cash dispenser does not have enough cash
```

```
                    new
ShowMessageScreen("ATMInsufficientCash.png",false,3); //Insufficient cash
available in the ATM.
                    dispose();
                }
            } // end if
            else // not enough money available in user's account
            {
                new
ShowMessageScreen("AccountOrBalanceError.png",false,3); //Insufficient
funds in your account.
                dispose();
            } // end else
        } else {
            new ShowMessageScreen("ErrorWithdrawal.png",false,3); //Only
the multiples of HKD100, HKD500, or HKD1000 are allowed. Try again.
            dispose();
        }
    } // end method execute

    public void screenCenter(JButton[] centerButton, JPanel panel){
        amountText = getTextField("",true,(screenWidth-309)/2, 300, 605/2,
32);
        panel.add (amountText);
    } //end method screenCenter

    public void waiting() {
        while(waitingEnter) {
            try
            {
                Thread.sleep(1);
            }
            catch (InterruptedException ie)
            {
                ie.printStackTrace();
            }
        }
    }// end method waiting
```

```
    public void actionPerformed( ActionEvent event )
    {

        switch (event.getActionCommand())
        {
            case "0" :

amountText.setText(amountText.getText().concat( event.getActionCommand() )
);
            break;

            case "00" :

amountText.setText(amountText.getText().concat( event.getActionCommand() )
);
            break;

             case "." :

amountText.setText(amountText.getText().concat( event.getActionCommand() )
);
                break;

            case "7" :

amountText.setText(amountText.getText().concat( event.getActionCommand() )
);
                break;

            case "8" :
amountText.setText(amountText.getText().concat( event.getActionCommand() )
);
                break;

            case "9" :
```

```
amountText.setText(amountText.getText().concat( event.getActionCommand() )
);
                break;

            case "6" :
amountText.setText(amountText.getText().concat( event.getActionCommand() )
);
                break;

            case "5" :

amountText.setText(amountText.getText().concat( event.getActionCommand() )
);
                break;

            case "4" :

amountText.setText(amountText.getText().concat( event.getActionCommand() )
);
                break;

            case "3" :

amountText.setText(amountText.getText().concat( event.getActionCommand() )
);
                break;

            case "2" :

amountText.setText(amountText.getText().concat( event.getActionCommand() )
);
                break;

            case "1" :

amountText.setText(amountText.getText().concat( event.getActionCommand() )
);
                break;
```

```
            case "Enter":
                //call withdrawl function here
                execute();
                waitingEnter = false;
                dispose();
                break;

            case "Change" :
                amountText.setText("");
                break;

            case "Cancel" :
                new MainMenuScreen("MainMenu.png");
                dispose();
                break;
        }
    } // end method actionPerformed
} // end class CustomWithdrawalScreen
```

FIXEDAMOUNTWITHDRAWALSCREEN.JAVA

```
import javax.swing.JButton;
import java.awt.event.ActionListener;
import javax.swing.JPanel;
import java.awt.event.ActionEvent;


public class FixedAmountWithdrawalScreen extends MainScreen implements
ActionListener{
 private static int withdrawlAmount;
    //FixedAmountWithdrawalScreen constructor initializes attributes
    public FixedAmountWithdrawalScreen(String getFileName, int getAmount)
{
    super(getFileName);
      withdrawlAmount = getAmount;
```

```
    } // end FixedAmountWithdrawalScreen constructor

    public void screenCenter(JButton[] centerButton, JPanel panel){
    }

    public void execute()
    {
        double availableBalance; // amount available for withdrawal
        // get available balance of account involved
        availableBalance =
        bankDatabase.getAvailableBalance( getAccount() );

        // check whether the user has enough money in the account
        if ( withdrawlAmount <= availableBalance && withdrawlAmount > 0)
        {
   // check whether the cash dispenser has enough money
   if ( cashDispenser.isSufficientCashAvailable( withdrawlAmount ) )
      {
                // update the account involved to reflect withdrawal
               bankDatabase.debit( getAccount(), withdrawlAmount );

               cashDispenser.dispenseCash( withdrawlAmount ); // dispense
cash

               new ShowMessageScreen("takeCard.png",false,3,
"withdrawlComplete1"); //Please take your cash now.
               dispose();
            } // end if
            else {// cash dispenser does not have enough cash
               new ShowMessageScreen("ATMInsufficientCash.png",false,3);
//Insufficient cash available in the ATM.
               dispose();
            }
            } // end if
            else // not enough money available in user's account
            {
```

```
                new
ShowMessageScreen("AccountOrBalanceError.png",false,3); //Insufficient
funds in your account.
                dispose();
            } // end else
        } // end method execute

        public void actionPerformed( ActionEvent event )
        {
            switch (event.getActionCommand())
            {
                case "Enter":
                    //call withdrawl function here
                    execute();
                    break;
                case "Change" :
                    new WithdrawalMenuScreen("WithdrawalMenu.png");
                    dispose();
                    break;

                case "Cancel" :
                    new MainMenuScreen("MainMenu.png");
                    dispose();
                    break;
        }
    } // end method actionPerformed
}// end class FixedAmountWithdrawalScreen
```

KEYPAD.JAVA

```
// Keypad.java
// Represents the keypad of the ATM
import java.util.Scanner; // program uses Scanner to obtain user input

public class Keypad
{
   private Scanner input; // reads data from the command line
```

```
   // no-argument constructor initializes the Scanner
   public Keypad()
   {
      input = new Scanner( System.in );
   } // end no-argument Keypad constructor

   // return an integer value entered by user
   public int getInput()
   {
      return input.nextInt(); // we assume that user enters an integer
   } // end method getInput

   // return an integer value entered by user
   public double getDoubleInput()
   {
      return input.nextDouble(); // we assume that user enters an integer
   } // end method getInput
} // end class Keypad
```

MAINMENU.JAVA

```
import javax.swing.JButton;
import java.awt.event.ActionListener;
import javax.swing.JPanel;
import java.awt.event.ActionEvent;


public class MainMenuScreen extends MainScreen implements ActionListener{

    public MainMenuScreen(String getFileName) {
        super(getFileName);
    }

    public void screenCenter(JButton[] centerButton, JPanel panel){
    } //end of screenCenter
```

```
    public void actionPerformed( ActionEvent event )
    {
        switch (event.getActionCommand())
        {
            case "L2" :
                new WithdrawalMenuScreen("WithdrawalMenu.png");
                dispose();
                break;
            case "L3":
                new ViewBalanceScreen("ViewBalance.png");
                dispose();
                break;
            case "R2":
                new Thread(() -> new TransferFunction()).start();
                dispose();
                break;
            case "R3":
                new ShowMessageScreen("CardEjected.png",false, 2,
"logout");
                dispose();
                break;

        }
    } // end method actionPerformed
}// end class MainMenuScreen
```

MAINSCREEN.JAVA

```
import javax.swing.JFrame;
import javax.swing.JButton;
import javax.swing.JPanel;
import java.awt.BorderLayout;
import javax.swing.JLabel;
import javax.swing.ImageIcon;
import java.awt.event.ActionListener;
import javax.swing.JTextField;
import java.awt.Color;
```

```
import java.awt.Font;
import javax.swing.JPasswordField;
import javax.swing.JOptionPane;
import java.awt.event.ActionEvent;

public class MainScreen extends JFrame implements ActionListener{

    protected static JButton leftButton[] = new JButton[4];
    protected static JButton centerButton[] = new JButton[1];
    protected static JButton rightButton[] = new JButton[4];
    protected static JButton bottomOfCenterButton[] = new JButton[16];
    protected static BankDatabase bankDatabase = new BankDatabase();
    protected static CashDispenser cashDispenser = new CashDispenser();
    protected boolean waitingEnter = true;

    protected static int userAccount = 0;
    private static int userPassword = 0;
    public final int screenWidth = 1024;
    public final int screenHeight = 768;


    public BankDatabase getBankDatabase() {
        return bankDatabase;
    }

    public CashDispenser getCashDispenser() {
        return cashDispenser;
    }

    public int getPassword() {
        return userPassword;
    }

    public int getAccount() {
        return userAccount;
    }

    public void setPassword(int pass) {
```

```
        userPassword = pass;
    }

    public void setAccount(int ac) {
        userAccount = ac;
    }

    public void logout() {
        userAccount = 0;
        userPassword = 0;
    }

    public void screenCenter(JButton[] centerButton, JPanel panel) {
    }

    public MainScreen() {
    }

    public MainScreen(String backgroundFileName) {
        setResizable(false);
        setSize (screenWidth, screenHeight);
        setVisible(true);
         JPanel panel = new JPanel ();

         panel.setLayout(new BorderLayout());
        getLeftButton(leftButton,panel);
        getRightButton(rightButton,panel);
        getBottomOfCenterButton(bottomOfCenterButton,panel);
        screenCenter(centerButton,panel);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

        JLabel background=new JLabel(new ImageIcon(backgroundFileName));
        panel.add(background);
        add(panel);
    }

    public void getLeftButton(JButton[] leftButton, JPanel panel){
        for ( int count = 0; count < 3; count++ )
```

```
        {
            leftButton[count] = new JButton( ">>" );

leftButton[count].setActionCommand("L"+Integer.toString(count+1));
            leftButton[count].addActionListener(this);
            leftButton[count].setBounds (0, 95*(count+1)+35, 180, 50);
            panel.add(leftButton[count]);
        } // end for
    }

    public void getRightButton(JButton[] rightButton, JPanel panel){
        for ( int count = 0; count < 3; count++ )
        {
            rightButton[count] = new JButton( "<<" );

rightButton[count].setActionCommand("R"+Integer.toString(count+1));
            rightButton[count].setBounds(screenWidth-180-15,
95*(count+1)+35, 180, 50);
            rightButton[count].addActionListener(this);
            panel.add(rightButton[count]);
        } // end for
    }

    public void getBottomOfCenterButton(JButton[] bottomOfCenterButton,
JPanel panel){
        String
keypad[]={"0","00",".","Enter","7","8","9","Change","4","5","6","Cancel","
1","2","3",""};
        int buttonCount = 0;
        for ( int runTimes = 0; runTimes < 4; runTimes++) {
            for ( int count = 0; count < 4; count++ )
            {
                bottomOfCenterButton[buttonCount] = new
JButton( keypad[buttonCount] );
                bottomOfCenterButton[buttonCount].setBounds
(screenWidth/2-160+80*(count), screenHeight-30*(runTimes+2)-100, 80, 25);
                bottomOfCenterButton[buttonCount].addActionListener(this);
                panel.add(bottomOfCenterButton[buttonCount]);
```

```
                buttonCount++;
            } // end for
        }//end for
    }// end method getBottomOfCenterButton

    public JTextField getTextField(String defaultString, boolean editable,
int bound_a, int bound_b, int bound_c, int bound_d) {
        JTextField genTextField = new JTextField(defaultString,20);
        genTextField.setEditable( editable );
        genTextField.setBackground(new Color(0,0,0,80));
        genTextField.setOpaque(false);
        genTextField.setForeground(Color.WHITE);

        Font fontSetting = new Font("SansSerif", Font.BOLD, 28);
        genTextField.setFont(fontSetting);
        genTextField.setBounds (bound_a, bound_b, bound_c, bound_d);
        return genTextField;
    } //end method getTextField

    public JPasswordField getPasswordTextField(String defaultString,
boolean editable, int bound_a, int bound_b, int bound_c, int bound_d) {
        JPasswordField getPasswordTextField = new
JPasswordField(defaultString,20);
        getPasswordTextField.setEditable( editable );
        getPasswordTextField.setBackground(new Color(0,0,0,80));
        getPasswordTextField.setOpaque(false);
        getPasswordTextField.setForeground(Color.WHITE);

        Font fontSetting = new Font("SansSerif", Font.BOLD, 28);
        getPasswordTextField.setFont(fontSetting);
        getPasswordTextField.setBounds (bound_a, bound_b, bound_c,
bound_d);
        return getPasswordTextField;
    } // end method getPasswordTextField

    public void show(String text) {
        JOptionPane.showMessageDialog( null, text );
    } //end method show
```

```
    public void actionPerformed( ActionEvent event )
    {
        JOptionPane.showMessageDialog( null, "You are in MainScreen." );
    } // end method actionPerformed
} // end class MainScreen
```

PASSWORDLOGINSCREEN.JAVA

```
import javax.swing.JPasswordField;
import java.awt.event.ActionListener;
import javax.swing.JButton;
import javax.swing.JPanel;
import java.awt.event.ActionEvent;

public class passwordLoginScreen extends MainScreen implements
ActionListener{
    private JPasswordField passwordText;

    public passwordLoginScreen(String getFileName) {
        super(getFileName);
    }

    public void screenCenter(JButton[] centerButton, JPanel panel){
        passwordText = getPasswordTextField("",true,(screenWidth-309)/2,
300, 605/2, 32);
        panel.add (passwordText);
    }//end of screenCenter

    public void waiting() {
        while(waitingEnter) {
            try
            {
                Thread.sleep(1);
            }
            catch (InterruptedException ie)
            {
```

```
                ie.printStackTrace();
            }
        }
    }// end method waiting

    public void actionPerformed( ActionEvent event )
    {
        switch (event.getActionCommand())
        {
            case "0" :

passwordText.setText(passwordText.getText().concat( event.getActionCommand
() ));
                break;

            case "00" :

passwordText.setText(passwordText.getText().concat( event.getActionCommand
() ));
                break;

            case "." :

passwordText.setText(passwordText.getText().concat( event.getActionCommand
() ));
                break;

            case "7" :

passwordText.setText(passwordText.getText().concat( event.getActionCommand
() ));
                 break;

            case "8" :

passwordText.setText(passwordText.getText().concat( event.getActionCommand
() ));
                break;
```

```
            case "9" :
passwordText.setText(passwordText.getText().concat( event.getActionCommand
() ));
                break;
            case "6" :
passwordText.setText(passwordText.getText().concat( event.getActionCommand
() ));
                break;
            case "5" :
passwordText.setText(passwordText.getText().concat( event.getActionCommand
() ));
                break;
            case "4" :
passwordText.setText(passwordText.getText().concat( event.getActionCommand
() ));
                break;
            case "3" :
passwordText.setText(passwordText.getText().concat( event.getActionCommand
() ));
                break;
           case "2" :
passwordText.setText(passwordText.getText().concat( event.getActionCommand
() ));
                break;
            case "1" :
```

```
                
passwordText.setText(passwordText.getText().concat( event.getActionCommand
() ));
                 break;

            case "Enter":

                 try {
                      setPassword(Integer.parseInt(passwordText.getText()));
                      waitingEnter = false;
                      dispose();
                 } catch(NumberFormatException e){

                 }

                 break;

            case "Change" :
                 passwordText.setText("");
                 break;

            case "Cancel" :
                 passwordText.setText("");
                 break;
        }// end switch statement
    } // end method actionPerformed
} // end class passwordLoginScreen
```

SAVINGACCOUNT.JAVA

```
// ChequeAccount.java
// Represents a bank saving account
public class SavingAccount extends Account
{
    // instance variables
    private double interestRate;
```

```
    // Saving Account constructor initializes attributes
    public SavingAccount(int theAccountNumber, int thePIN,
    double theAvailableBalance, double theTotalBalance)
    {
        super( theAccountNumber,  thePIN,theAvailableBalance,
theTotalBalance); // pass to Account constructor
        setInterestRate(0.1);
    } // end SavingAccount constructor

    //return double Interest Rate
    public double getInterestRate()
    {
        return interestRate;
    } // end method getInterestRate

    //set Interest Rate
    public void setInterestRate( double interestInput )
    {
        interestRate= interestInput/100;//interest rate per annum
    } //end method setInterestRate

    @Override
    public String toString()
    {
        return String.format("%s\nAvailableBalance: %s\nTotal Balance:
%s\nInterest Rate:%s%%\n",
            "Account type: Saving Account",
            getAvailableBalance(),
            getTotalBalance(),
            getInterestRate()*100);
    } // end method toString
} //end public class SavingAccount
```

SCREEN.JAVA

```
// Screen.java
// Represents the screen of the ATM
```

```
public class Screen
{
   // displays a message without a carriage return
   public void displayMessage( String message )
   {
      System.out.print( message );
   } // end method displayMessage

   // display a message with a carriage return
   public void displayMessageLine( String message )
   {
      System.out.println( message );
   } // end method displayMessageLine

   // display a dollar amount
   public void displayDollarAmount( double amount )
   {
      System.out.printf( "$%,.2f", amount );
   } // end method displayDollarAmount
} // end class Screen
```

SHOWMESSAGESCREEN.JAVA

```
import javax.swing.JPanel;
import javax.swing.JLabel;
import javax.swing.JTextField;
import javax.swing.JPasswordField;
import javax.swing.JButton;
import java.awt.event.ActionListener;
import java.awt.event.ActionEvent;
import java.awt.BorderLayout;
import javax.swing.JOptionPane;
import java.awt.GridLayout;
import javax.swing.ImageIcon;
import javax.swing.Icon;
import javax.swing.JFrame;
```

```
import java.awt.*;
import javax.swing.Timer;

public class ShowMessageScreen extends MainScreen implements
ActionListener
{
    private JLabel background;
    private JButton button;
    private boolean endProgram = false;
    private int delayMS = 0;
    private String runNext = "";
    private boolean waitingGo = true;

    public ShowMessageScreen(String getFileName, boolean endChoice, int
waitingSecond) {
        super(getFileName);
        endProgram = endChoice;
        delayMS = waitingSecond * 1000;
        waitingThenNext(delayMS);
    }

    public ShowMessageScreen(String getFileName, boolean endChoice, int
waitingSecond, String runNext) {
        super(getFileName);
        endProgram = endChoice;
        delayMS = waitingSecond * 1000;
        this.runNext = runNext;
        waitingThenNext(delayMS);
    }
    private void waitingThenNext(int waitingMS) {
        Timer timer = new Timer(waitingMS, new ActionListener() {
                    public void actionPerformed(ActionEvent e) {
                        if(!endProgram) {
                            switch(runNext) {
                                case "empty":
                                    break;
                                case "logout":
                                    logout();
```

```
                                    break;
                                case "withdrawlComplete1":
                                    new
ShowMessageScreen("CardEjected.png",false,3, "withdrawlComplete2");
                                    break;
                                case "withdrawlComplete2":
                                    new
ShowMessageScreen("takeCash.png",false,3, "withdrawlComplete3");
                                    break;
                                case "withdrawlComplete3":
                                    new
ShowMessageScreen("CASHdispensed.png",false,3, "logout");
                                    break;
                                default:
                                    new MainMenuScreen("MainMenu.png");
                                    break;
                            }
                        }
                        waitingGo = false;
                        dispose();
                        ((Timer)e.getSource()).stop();
                    }
                });
        timer.start();
    }// end method waitingThenNext

    public void waiting() {
        while(waitingGo) {
            try
            {
                Thread.sleep(1);
            }
            catch (InterruptedException ie)
            {
                ie.printStackTrace();
            }
        }
    }//end method waiting
```

```
    public void screenCenter(JButton[] centerButton, JPanel panel){

    } //end method screenCenter

    public void actionPerformed( ActionEvent event )
    {
        switch (event.getActionCommand())
        {

        }
    } // end method actionPerformed
}// end class ShowMessageScreen
```

TRANSACTION.JAVA

```
// Transaction.java
// Abstract superclass Transaction represents an ATM transaction

public abstract class Transaction
{
   private int accountNumber; // indicates account involved
   private Screen screen; // ATM's screen
   private BankDatabase bankDatabase; // account info database

   // Transaction constructor invoked by subclasses using super()
   public Transaction( int userAccountNumber, Screen atmScreen,
      BankDatabase atmBankDatabase )
   {
      accountNumber = userAccountNumber;
      screen = atmScreen;
      bankDatabase = atmBankDatabase;
   } // end Transaction constructor

   // return account number
   public int getAccountNumber()
   {
```

```
      return accountNumber;
   } // end method getAccountNumber

   // return reference to screen
   public Screen getScreen()
   {
      return screen;
   } // end method getScreen

   // return reference to bank database
   public BankDatabase getBankDatabase()
   {
      return bankDatabase;
   } // end method getBankDatabase

   // perform the transaction (overridden by each subclass)
   abstract public void execute();
} // end class Transaction
```

TRANSFER.JAVA

```
// Represents a withdrawal ATM transaction

public class Transfer extends Transaction
{
    private double amount; // transfer amount
    private Keypad keypad; // reference to keypad
    private int transferAccount; //the account to be transferred

    // constant corresponding to menu option to cancel
    private final static int CANCELED = 3;

    // Withdrawal constructor
    public Transfer( int userAccountNumber, Screen atmScreen,
    BankDatabase atmBankDatabase, Keypad atmKeypad)
    {
        // initialize superclass variables
```

```
        super( userAccountNumber, atmScreen, atmBankDatabase );

        // initialize references to keypad and cash dispenser
        keypad = atmKeypad;
    } // end Withdrawal constructor

    // perform transaction
    public void execute()
    {
        boolean transferred = false; // cash was not dispensed yet
        double availableBalance; // amount available for withdrawal

        // get references to bank database and screen
        BankDatabase bankDatabase = getBankDatabase();
        Screen screen = getScreen();

        // loop until cash is dispensed or the user cancels
        do
        {
            if(displayMenu() == CANCELED) {

                screen.displayMessageLine( "\nCanceling transaction..." );
                return; // return to main menu because user canceled
            }
            if(getAccountNumber() == transferAccount) {
                screen.displayMessageLine( "\nTransfers to your own
account are not allowed." );
                return; // return to main menu because can't transfer
money to myself
            }

            if(bankDatabase.checkAccountExist(transferAccount)) {

                // get available balance of account involved
                availableBalance =
                bankDatabase.getAvailableBalance( getAccountNumber() );

                // check whether the user has enough money in the account
```

```
                if ( amount <= availableBalance && amount > 0)
                {
                    // update the account involved to reflect withdrawal
                    bankDatabase.debit( getAccountNumber(), amount );
                    bankDatabase.transferAdd( transferAccount, amount );

                    transferred = true; // Successfully transferred.

                    // instruct user to take cash
                    screen.displayMessageLine(
                        "\nSuccessfully transferred." );
                } // end if
                else // not enough money available in user's account
                {
                    screen.displayMessageLine(
                        "\nInsufficient funds in your account." +
                        "\n\nPlease enter a smaller amount or positive
amount." );
                } // end else
            } else {
                //transfer account not exist
                screen.displayMessageLine(
                    "\nTransfer account does not exist. Please try
again." );
            }

        } while ( !transferred );

    } // end method execute

    // display a menu of withdrawal amounts and the option to cancel;
    // return the chosen amount or 0 if the user chooses to cancel
    private int displayMenu()
    {
        int userChoice = 0; // local variable to store return value

        Screen screen = getScreen(); // get screen reference
```

```
        // loop while no valid choice has been made
        while ( userChoice == 0 )
        {
            // display the menu
            screen.displayMessageLine( "\nTransfer Menu:" );
            screen.displayMessage( "\nPlease enter the account number you
want to transfer to: " );

            int accountNumber = keypad.getInput(); // get user input
through keypad

            screen.displayMessage( "\nPlease enter the amount you want to
transfer: " );

            double inputAmount = keypad.getDoubleInput(); // get user
input through keypad

            screen.displayMessageLine("Please confirm the account number
and amount you entered.\n");
            screen.displayMessageLine("Account Number: " +
Integer.toString(accountNumber));
            screen.displayMessageLine("Amount transferred: " +
Double.toString(inputAmount));

            // display the menu
            screen.displayMessageLine( "\nTransfer Confirmation:" );
            screen.displayMessageLine( "1 - Confirm" );
            screen.displayMessageLine( "2 - Re-enter" );
            screen.displayMessageLine( "3 - Cancel transaction" );
            screen.displayMessage( "\nChoose an action: " );

            userChoice = keypad.getInput(); // get user input through
keypad
            switch(userChoice) {
                case 1:
                    //Confirm
                    transferAccount = accountNumber;
                    amount = inputAmount;
```

```
                    break;
                case 2:
                    userChoice = 0;
                    break;
                case CANCELED:
                    userChoice = CANCELED;
                    break;
                default: // the user did not enter a value from 1-6
                    screen.displayMessageLine(
                        "\nIvalid selection. Try again." );
            }
        } // end while

        return userChoice; // return withdrawal amount or CANCELED
    } // end method displayMenuOfAmounts
} // end class Withdrawal
```

TRANSFERFUNCTION.JAVA

```
public class TransferFunction extends MainScreen
{   private Confirmation confirmation;
    private String userID="";
    private String amount="" ;
    public TransferFunction() {

            TransferMoneyUserID transferMoneyUserID = new
TransferMoneyUserID("TransferMoneyUserID.png"); //show
TransferMoneyUserID.png
            transferMoneyUserID.setVisible(true);
            transferMoneyUserID.waiting(); // wait the user to input value

            TransferMoneyAmount transferMoneyAmount = new
TransferMoneyAmount("TransferMoneyAmount.png");//show
TransferMoneyAmount.png
            transferMoneyAmount.setVisible(true);
            transferMoneyAmount.waiting(); // wait the user to input value
```

```
            try{
                userID = transferMoneyUserID.getTransferMoneyUserID();
//get the transfer ID
                amount = transferMoneyAmount.getTransferMoneyAmount();
//get the money amount to tranfer
            }
            catch(Exception ex){
            }
            transferMoneyUserID = null;
            transferMoneyAmount = null;

            try{
                Confirmation confirmation = new
Confirmation("Confirmation.png",Integer.parseInt(userID),Double.parseDoubl
e(amount));// put the userID and admount value to a new class
                confirmation.setVisible(true);
            }
            catch(Exception ex){
     ShowMessageScreen showErrorMessage = new
     ShowMessageScreen("AccountOrBalanceError.png",false,3); //show error
message if
                                                    account or amount has
error
            showErrorMessage.setVisible(true);
            showErrorMessage.waiting();
}


    }
}
```

TRANSFERMONEYAMOUNT.JAVA

```
import java.awt.event.ActionListener;
import javax.swing.JTextField;
import javax.swing.JButton;
import javax.swing.JPanel;
```

```
import java.awt.event.ActionEvent;

public class TransferMoneyAmount extends MainScreen implements
ActionListener{
    private JTextField transferMoneyAmount;
    public TransferMoneyAmount(String getFileName) {
        super(getFileName);
    }

    public String getTransferMoneyAmount() {
        return transferMoneyAmount.getText();
    }

    public void screenCenter(JButton[] centerButton, JPanel panel){

        transferMoneyAmount = getTextField("",true,(screenWidth-309)/2,
300, 605/2, 32);
        panel.add (transferMoneyAmount);
    }//end of Center

    public void waiting() {
        while(waitingEnter) {
            try
            {
                Thread.sleep(1);
            }
            catch (InterruptedException ie)
            {
                ie.printStackTrace();
            }
        }// end while
    } //end method waiting

    public void actionPerformed( ActionEvent event )
    {
        switch (event.getActionCommand())
        {
            case "0" :
```

```
                transferMoneyAmount.setText(transferMoneyAmount.getText().concat( event.getActionCommand() ));
                 break;

            case "00" :
                transferMoneyAmount.setText(transferMoneyAmount.getText().concat( event.getActionCommand() ));
                 break;

            case "." :
                transferMoneyAmount.setText(transferMoneyAmount.getText().concat( event.getActionCommand() ));
                 break;

            case "7" :
                transferMoneyAmount.setText(transferMoneyAmount.getText().concat( event.getActionCommand() ));
                 break;

            case "8" :
                transferMoneyAmount.setText(transferMoneyAmount.getText().concat( event.getActionCommand() ));
                 break;

            case "9" :
                transferMoneyAmount.setText(transferMoneyAmount.getText().concat( event.getActionCommand() ));
                 break;

            case "6" :
```

```
                transferMoneyAmount.setText(transferMoneyAmount.getText().concat( event.getActionCommand() ));
                 break;

            case "5" :
                transferMoneyAmount.setText(transferMoneyAmount.getText().concat( event.getActionCommand() ));
                 break;

            case "4" :
                transferMoneyAmount.setText(transferMoneyAmount.getText().concat( event.getActionCommand() ));
                 break;

            case "3" :
                transferMoneyAmount.setText(transferMoneyAmount.getText().concat( event.getActionCommand() ));
                 break;

            case "2" :
                transferMoneyAmount.setText(transferMoneyAmount.getText().concat( event.getActionCommand() ));
                 break;

            case "1" :
                transferMoneyAmount.setText(transferMoneyAmount.getText().concat( event.getActionCommand() ));
                 break;

            case "Enter":

                 waitingEnter = false;
```

```
                dispose();

                break;
            case "Change" :
                transferMoneyAmount.setText("");
                break;

            case "Cancel" :
                new MainMenuScreen("MainMenu.png");
                dispose();
                break;
        } // end switch statement
    } // end method actionPerformed
}// end class TransferMoneyAmount
```

TRANSFERMONEYUSERID.JAVA

```
import javax.swing.JTextField;
import java.awt.event.ActionListener;
import javax.swing.JPanel;
import javax.swing.JButton;
import java.awt.event.ActionEvent;

public class TransferMoneyUserID extends MainScreen implements
ActionListener{

    private JTextField transferMoneyUserID;

    public TransferMoneyUserID(String getFileName) {
        super(getFileName);
    }

    public String getTransferMoneyUserID() {
        return transferMoneyUserID.getText();
    }

    public void screenCenter(JButton[] centerButton, JPanel panel){
```

```
        transferMoneyUserID = getTextField("",true,(screenWidth-309)/2,
300, 605/2, 32) ;
        panel.add (transferMoneyUserID);
    }//end of Center

    public void waiting() {
        while(waitingEnter) {
            try
            {
                Thread.sleep(1);
            }
            catch (InterruptedException ie)
            {
                ie.printStackTrace();
            }
        }
    }// end method waiting

    public void actionPerformed( ActionEvent event )
    {
        switch (event.getActionCommand())
        {
            case "0" :
                
transferMoneyUserID.setText(transferMoneyUserID.getText().concat( event.ge
tActionCommand() ));
                break;

            case "00" :
                
transferMoneyUserID.setText(transferMoneyUserID.getText().concat( event.ge
tActionCommand() ));
                break;

            case "." :

transferMoneyUserID.setText(transferMoneyUserID.getText().concat( event.ge
tActionCommand() ));
```

```
                break;

            case "7" :

transferMoneyUserID.setText(transferMoneyUserID.getText().concat( event.ge
tActionCommand() ));
                break;

            case "8" :

transferMoneyUserID.setText(transferMoneyUserID.getText().concat( event.ge
tActionCommand() ));
                break;

            case "9" :

transferMoneyUserID.setText(transferMoneyUserID.getText().concat( event.ge
tActionCommand() ));
                break;

            case "6" :

transferMoneyUserID.setText(transferMoneyUserID.getText().concat( event.ge
tActionCommand() ));
                break;

            case "5" :

transferMoneyUserID.setText(transferMoneyUserID.getText().concat( event.ge
tActionCommand() ));
                break;

            case "4" :

transferMoneyUserID.setText(transferMoneyUserID.getText().concat( event.ge
tActionCommand() ));
                break;
```

```
            case "3" :
                transferMoneyUserID.setText(transferMoneyUserID.getText().concat( event.getActionCommand() ));
                 break;

            case "2" :
                transferMoneyUserID.setText(transferMoneyUserID.getText().concat( event.getActionCommand() ));
                 break;

            case "1" :
                transferMoneyUserID.setText(transferMoneyUserID.getText().concat( event.getActionCommand() ));
                 break;

            case "Enter":

                    waitingEnter = false;
                    dispose();


                 break;
            case "Change" :
                 transferMoneyUserID.setText("");
                 break;

            case "Cancel" :
                 new MainMenuScreen("MainMenu.png");
                 dispose();
                 break;
         }// end switch statment
    } // end method actionPerformed
} // end class TransferMoneyUserID
```

VIEWBALANCESCREEN.JAVA

```
//ViewBalanceScreen.java
import javax.swing.JPanel;
import javax.swing.JLabel;
import javax.swing.JTextField;
import javax.swing.JPasswordField;
import javax.swing.JButton;
import java.awt.event.ActionListener;
import java.awt.event.ActionEvent;
import java.awt.BorderLayout;
import javax.swing.JOptionPane;
import java.awt.GridLayout;
import javax.swing.ImageIcon;
import javax.swing.Icon;
import javax.swing.JFrame;
import java.awt.*;

public class ViewBalanceScreen extends MainScreen {
    private JTextField viewAvailableBalance;
    private JTextField viewTotalBalance;
    private boolean waitingEnter = true;
    private int accountNumber;
    private double availableBalance;
    private double totalBalance;

    //ViewBalanceScreen constructor
    public ViewBalanceScreen(String getFileName) {
        super(getFileName);

        // get the available balance for the account involved
         availableBalance =
bankDatabase.getAvailableBalance( getAccount() );
         viewAvailableBalance.setText(Double.toString(availableBalance));

        // get the total balance for the account involved
         totalBalance = bankDatabase.getTotalBalance( getAccount() );
         viewTotalBalance.setText(Double.toString(totalBalance));
```

```
    } // end ViewBalanceScreen constructor


    public void screenCenter(JButton[] centerButton, JPanel panel){
        viewAvailableBalance = getTextField("",false,(screenWidth)/2+25,
167, 248, 32);
        panel.add (viewAvailableBalance);

        viewTotalBalance = getTextField("",false,(screenWidth)/2-30, 260,
605/2, 32);
        panel.add (viewTotalBalance);
        //end of Center
    }

    public void waiting() {
        while(waitingEnter) {
            try
            {
                Thread.sleep(1);
            }
            catch (InterruptedException ie)
            {
                ie.printStackTrace();
            }
        }
    }// end method waiting

    public void actionPerformed( ActionEvent event )
    {
        switch (event.getActionCommand())
        {
            case "Enter":
                new MainMenuScreen("MainMenu.png");
                dispose();
                break;
        }
    } // end method actionPerformed
}//end class ViewBalanceScreen
```

WITHDRAWAL.JAVA

```java
// Withdrawal.java
// Represents a withdrawal ATM transaction

public class Withdrawal extends Transaction
{
   private int amount; // amount to withdraw
   private Keypad keypad; // reference to keypad
   private CashDispenser cashDispenser; // reference to cash dispenser

   // constant corresponding to menu option to cancel
   private final static int CANCELED = 5;

   // Withdrawal constructor
   public Withdrawal( int userAccountNumber, Screen atmScreen,
      BankDatabase atmBankDatabase, Keypad atmKeypad,
      CashDispenser atmCashDispenser )
   {
      // initialize superclass variables
      super( userAccountNumber, atmScreen, atmBankDatabase );

      // initialize references to keypad and cash dispenser
      keypad = atmKeypad;
      cashDispenser = atmCashDispenser;
   } // end Withdrawal constructor

   // perform transaction
   public void execute()
   {
      boolean cashDispensed = false; // cash was not dispensed yet
      double availableBalance; // amount available for withdrawal

      // get references to bank database and screen
      BankDatabase bankDatabase = getBankDatabase();
      Screen screen = getScreen();
```

```
// loop until cash is dispensed or the user cancels
do
{
   // obtain a chosen withdrawal amount from the user
   amount = displayMenuOfAmounts();

   // check whether user chose a withdrawal amount or canceled
   if ( amount != CANCELED )
   {
      // get available balance of account involved
      availableBalance =
         bankDatabase.getAvailableBalance( getAccountNumber() );

      // check whether the user has enough money in the account
      if ( amount <= availableBalance && amount > 0)
      {
         // check whether the cash dispenser has enough money
         if ( cashDispenser.isSufficientCashAvailable( amount ) )
         {
            // update the account involved to reflect withdrawal
            bankDatabase.debit( getAccountNumber(), amount );

            cashDispenser.dispenseCash( amount ); // dispense cash
            cashDispensed = true; // cash was dispensed

            // instruct user to take cash
            screen.displayMessageLine(
               "\nPlease take your cash now." );
         } // end if
         else // cash dispenser does not have enough cash
            screen.displayMessageLine(
               "\nInsufficient cash available in the ATM." +
               "\n\nPlease choose a smaller amount." );
      } // end if
      else // not enough money available in user's account
      {
         screen.displayMessageLine(
```

```
                  "\nInsufficient funds in your account." +
                  "\n\nPlease enter a smaller amount or positive
amount." );
            } // end else
         } // end if
         else // user chose cancel menu option
         {
            screen.displayMessageLine( "\nCanceling transaction..." );
            return; // return to main menu because user canceled
         } // end else
      } while ( !cashDispensed );

   } // end method execute

   // display a menu of withdrawal amounts and the option to cancel;
   // return the chosen amount or 0 if the user chooses to cancel
   private int displayMenuOfAmounts()
   {
      int userChoice = 0; // local variable to store return value

      Screen screen = getScreen(); // get screen reference

      // array of amounts to correspond to menu numbers
      int amounts[] = { 0, 100, 500, 1000 };

      // loop while no valid choice has been made
      while ( userChoice == 0 )
      {
         // display the menu
         screen.displayMessageLine( "\nWithdrawal Menu:" );
         screen.displayMessageLine( "1 - $100" );
         screen.displayMessageLine( "2 - $500" );
         screen.displayMessageLine( "3 - $1000" );
         screen.displayMessageLine( "4 - Custom" );
         screen.displayMessageLine( "5 - Cancel transaction" );
         screen.displayMessage( "\nChoose a withdrawal amount: " );

         int input = keypad.getInput(); // get user input through keypad
```

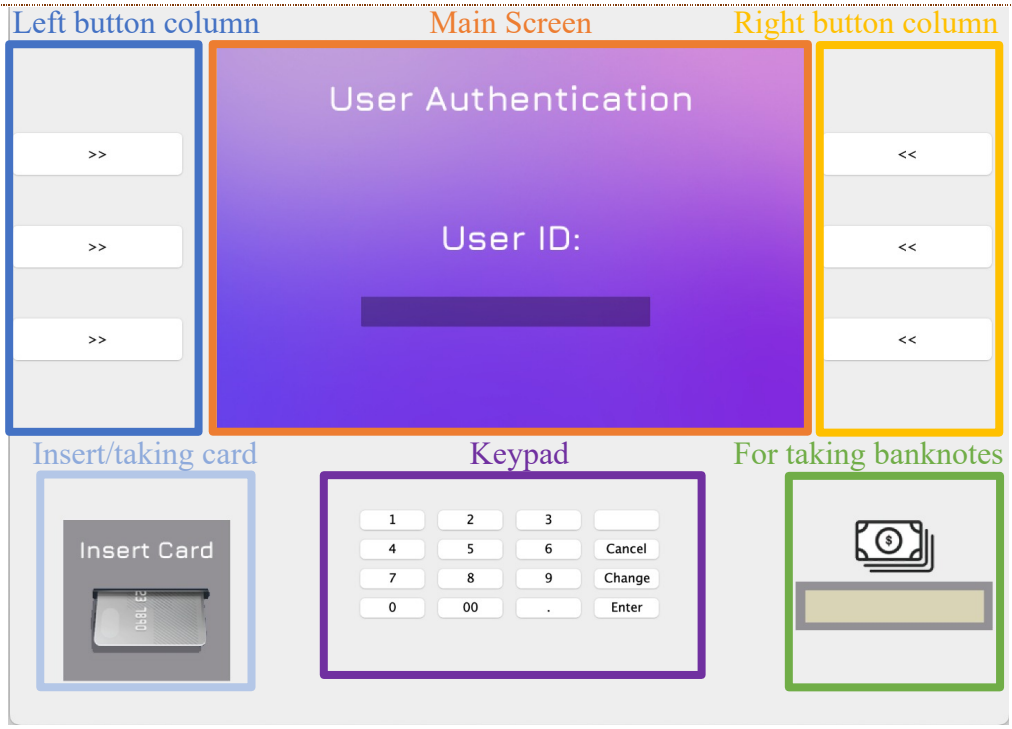
```
         // determine how to proceed based on the input value
         switch ( input )
         {
            case 1: // if the user chose a withdrawal amount
                userChoice = amounts[ input ];
                break;
            case 2: // (i.e., chose option 1, 2, 3, 4 or 5), return the
                userChoice = amounts[ input ];
                break;
            case 3: // corresponding amount from amounts array
                userChoice = amounts[ input ];
                break;
           case 4:
                screen.displayMessageLine( "\nPlease enter the amount you
want to withdraw:" );
                int customAmount = keypad.getInput();
                if(customAmount % 100 == 0) {
                    userChoice = customAmount;
                } else {
                     screen.displayMessageLine( "\nOnly the multiples of
HKD100, HKD500, or HKD1000 are allowed. Try again." );
                }
                break;
            case CANCELED: // the user chose to cancel
               userChoice = CANCELED; // save user's choice
               break;
            default: // the user did not enter a value from 1-6
               screen.displayMessageLine(
                  "\nIvalid selection. Try again." );
         } // end switch
      } // end while

      return userChoice; // return withdrawal amount or CANCELED
   } // end method displayMenuOfAmounts
} // end class Withdrawal
```

WITHDRAWALMENUSCREEN.JAVA

```
import java.awt.event.ActionListener;
import javax.swing.JButton;
import javax.swing.JPanel;
import java.awt.event.ActionEvent;

public class WithdrawalMenuScreen extends MainScreen implements
ActionListener{

    //WithdrawalMenuScreen constructor initializes the Scanner
    public WithdrawalMenuScreen(String getFileName) {
        super(getFileName);
    }//end WithdrawalMenuScreen constructor

    public void screenCenter(JButton[] centerButton, JPanel panel){

    }//end of Center

    public void actionPerformed( ActionEvent event )
    {
        switch (event.getActionCommand())
        {
            case "L1" :
                new FixedAmountWithdrawalScreen("Withdrawal100.png",100);
                dispose();
                break;
            case "L2" :
                new FixedAmountWithdrawalScreen("Withdrawal500.png",500);
                dispose();
                break;
            case "L3" :
                new
FixedAmountWithdrawalScreen("Withdrawal1000.png",1000);
                dispose();
                break;
            case "R1" :
                new CustomWithdrawalScreen("customWithdrawal.png");
```

```
                dispose();
                break;
            case "R3" :
                new MainMenuScreen("MainMenu.png");
                dispose();
                break;
            case "Cancel":
                new MainMenuScreen("MainMenu.png");
                dispose();
                break;
        }
    } // end method actionPerformed
}// end class WithdrawalMenuScreen
```

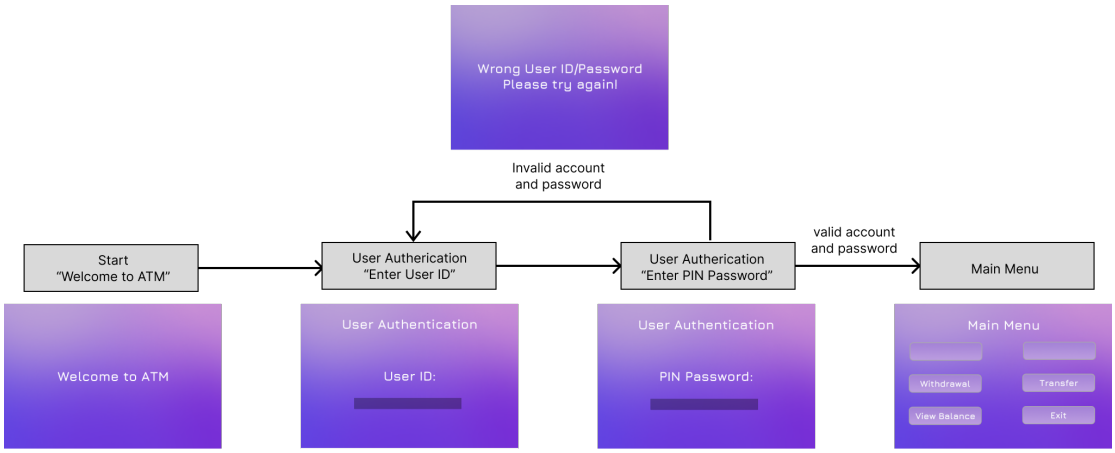
ATM USER INTERFACE



ASSUMPTION

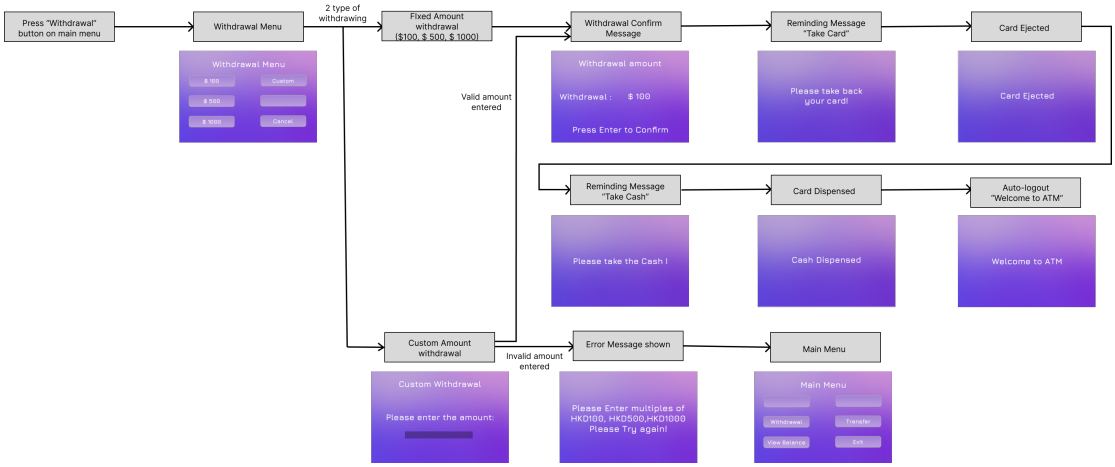
1. The ATM only accept integer and double input.
2. The input is mainly inputted by the physical keypad and the button on both side of ATM

User Authentication:

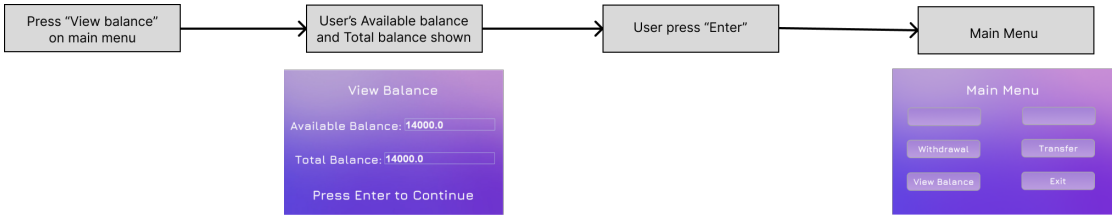


After main menu, user could use different functions of ATM, which is withdrawal, view balance, transfer, and exit.

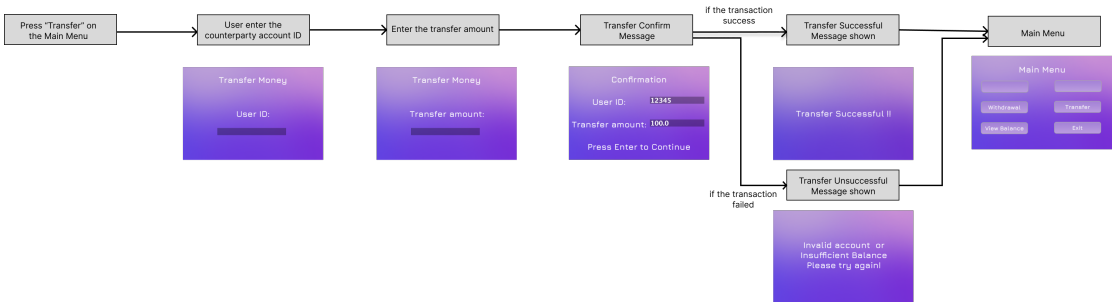
Withdrawal function:



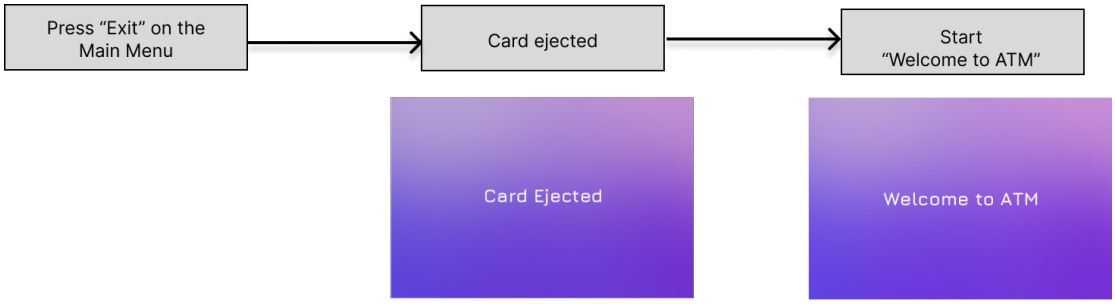
View balance function:



Transfer function:



Exit function:

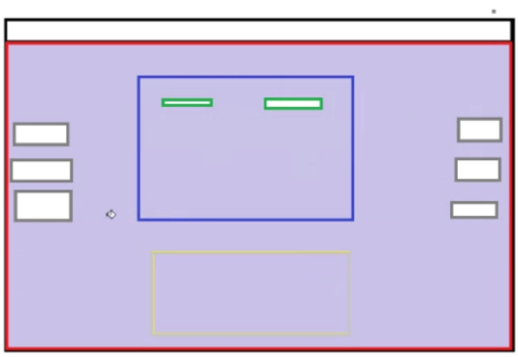


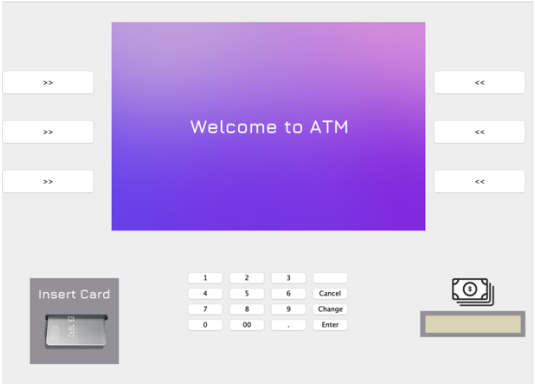
To build up the ATM function, the GUI program will ask the user to input the value and pass the inputted value to the main program. For instance: View balance, Withdraw, Transaction. However, the **System.out.print()** function is removed due to the use of GUI program. The concept of GUI program will use the Interger and Boolean value to display the screen of the ATM with ATM.java and it will pass the value to other java for choosing the proper GUI panel as well as display it.

There are some functions that used frequently:
Gettext.concat( ): It is used to get the input through the whole program of GUI.
Setbound( ): It is used to set the position and size of components. It mainly uses to set the display screen.
JTextField: It is used to create a new text field in the GUI which allows the user to edit a single line of text.
JPasswordField: It is a text component specialized for password entry which allows the editing of a single line of text with a default document, null starting text string, and 0 column width.
ActionListener: It is notified whenever you click on the button or menu item. It is notified against ActionEvent. The ActionListener interface is found in java.awt.event package.
ActionEvent: The ActionEvent class mainly deals with the event of the button being pressed. After implementing the ActionListener interface, the parameter e of actionPerformed() is the ActionEvent object.

ATM PANEL

To introduce the GUI for the ATM, a new class Mainscreen.java is created. In the ATM, there are some components such as the buttons, panels, text field. To explain more, there are some details of the program:





- The **red** outer frame is the display screen of the ATM machine and space for all our ATM buttons.
- The **blue** outer frame is the main display of our ATM. It shows what happens when the user uses the ATM.
- The **green** outer frames are the space to show the user what they are being asked to enter and enter user information.
- The **grey** outer frames are buttons for the user to select functions, such as View My Balance, Withdrawals and Transactions.
- The **yellow** frame is the string keypad for the user to input numbers **“0-9”**, **“00”**, **"."**, **"Cancel"**, **"Change"** and **"Enter"**.

```
public MainScreen(String backgroundFileName) {
    setResizable(false);
    setSize (screenWidth, screenHeight);
    setVisible(true);
    JPanel panel = new JPanel ();

    panel.setLayout(new BorderLayout());
    getLeftButton(leftButton,panel);
    getRightButton(rightButton,panel);
    getBottomOfCenterButton(bottomOfCenterButton,panel);
    screenCenter(centerButton,panel);
    setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

    JLabel background=new JLabel(new ImageIcon(backgroundFileName));
    panel.add(background);
    add(panel);
}
```

To set up the main frame of ATM, the frame is used the border layout to spill out the frame into the left, right, center, bottom.

```
public void getLeftButton(JButton[] leftButton, JPanel panel){
    for ( int count = 0; count < 3; count++ )
    {
        leftButton[count] = new JButton( ">>" );
        leftButton[count].setActionCommand("L"+Integer.toString(count+1));
        leftButton[count].addActionListener(this);
        leftButton[count].setBounds (0, 95*(count+1)+35, 180, 50);
        panel.add(leftButton[count]);
    } // end for
}
```

```
public void getRightButton(JButton[] rightButton, JPanel panel){
    for ( int count = 0; count < 3; count++ )
    {
        rightButton[count] = new JButton( "<<" );
        rightButton[count].setActionCommand("R"+Integer.toString(count+1));
        rightButton[count].setBounds(screenWidth-180-15, 95*(count+1)+35, 18
        rightButton[count].addActionListener(this);
        panel.add(rightButton[count]);
    } // end for
}
```

For left and right side of ATM, it has declared some buttons on the ATM panel. There are three buttons on respective left and right sides. A for loop statement is used to display the buttons. An actionlister is used to give the nociception to the user when the user presses the button.

```
public void getBottomOfCenterButton(JButton[] bottomOfCenterButton, JPanel panel){
    String keypad[]={"0","00",".","Enter","7","8","9","Change","4","5","6","Cancel","1","2","3",""};
    int buttonCount = 0;
    for ( int runTimes = 0; runTimes < 4; runTimes++) {
        for ( int count = 0; count < 4; count++ )
        {
            bottomOfCenterButton[buttonCount] = new JButton( keypad[buttonCount] );
            bottomOfCenterButton[buttonCount].setBounds (screenWidth/2-160+80*(count), screenHeight-30*(runTimes+2)-100, 80, 25);
            bottomOfCenterButton[buttonCount].addActionListener(this);
            panel.add(bottomOfCenterButton[buttonCount]);
            buttonCount++;
        } // end for
    }//end for
}// end method getBottomOfCenterButton
```

For the bottom part of ATM, it has declared the keypad on the ATM panel. There are 15 buttons in the string keypad, those are "0", "1", "2", "3", "4", "5", "6", "7", "8", "9", "0", "00", ".", "Cancel", "Change", "Enter". Those buttons are provided to the users to input their ID, password, withdraw amount, transfer amount, reset the input, and enter to continue.

For the center position, a screen is displayed to represent the reality screen of ATM. It is used to show the user's status while the users are using ATM.

```
// start ATM
public void run()
{
    // welcome and authenticate user; perform transactions
    while ( true )
    {
        // loop while user is not yet authenticated
        while ( !userAuthenticated )
        {
            ShowMessageScreen welcomeMessage = new ShowMessageScreen("welcomeATM.png",false,3,"empty");
            welcomeMessage.setVisible(true);
            welcomeMessage.waiting();
            authenticateUser(); // authenticate user
        } // end while

        performTransactions(); // user is now authenticated
        userAuthenticated = false; // reset before next ATM session
    } // end while
} // end method run
```

To create an interface of the ATM, an ATM.java is created. To adjust the size of screen, setsize is used for setting the value of ATM, it is set as 1024:768 which is a rectangular area of 1024 pixels wide and 768 pixels high. It is in 4:3 ratio. The default size of a frame is 0 x 0 pixels.

However, setresizable is used to prevent the user unhinged the screen. If the user undraw the screen (the value is true), all component in the frame will be separated and

the layout of the ATM will be different. It may provide a bad experience for the user who uses the ATM. Therefore, setresizeable is used to restrict the size of the ATM and provide a better experience to the user.

Besides, setvisible (true) is used to hide the ID password entered by the user. This feature is used because all account passwords and IDs are personal information and should not be easily shared and shown to others. To provide a secure networked ATM system, a commitment must be made to users who do not share personal information. Therefore, the password will be hidden when the user enters the password in the ATM.

```
// attempts to authenticate user against database
private void authenticateUser()
{
    accountLoginScreen loginAccount = new accountLoginScreen("UserID.png");
    loginAccount.setVisible(true);
    loginAccount.waiting();

    int accountNumber = loginAccount.getAccount();
    loginAccount = null;

    passwordLoginScreen loginPassword = new passwordLoginScreen("Password.png");
    loginPassword.setVisible(true);
    loginPassword.waiting();
    int pin = loginPassword.getPassword();
    loginPassword = null;

    // set userAuthenticated to boolean value returned by database
    userAuthenticated =
    bankDatabase.authenticateUser( accountNumber, pin );

    // check whether authentication succeeded
    if ( userAuthenticated )
    {
        getMainScreen.setAccount(accountNumber); // save user's account #
    } // end if
    else {
        ShowMessageScreen showMessage = new ShowMessageScreen("WrongLogin.png",false, 3, "empty");
        showMessage.setVisible(true);
        showMessage.waiting();
    }
} // end method authenticateUser
```

bankDatabase.authenticateUser(accountNumber, pin) is a function used to ask the user to enter their account number and password. Then, it could check that the account number and password are both equal to the database data. If any of these are not equal user Authenticated will be false, the Show Message Screen will display a message "Wronglogin".

ATM

To run the program, a while loop statement in **public void run()** is used to run the program of ATM. When the start of program, a welcome screen will be showed on the screen with **setvisible()**. It is used to show the screen of the welcome message of ATM. After waiting 3 seconds, a user authentication screen is shown.

For login screen,

It is separated into two screens: the account login screen and password login screen. A text label and text field are created to allow the user to input their userID and password respectively. If the user inputted the invalid value of userID and Password, it would request the user to input the value again. However, the account and password logic screen have similar design so it will explain the statement together.

```
public void screenCenter(JButton[] centerButton, JPanel panel){
    userText = getTextField("",true,(screenWidth-309)/2, 300, 605/2, 32) ;
    panel.add (userText);
}//end of Center
```

```
public void screenCenter(JButton[] centerButton, JPanel panel){
    passwordText = getPasswordTextField("",true,(screenWidth-309)/2, 300, 605/2, 32);
    panel.add (passwordText);
}//end of Center
```

Firstly, setbound is used for setting the value of position and size of the userlabel as well as the userText. Then, added them to the panel and show on the screen. The actionlistener is used for giving response whenever the user inputting the value by clicking the button or keypad of the main frame.

```
public void actionPerformed( ActionEvent event )
{
    switch (event.getActionCommand())
    {
        case "0" :
            passwordText.setText(passwordText.getText().concat( event.getActionCommand() ));
            break;
```

After the user inputted with the keypad, **public void actionPerformed (ActionEvent event)** is used for getting the inputted value. A **switch** statement is used to identify the value of the user inputted. For instance, the user input the “0”, **userText.setText(userText.getText() . concat (event.getActionCommand()));** is used to store value in the array of the getBottomOfCenterButton and allow to have the connection with screen by using **userText.getText() . concat (event.getActionCommand()).** The inputted value will be retrieved from **event.getActionCommand()** and connect to the text field of screen by showing the inputted value.

Here is the array of keypad:

```
public void getBottomOfCenterButton(JButton[] bottomOfCenterButton, JPanel panel){
    String keypad[]={"0","00",".","Enter","7","8","9","Change","4","5","6","Cancel","1","2","3",""};
```

For public void waiting (), the program will wait the user to press “enter” button.

```
public void waiting() {
    while(waitingEnter) {
        try
        {
            Thread.sleep(1);
        }
        catch (InterruptedException ie)
        {
            ie.printStackTrace();
        }
    }
}
```

If the user press “enter” button on the keypad, the while loop is break and public void waiting () is inactive. A main screen is displayed after pressing “enter” button.

```
case "Enter":
    waitingEnter = false;
    dispose();
    break;
```

```
case "Enter":

    try {
        setPassword(Integer.parseInt(passwordText.getText()));
        waitingEnter = false;
        dispose();
    } catch(NumberFormatException e){

    }
```

For another button,

```
case "Change" :
    passwordText.setText("");
    break;

case "Cancel" :
    passwordText.setText("");
    break;
```

When the user pressed the change button, all value is discarded, the value is changed to null.

When the user pressed the cancel button, the operation is cancelled and the ATM will release the card to the user.

For the Screen of ShowMessage:

When user complete the withdrawal transactions, the program will direct to ShowMessageScreen.java then show a message depend on the needs of the program on the screen.

```
case "withdrawlComplete1":
    new ShowMessageScreen("CardEjected.png",false,3, "withdrawlComplete2");
    break;
case "withdrawlComplete2":
    new ShowMessageScreen("takeCash.png",false,3, "withdrawlComplete3");
    break;
case "withdrawlComplete3":
    new ShowMessageScreen("CASHdispensed.png",false,3, "logout");
    break;
```

After the user had choose the withdrawal amount and confirm, the program will pay out money depend on the withdrawal amount user choose, then show the message "Cash Ejected!" to remind user to take the cash. The screen will hold for three seconds, the card will then pop up and show the message "Cash Ejected!" and hold the screen for 3 seconds.

```
public ShowMessageScreen(String getFileName, boolean endChoice, int waitingSecond) {
    super(getFileName);
    endProgram = endChoice;
    delayMS = waitingSecond * 1000;
    waitingThenNext(delayMS);
}

public ShowMessageScreen(String getFileName, boolean endChoice, int waitingSecond, String runNext) {
    super(getFileName);
    endProgram = endChoice;
    delayMS = waitingSecond * 1000;
    this.runNext = runNext;
    waitingThenNext(delayMS);
}
```

To show the correct message in different situation, ShowMessageScreen method was create by overloading. If only the message, decision of ending program and the waiting seconds were provided, the program will only show the file message and the waiting second that provided in getFileName and waitingSecond. After that, the program will redirect to the last java and continue.

```
private void waitingThenNext(int waitingMS) {
    Timer timer = new Timer(waitingMS, new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                if(!endProgram) {
                    switch(runNext) {
                        case "empty":
                            break;
                        case "logout":
                            logout();
                            break;
                        case "withdrawlComplete1":
                            new ShowMessageScreen("CardEjected.png",false,3, "withdrawlComplete2");
                            break;
                        case "withdrawlComplete2":
                            new ShowMessageScreen("takeCash.png",false,3, "withdrawlComplete3");
                            break;
                        case "withdrawlComplete3":
                            new ShowMessageScreen("CASHdispensed.png",false,3, "logout");
                            break;
                        default:
                            new MainMenuScreen("MainMenu.png");
                            break;
                    }
                }
                waitingGo = false;
                dispose();
                ((Timer)e.getSource()).stop();
            }
        });
    timer.start();
}//end method waitingThenNext
```

Also, if the program needs to direct to the other screen after showing the message for a specific second. A string variable runNext should be provided when the

method was called. Excepting the program will be ended, the program will direct to main menu after the message shown if the runNext status is logout. If the runNext is equal to empty, the switch statement will break out by no directing to any classes or screen and continue the program

```
private JLabel background;
private JButton button;
private boolean endProgram = false;
private int delayMS = 0;
private String runNext = "";
private boolean waitingGo = true;
```

To have better user experience, it has designed the waiting second in a unit of second which is microsecond by default. To change the unit of time by microsecond to second, the value of input will be a product of a thousand. The default of waiting second is 0 and the status of program terminate is false.

```
public ShowMessageScreen(String getFileName, boolean endChoice, int waitingSecond) {
    super(getFileName);
    endProgram = endChoice;
    delayMS = waitingSecond * 1000;
    waitingThenNext(delayMS);
}

private void waitingThenNext(int waitingMS) {
    Timer timer = new Timer(waitingMS, new ActionListener() {
                public void actionPerformed(ActionEvent e) {
                    if(!endProgram) {
                        new MainMenuScreen("MainMenu.png");
                    }
                    dispose();
                    ((Timer)e.getSource()).stop();
                }
            });
    timer.start();
}
```

MAIN MENU

Mainmenuscreen.java:

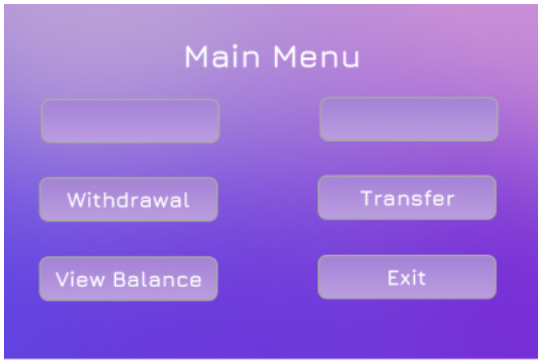
After the user authentication, the main menu is displayed on the ATM. The main menu screen was showing the functions of an ATM which are withdrawing money, viewing the user account balance, transferring money to others, and logout functions. Users can choose the role they want by pressing the button on the side:

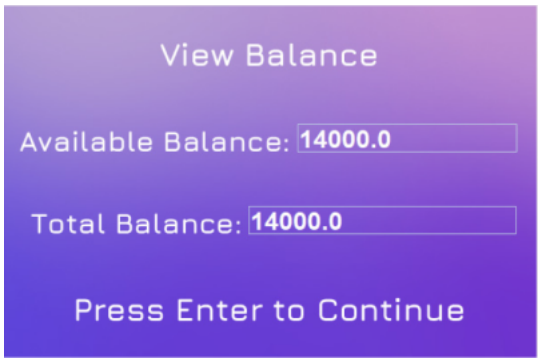
If the user moves the mouse and presses the button on the left-central, the program will direct to the WithdrawalMenuScreen.java and start the withdrawal procedure. If the user presses the bottom-right button, then the program will direct to ViewBalanceScreen.java and show the user's balances.

For the central-right button, the program will refer to the TransferFunction.java and start transferring money to another account.
For the logout function on the bottom-right, the program will eject the bank card and show the message to remind the user to take the card. Then, the program will initialize the account information for logout.

VIEW MY BALANCE

Viewbalance.java:





When the user presses the button on the left side of the ATM, the program will direct to the viewbalancescreen.java to show the View Balance Screen. In the View Balance Screen, there are two text field for showing the available balance and total balance of the client.

To retrieve the value of the balance on the screen, **private BankDatabase bankDatabase = new BankDatabase();**, an object is created to have a connection on the bankDatabase in the previous code:

```
public BankDatabase()
{
    accounts = new Account[ 5 ]; //  accounts for testing
    accounts[ 0 ] = new Account( 12345, 54321, 1000.0, 1200.0 );
    accounts[ 1 ] = new Account( 98765, 56789, 200.0, 200.0 );
    accounts[ 2 ] = new SavingAccount(55555, 55555, 2000.0, 1000.0);
    accounts[ 3 ] = new ChequeAccount(66666, 66666, 4000.0, 2000.0);
    accounts[ 4 ] = new Account(1, 1, 14000.0, 14000.0);
} // end no-argument BankDatabase constructor
```

To show the value to the screen of ATM, availableBalance = bankDatabase. getAvailableBalance (getAccountNumber()); is used to get available balance of the client from the bankDatabase by checking the account number of the client. The account number of the client is retrieved when the user press “view balance” button:

```
// retrieve Account object containing specified account number
private Account getAccount( int accountNumber )
{
    // loop through accounts searching for matching account number
    for ( Account currentAccount : accounts )
    {
        // return current account if match found
        if ( currentAccount.getAccountNumber() == accountNumber )
            return currentAccount;
    } // end for

    return null; // if no matching account was found, return null
} // end method getAccount
```

When the value is retrieved from the bankdatabase, the data type of value is in double which cannot be shown on the screen directly. Double.toString () is using to change String type to Double.Moreover, viewAvailableBalance.setText (Double.toString (availableBalance)); is used to setText on the JTextfield for showing the value on the Screen. String value is required for the input of jtextfield on the GUI.

```
public ViewBalanceScreen(String getFileName) {
    super(getFileName);

    // get the available balance for the account involved
    availableBalance = bankDatabase.getAvailableBalance( getAccountNumber() );
    viewAvailableBalance.setText(Double.toString(availableBalance));

    // get the total balance for the account involved
    totalBalance = bankDatabase.getTotalBalance( getAccountNumber() );
    viewTotalBalance.setText(Double.toString(totalBalance));

}
```

After viewing the balance, the user could press the “enter” button on the keypad, the ActionListener is activated and return to the main menu for further transaction or withdrawal the cash. There is the code of the enter button:

```
public void actionPerformed( ActionEvent event )
{
    switch (event.getActionCommand())
    {
        case "Enter":
            new MainMenuScreen("MainMenu.png");
            dispose();
            break;
    }
} // end method actionPerformed
```

WITHDRAW

Withdrawalscreen.java

```
public void actionPerformed( ActionEvent event )
{
    switch (event.getActionCommand())
    {
        case "L1" :
            new FixedAmountWithdrawalScreen("Withdrawal100.png",100);
            dispose();
            break;
        case "L2" :
            new FixedAmountWithdrawalScreen("Withdrawal500.png",500);
            dispose();
            break;
        case "L3" :
            new FixedAmountWithdrawalScreen("Withdrawal1000.png",1000);
            dispose();
            break;
        case "R1" :
            new CustomWithdrawalScreen("customWithdrawal.png");
            dispose();
            break;
        case "R3" :
            new MainMenuScreen("MainMenu.png");
            dispose();
            break;
        case "Cancel":
            new MainMenuScreen("MainMenu.png");
            dispose();
            break;
    }
} // end method actionPerformed
```

After the user press the “withdrawal” button in the main menu, a withdraw menu will be displayed on the screen. When the user presses the button of withdrawal menu, it will direct to different pages with the destination button. A switch statement is used to separate the function of withdrawal.

```
public FixedAmountWithdrawalScreen(String getFileName, int getAmount) {
    super(getFileName);
    withdrawlAmount = getAmount;
}
```

When the user presses the button of “$100”, “$500”, “$1000”, **withdrawlAmount = getAmount;** will get the value from the button that the user pressed and direct to the fixed amount screen.

Fixed AmountWithdrawalScreen.java

```
    public void actionPerformed( ActionEvent event )
    {

    switch (event.getActionCommand())
    {
        case "Enter":
            //call withdrawl function here
            execute();
            break;
        case "Change" :
            new WithdrawalMenuScreen("WithdrawalMenu.png");
            dispose();
            break;

        case "Cancel" :
            new MainMenuScreen("MainMenu.png");
            dispose();
            break;
    }
} // end method actionPerformed
```

For the fixed amount screen, it shows the value which is selected by the user. For instance, the user selected the $100 button, the program will display the value of "$100" on the screen.

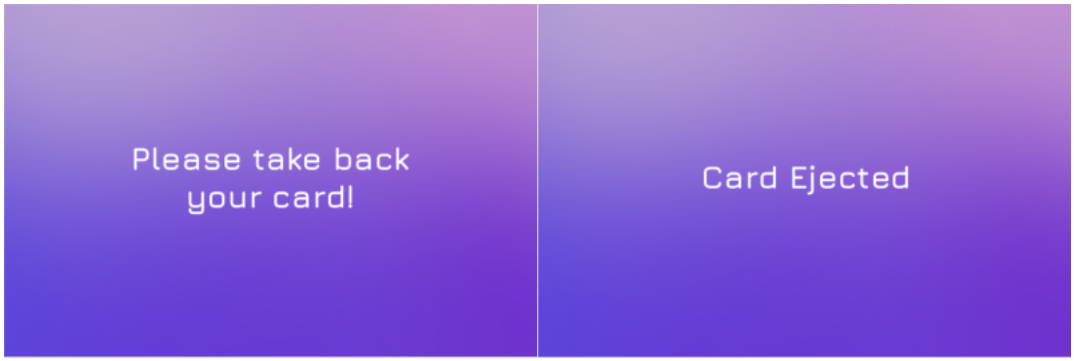
After the user press "enter" button, a execute () function in fixedamountscreen.java is activated. The program will store the available balance in the bankdatabase.java and get the value of the available balance with the code of bankDatabase:

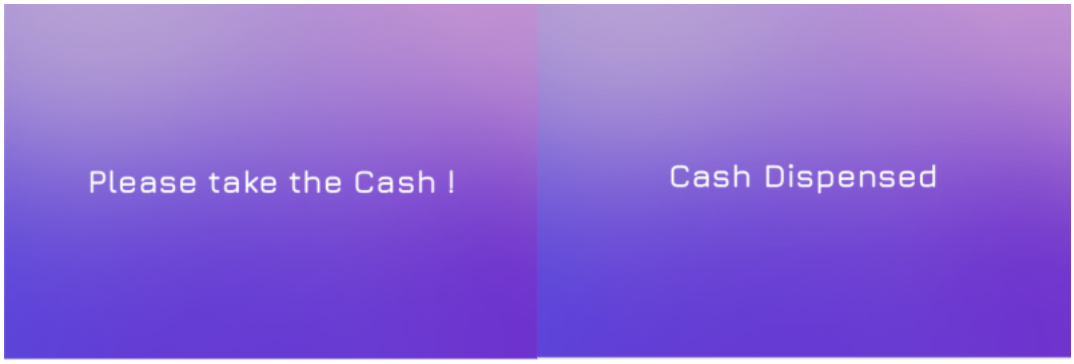
```
// return available balance of Account with specified account number
public double getAvailableBalance( int userAccountNumber )
{
    return getAccount( userAccountNumber ).getAvailableBalance();
} // end method getAvailableBalance
```

After getting the value of the available balance of client with the code of **withdrawlAmount = getAmount;**, some if statement is identified whenever the client would retrieved the cash from the ATM:

```
// check whether the user has enough money in the account
if ( withdrawlAmount <= availableBalance && withdrawlAmount > 0)
{
    // check whether the cash dispenser has enough money
    if ( cashDispenser.isSufficientCashAvailable( withdrawlAmount ) )
    {
        // update the account involved to reflect withdrawal
        bankDatabase.debit( getAccount(), withdrawlAmount );

        cashDispenser.dispenseCash( withdrawlAmount ); // dispense cash


        new ShowMessageScreen("takeCard.png",false,3, "withdrawlComplete1"); //Please take your cash now.
        dispose();
    } // end if
    else {// cash dispenser does not have enough cash
        new ShowMessageScreen("ATMInsufficientCash.png",false,3); //Insufficient cash available in the ATM.
        dispose();
    }
    } // end if
    else // not enough money available in user's account
    {
        new ShowMessageScreen("AccountOrBalanceError.png",false,3); //Insufficient funds in your account.
        dispose();
    } // end else
```

If the withdraw amount is smaller than the available balance of the client, it means that there is enough amount for the user to withdrawal, some display message will be displayed and ask the client to take card and cash which is showed on the screen.





For some situation:

```
else {// cash dispenser does not have enough cash
    new ShowMessageScreen("ATMInsufficientCash.png",false,3); //Insufficient cash available in the ATM.
    dispose();
}
```

Moreover, the withdraw amount is larger than the available balance of the client, an error message is displayed on the screen to indicate there are insufficient amount in the available balance for withdraw the cash.

```
else // not enough money available in user's account
{
    new ShowMessageScreen("AccountOrBalanceError.png",false,3); //Insufficient funds in your account.
    dispose();
} // end else
```

However, there may not be enough money in the ATM for the user to withdraw the money. In those situations, the program will display a message “” on the screen that indicates there is insufficient cash for the user to withdraw. “Direct to what? “ There is similar button in the program:

For example, the case of “change” in the keypad is used to direct to the withdrawal menu for rechoose the withdraw amount. It will direct to the menu of withdraw and allow the user to input another value.

Besides, it will redirect to the main menu if the user press “cancel” button. It means that the withdrawal action is ended and allows the user to have another action (e.g. transfer money, transaction) on the screen of main menu.

Customwithdrawal.java

```
if(withdrawlAmount % 100 == 0) {
    // check whether the user has enough money in the account
    if ( withdrawlAmount <= availableBalance && withdrawlAmount > 0)
    {
        // check whether the cash dispenser has enough money
        if ( cashDispenser.isSufficientCashAvailable( withdrawlAmount ) )
        {
            new confirmWithdrawalScreen("ConfirmWithdrawal.png",withdrawlAmount); //Insufficient cash available in the ATM.
            dispose();
        } // end if
        else {// cash dispenser does not have enough cash
            new ShowMessageScreen("ATMInsufficientCash.png",false,3); //Insufficient cash available in the ATM.
            dispose();
        }
    } // end if
    else // not enough money available in user's account
    {
        new ShowMessageScreen("AccountOrBalanceError.png",false,3); //Insufficient funds in your account.
        dispose();
    } // end else
} else {
    new ShowMessageScreen("ErrorWithdrawal.png",false,3); //Only the multiples of HKD100, HKD500, or HKD1000 are allowed. Try again.
    dispose();
}
```

When the user presses the custom button, the custom withdrawal is displayed. User were only can input the amount which can divide by 100 because the minimum banknotes of the ATM are 100 and user can only withdrawal the amount with the product of 100. Otherwise, the screen will show the message with “Only the multiples of HKD100, HKD500, or HKD1000 are allowed. Try again.” Then redirect to input the withdrawal amount. Also, the withdrawing amount cannot be larger than the balance of the user. Therefore, the program will compare the user input and the balance in bank database debt balance. If the balance is smaller than the withdraw amount, the message “Insufficient funds in your account. Please enter a smaller amount or positive amount.” will show on the screen. If the withdrawal amount is valid, the withdrawing procedure will start.

```
public void waiting() {
    while(waitingEnter) {
        try
        {
            Thread.sleep(1);
        }
        catch (InterruptedException ie)
        {
            ie.printStackTrace();
        }
    }
}
```

```
case "Enter":
    //call withdrawl function here
    execute();
    waitingEnter = false;
    dispose();
    break;
```

When the user enters the withdrawal amount and presses the enter button, the waiting method will be called for transferring the user input from getTextField into the amountText. Thread.sleep(1) is for the program to trigger a trap. If it is an error, the program will implement the catch statement ie.printStackTrace() which is to print out the position and the reason of the error.

```
public void actionPerformed( ActionEvent event )
{
    switch (event.getActionCommand())
    {
        case "0" :
            amountText.setText(amountText.getText().concat( event.getActionCommand() ));
            break;
```

When the user enters the withdrawal amount by pressing the button from the keypad, the actionPerformed method will be called. Every time user enters the number by pressing on the keypad, the actionPerform method will transform the ActionCommand to the amountText according to the button the user press on the keypad.

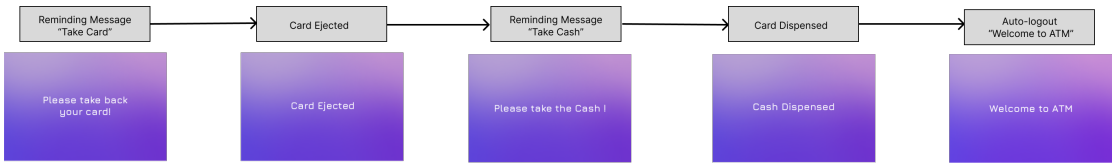
After the user selects the transfer function, the use can enter the ID to be transferred. Once enter the ID, the program will double check that the ID exists through the database. If it is invalid, the user can continue their transfer process and the display will show a confirmation message; if not, the transfer function will exist and return to the main menu.

After inputting the userID, the user is required to enter the transfer amount. The amount is defined by the user. Before determining the transfer user ID and transfer amount, the user needs to be double confirmed and press the “enter” button to confirm the value is correct. When the user presses the button, the program is directed to the transferfunction.java. It allows the authentication of the account number of another user and checks whether the available balance of the user is sufficient to transfer to another user. If the account number of another user is invalid or there is insufficient amount to have transaction. A message will be displayed on the screen.

```
try{
    Confirmation confirmation = new Confirmation("Confirmation.png",Integer.parseInt(userID),Double.parseDouble(amount));
    confirmation.setVisible(true);
    loop=false;
}
catch(Exception ex){}
```

When the transfer is complete, the user is shown a message that the transfer process is complete.

```
try{
    userID = transferMoneyUserID.getTransferMoneyUserID();
    amount = transferMoneyAmount.getTransferMoneyAmount();
}
catch(Exception ex){
}
transferMoneyUserID = null;
transferMoneyAmount = null;
```

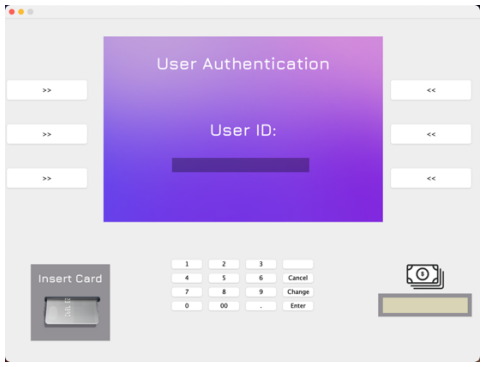
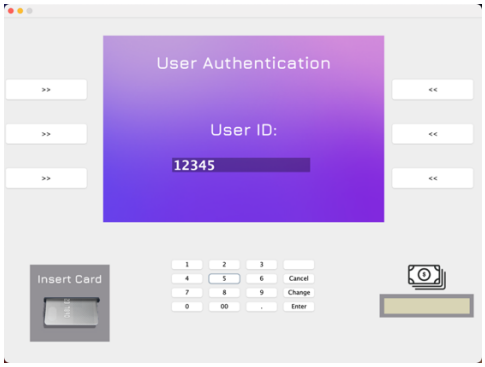


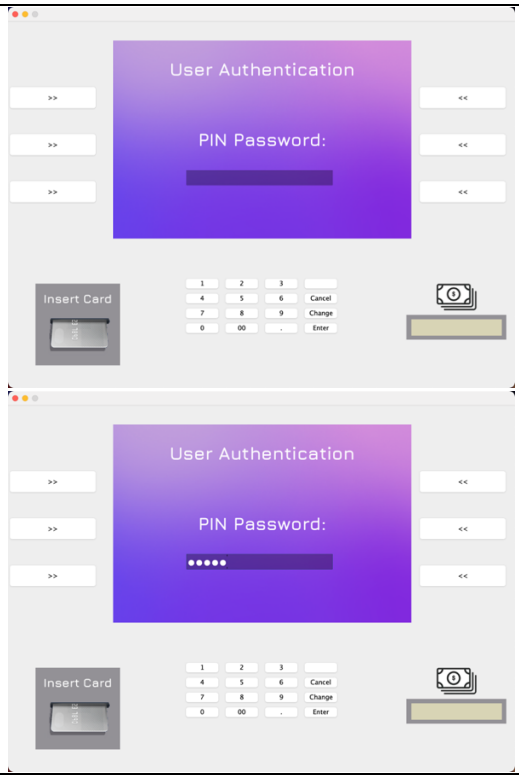
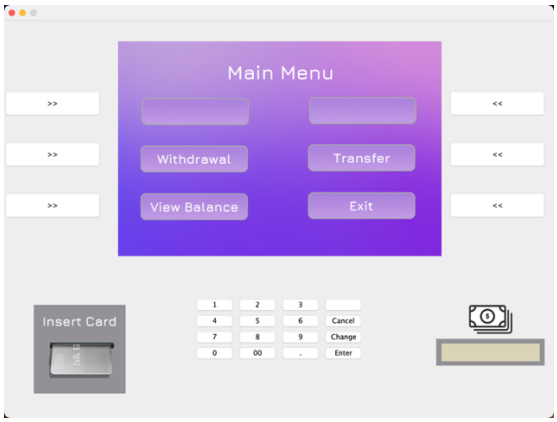
By using the above program, it will get the values entered: User ID and Amount. If transferMoneyUserID and transferMoneyAmount are empty, exist and return to the main menu.

VALID TEST CASE

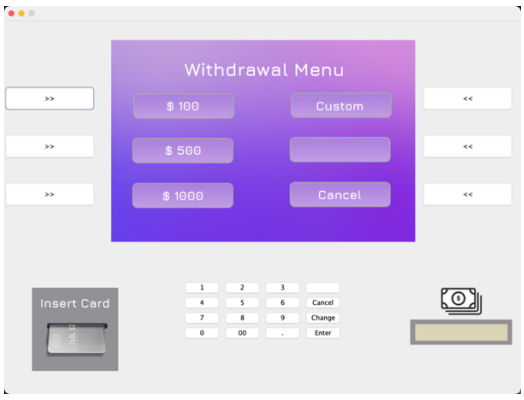
[Account number: 12345, Password: 54321]

1. User Authentication

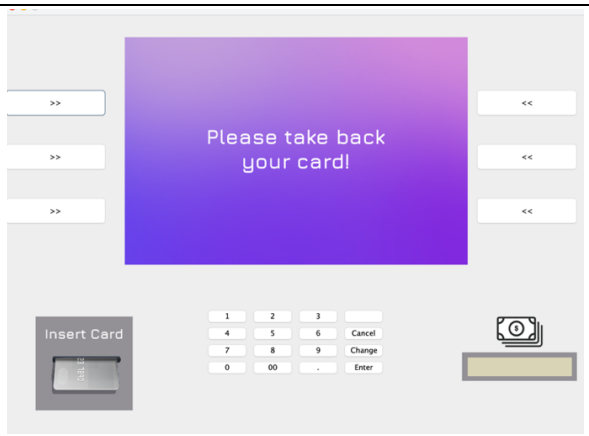
Ask the user to input the account number: Account number: 12345	 

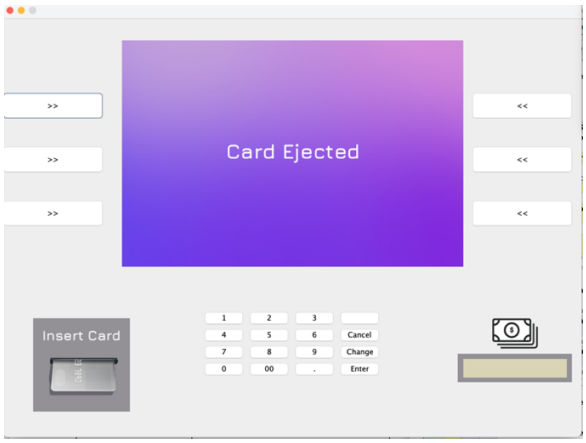
Ask the user to input the account number: Password: 54321	
The user authentication is successful, show the main menu of ATM	

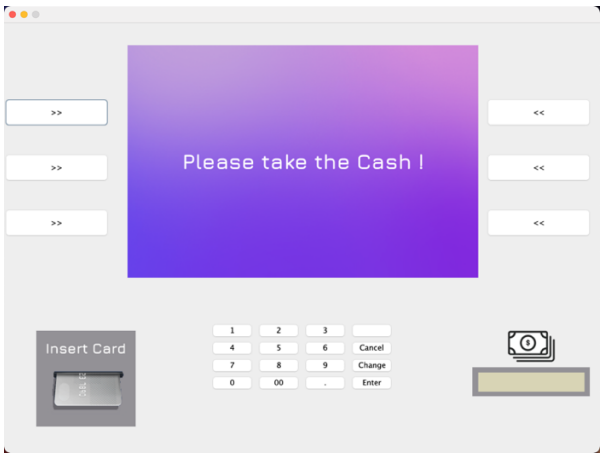
2. Withdrawal Menu

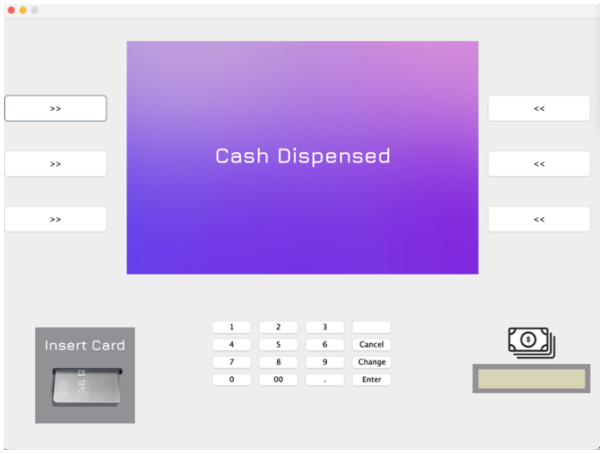
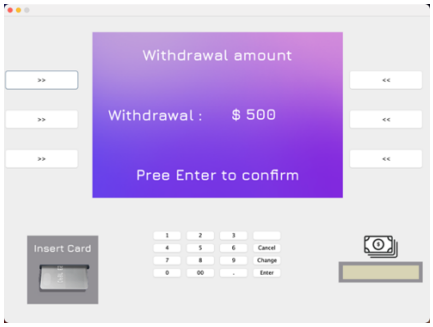
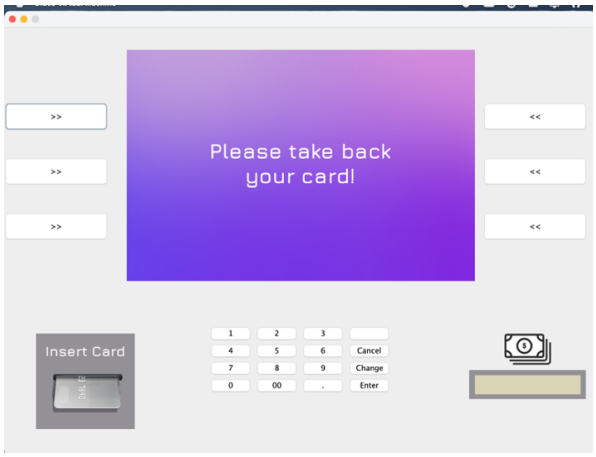
Withdrawal Menu	
After the user press the "withdrawal button", a withdrawal menu is displayed.	
Withdrawal in Fixed Amount	
Withdraw $ 100	After withdrawal $ 100

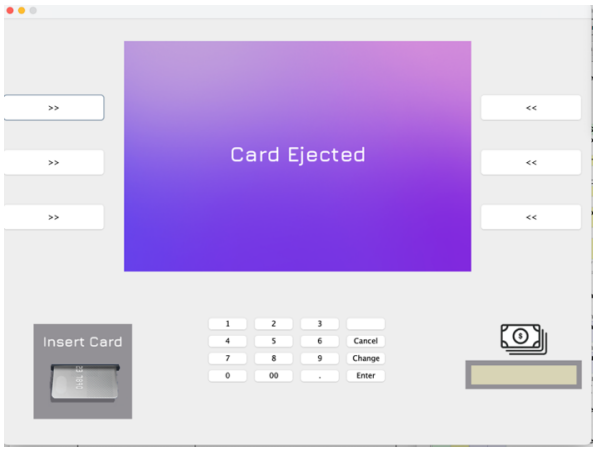

Withdrawal amount
Withdrawal : $ 100
Pree Enter to confirm
Insert Card

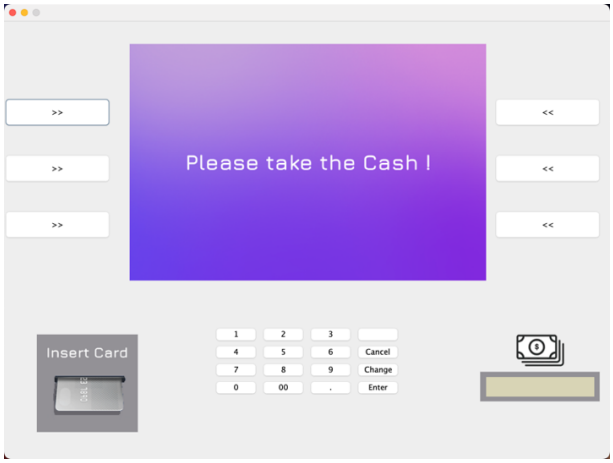

Please take back
your card!
Insert Card

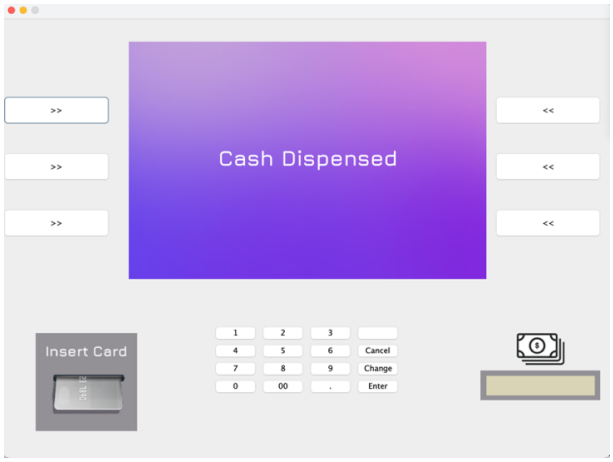
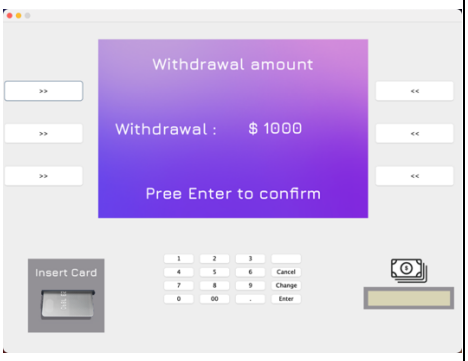
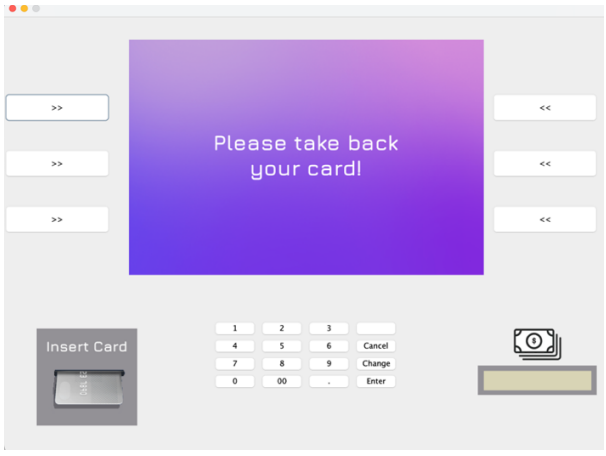

Card Ejected
Insert Card

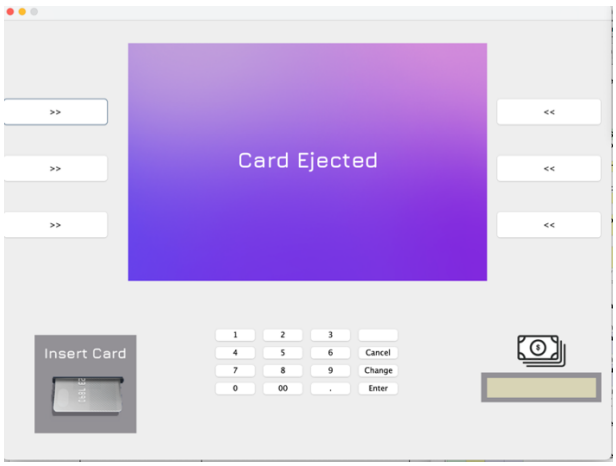
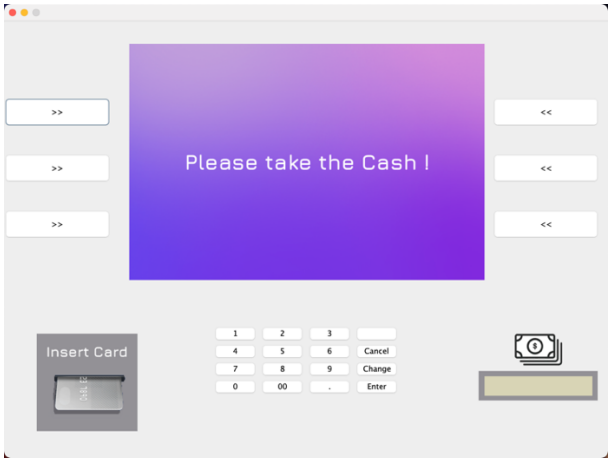
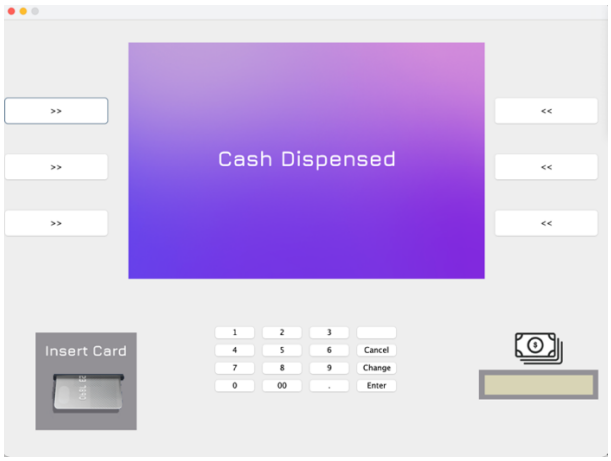

Please take the Cash !
Insert Card

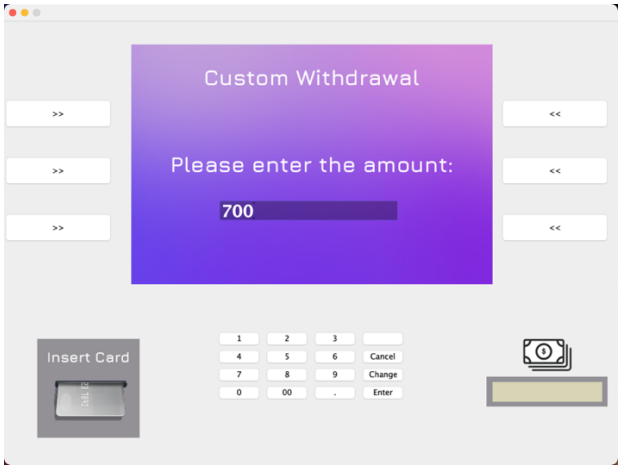
<table>
<tr><td></td><td> 
</td></tr>
<tr><td>Withdraw $ 500
 
</td><td>After withdrawal $ 500
 

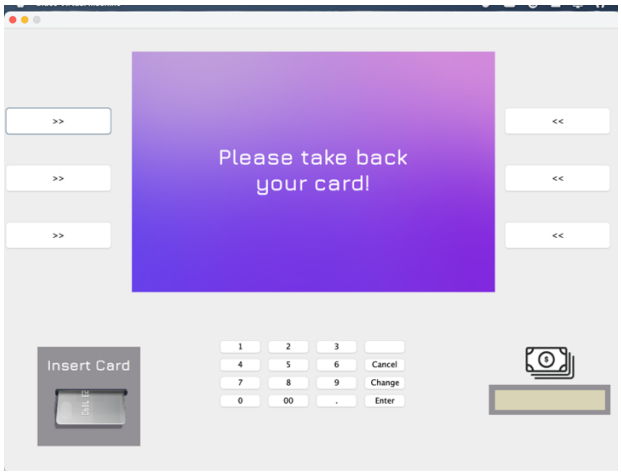
 
</td></tr>
</table>

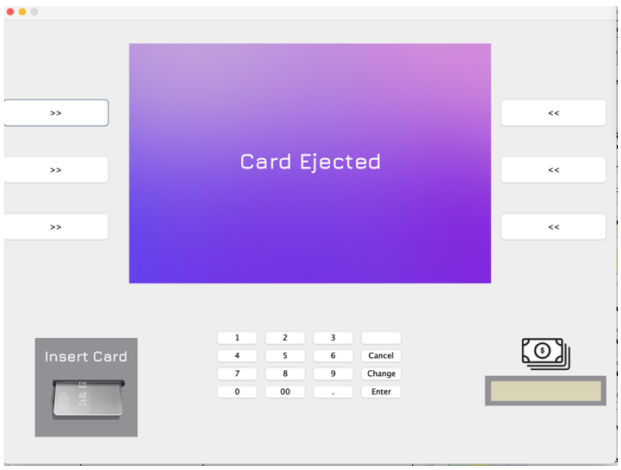
<table>
<tr>
<td></td>
<td> 

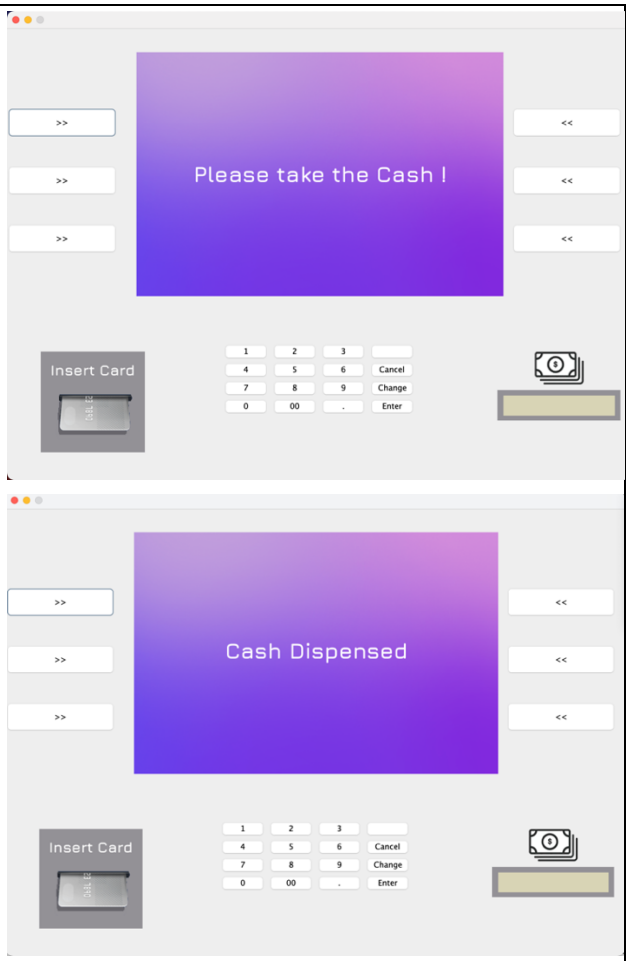

</td>
</tr>
<tr>
<td>Withdraw $ 1000
 
</td>
<td>After withdrawal $ 1000
 
</td>
</tr>
</table>

	  
Withdrawal in Custom Amount	
Withdraw money in custom	After withdrawal $700 (custom enter)

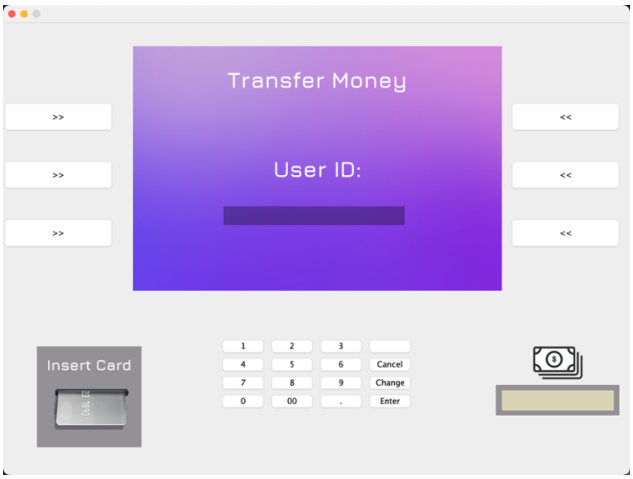

Custom Withdrawal
Please enter the amount:
700
Insert Card
Cancel
Change
Enter

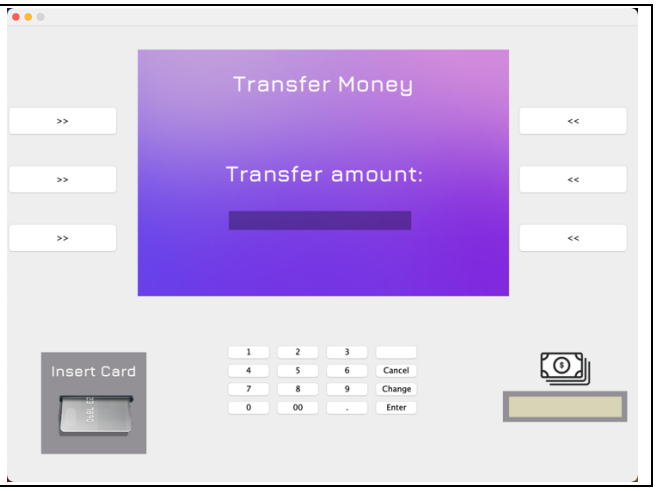

Please take back
your card!
Insert Card
Cancel
Change
Enter

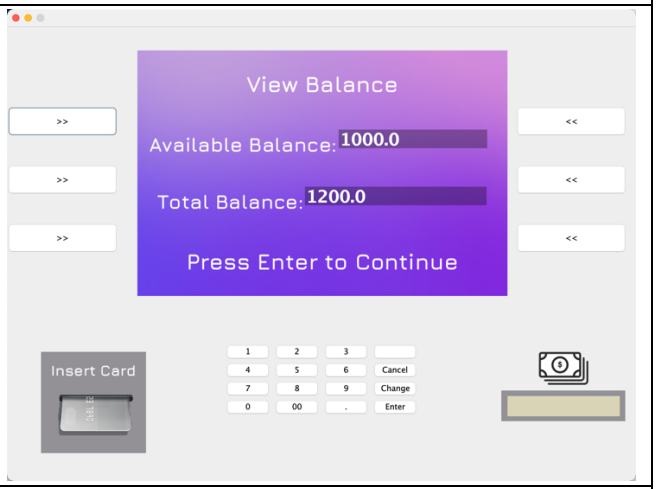
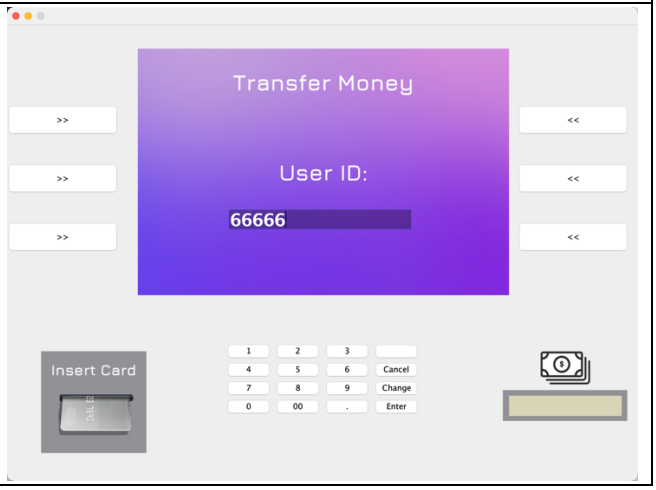

Card Ejected
Insert Card
Cancel
Change
Enter

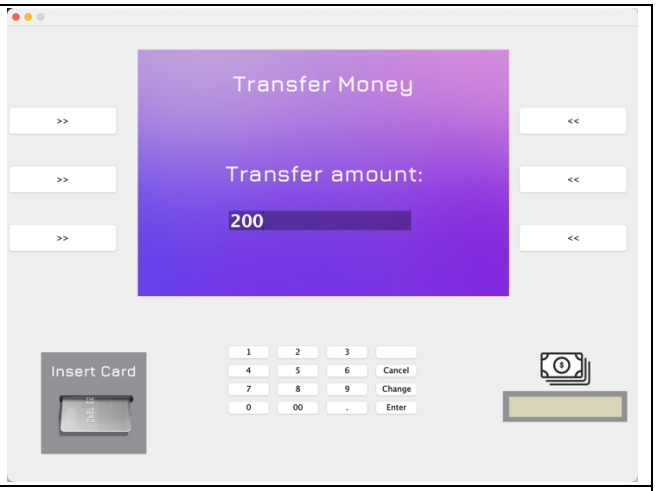
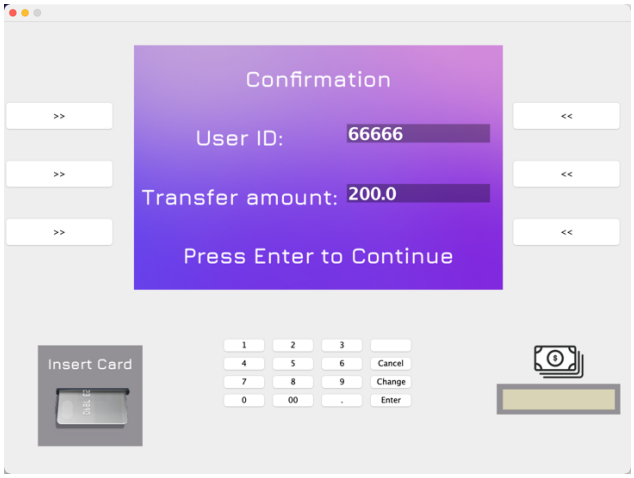
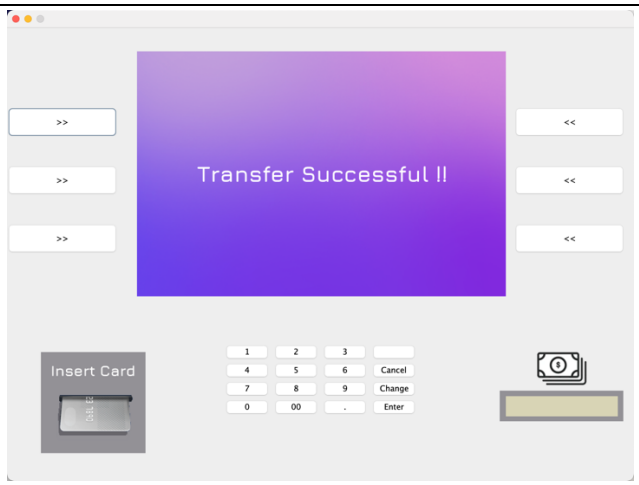
	

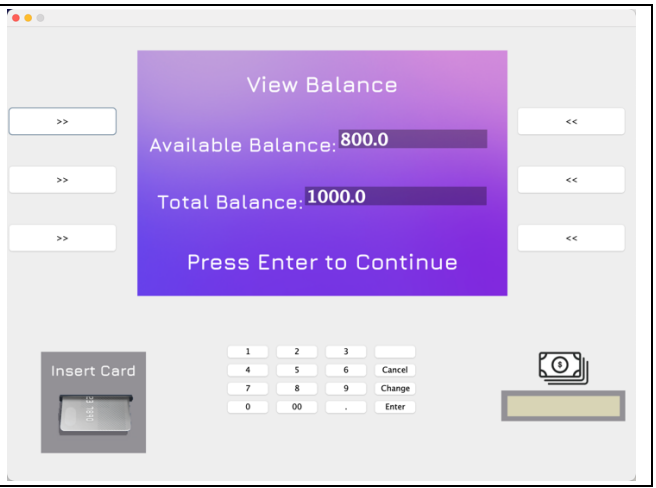
3. Transfer money

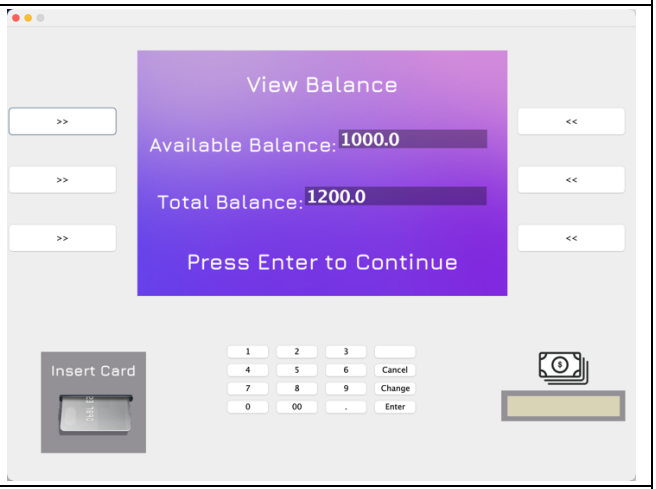
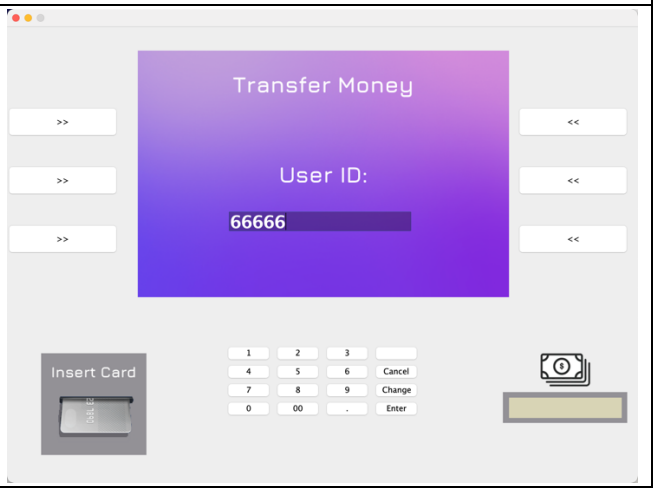
User Authentication and Transfer Amount	
Input the account number of another user	

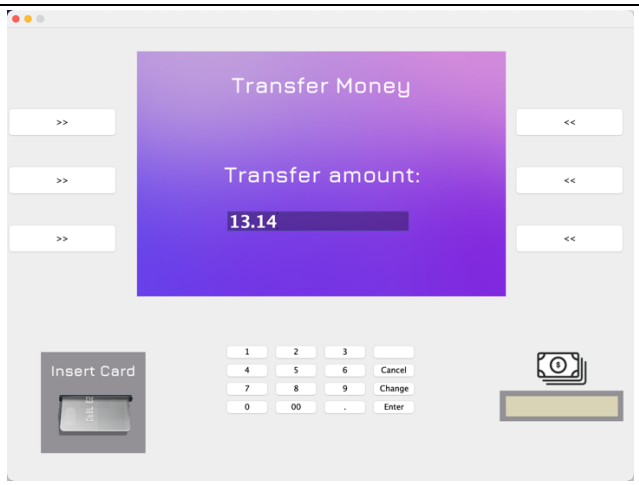
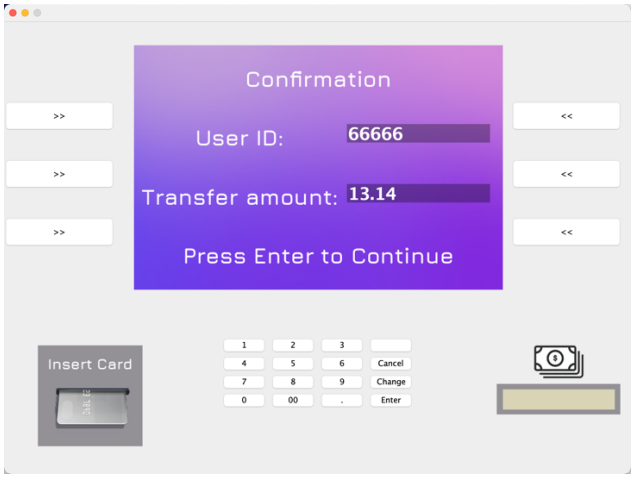
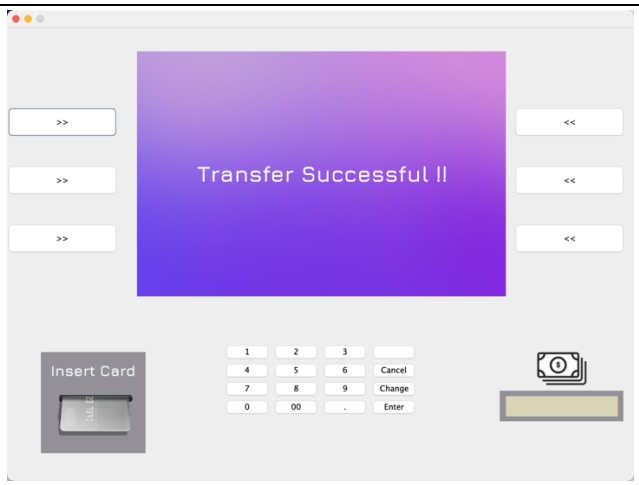
Input the transfer amount to another user	

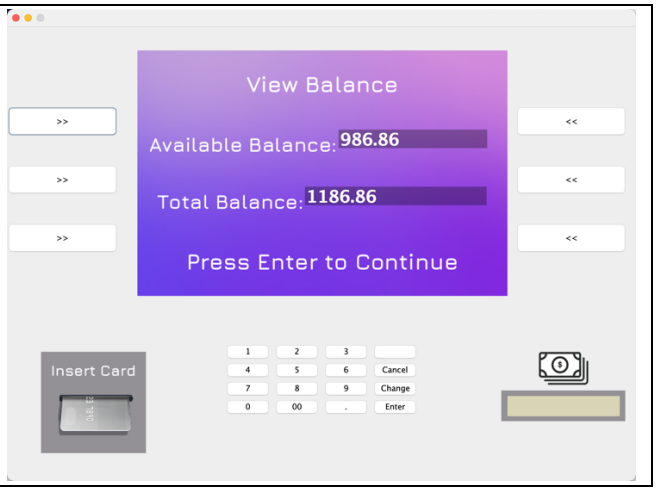
Case 1: User transfer HKD 200 to another user (Integer)	
Initial Balance before transfer (Login Account: 12345)	
Initial Balance before transfer (Login Account: 66666)	

Transfer to another account (66666) with HKD $200	
Double confirming	
Complete	

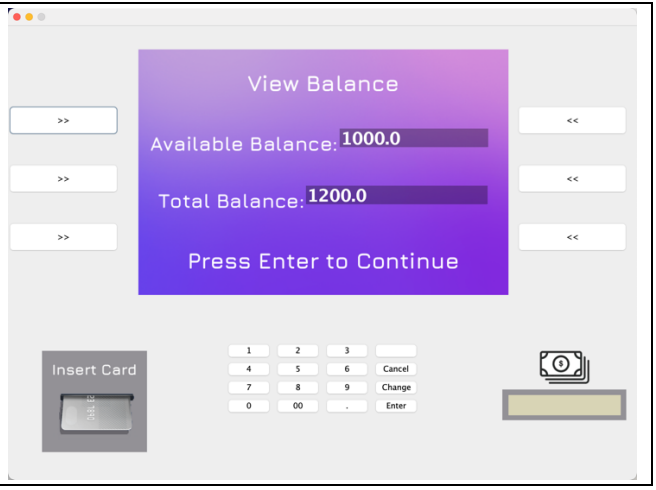
After Transfer Balance (Account: 12345)	

Case 2: User transfer HKD $13.14 to another user (Decimal place)	
Initial Balance before transfer (Login Account: 12345)	
Initial Balance before transfer (Login Account: 66666)	

Transfer to another account (66666) with HKD $13.14	
Double confirming	
Complete	

After Transfer Balance (Account: 12345)	

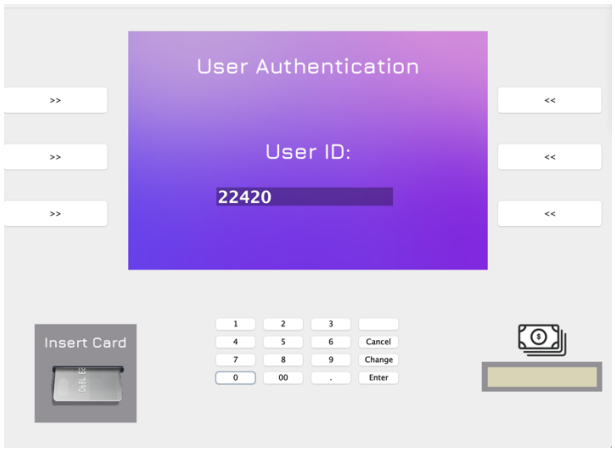
4. View balance

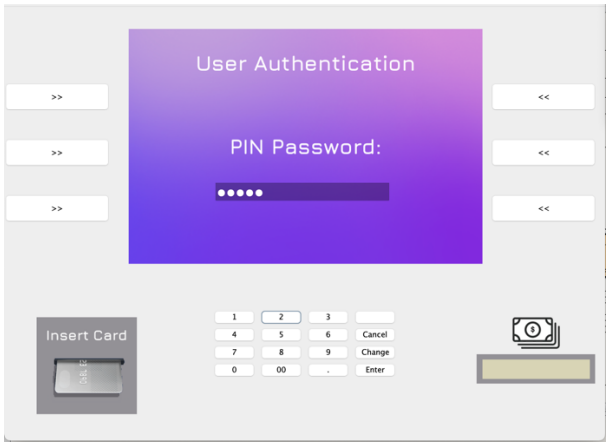
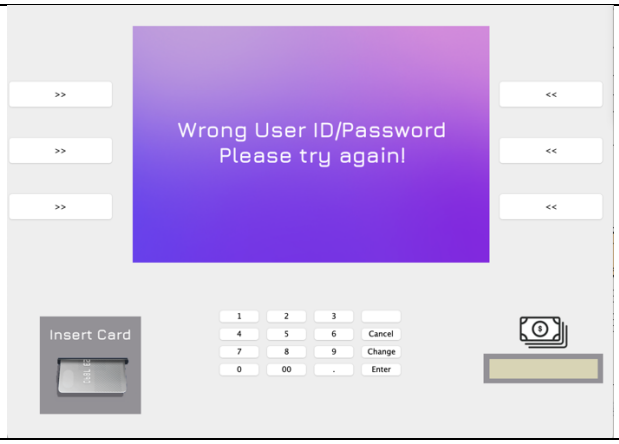
When the user press “view balance” button, the available and total balance is showed:	

UNSUCCESSFUL CASE

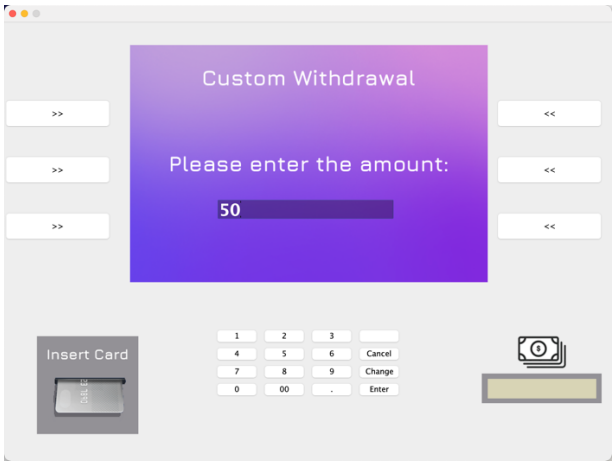
[Account number: 22420, Password: 02422]

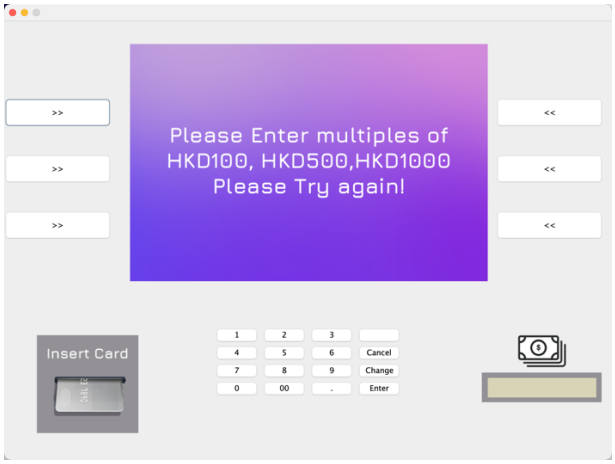
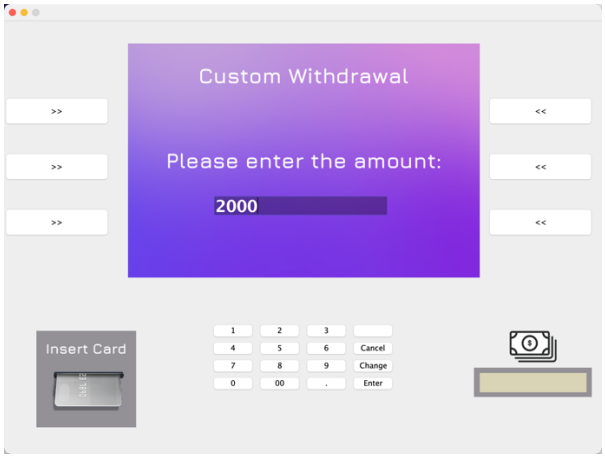
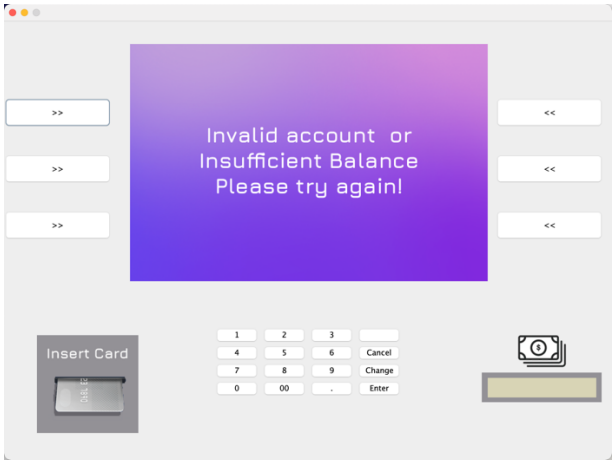
1. User Authentication

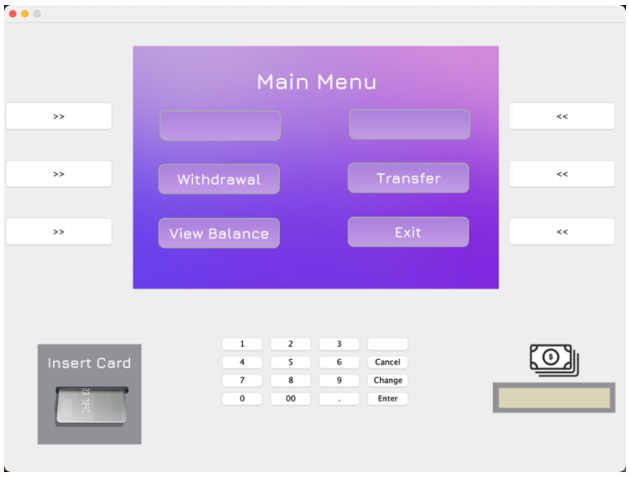
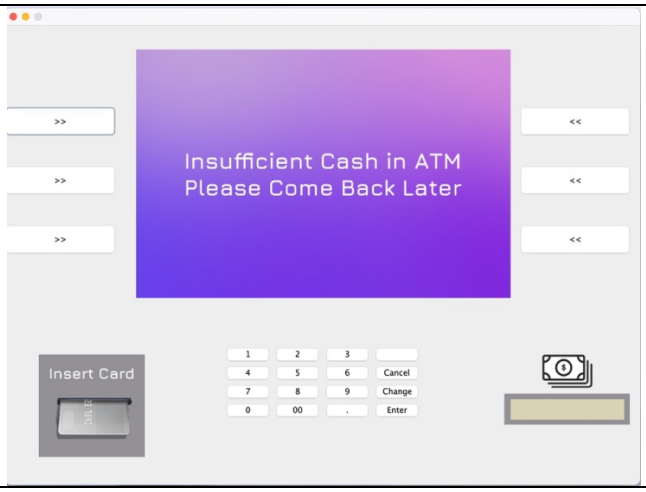
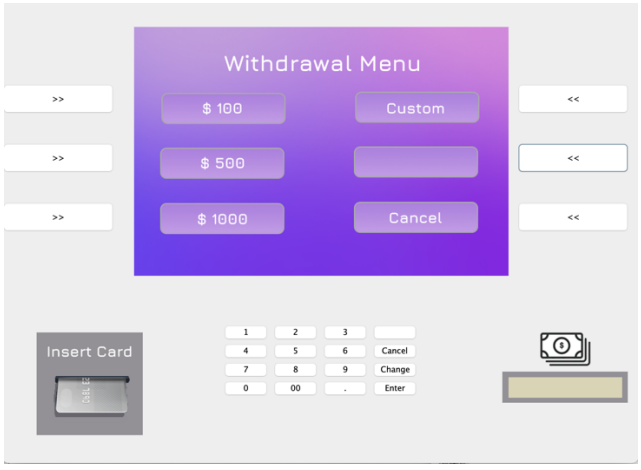
Ask the user to input the account number: Account number: 22420	

Ask the user to input the account number: Password: 02422	
The user authentication is unsuccessful, an error message is displayed	

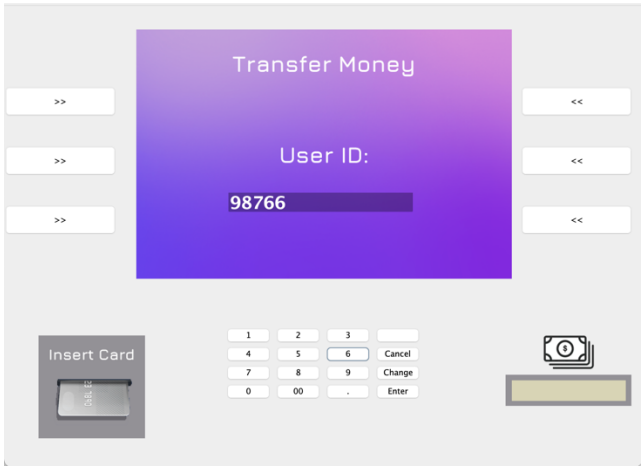
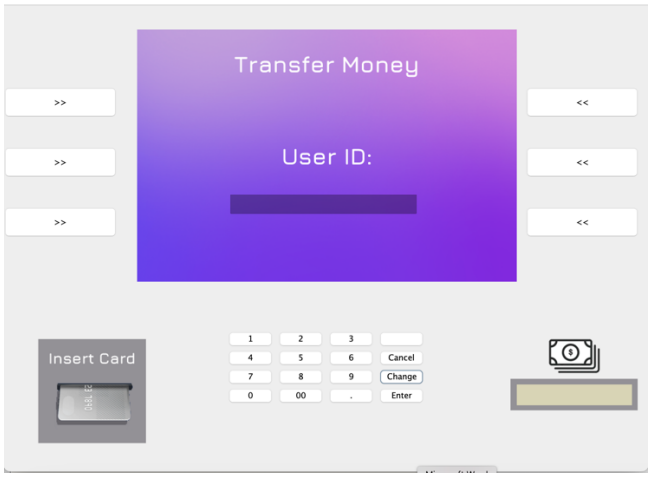
2. Withdrawal Menu

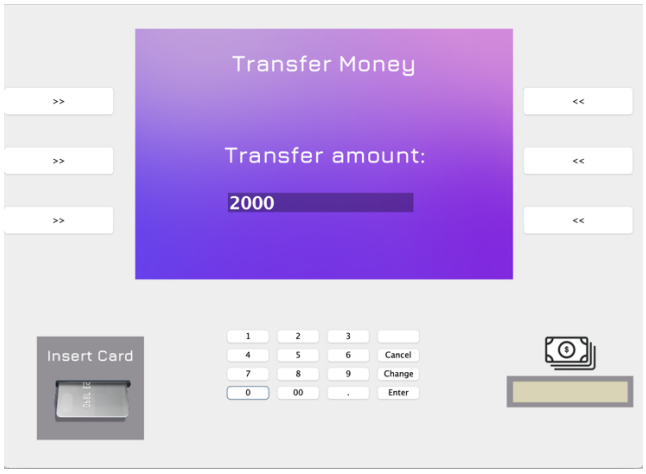
Withdraw $ 50 (Not multiple of 100)	

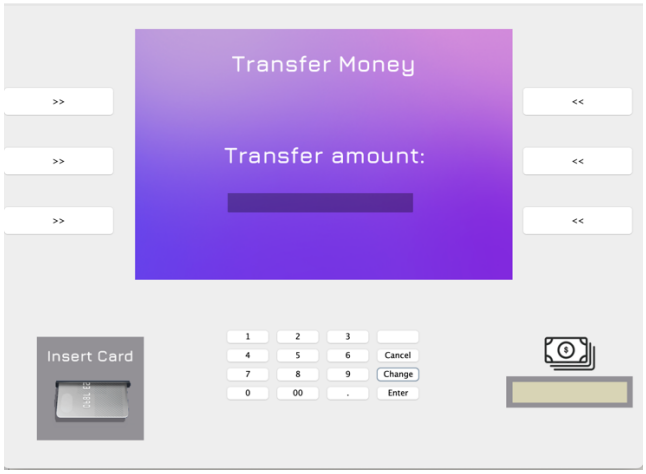
	
Withdraw $2000 (Insufficient funds in the account)	 

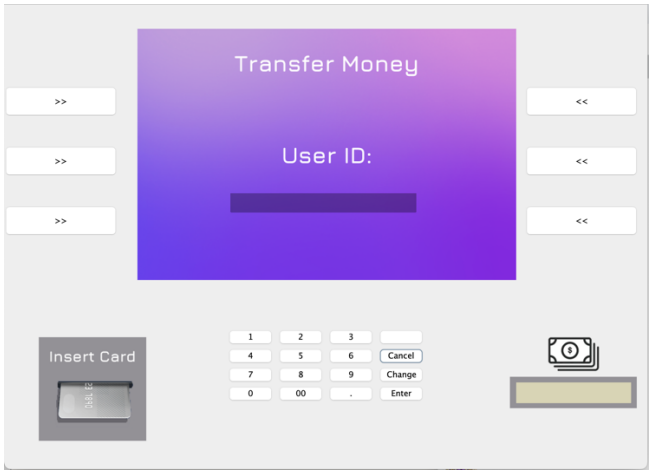
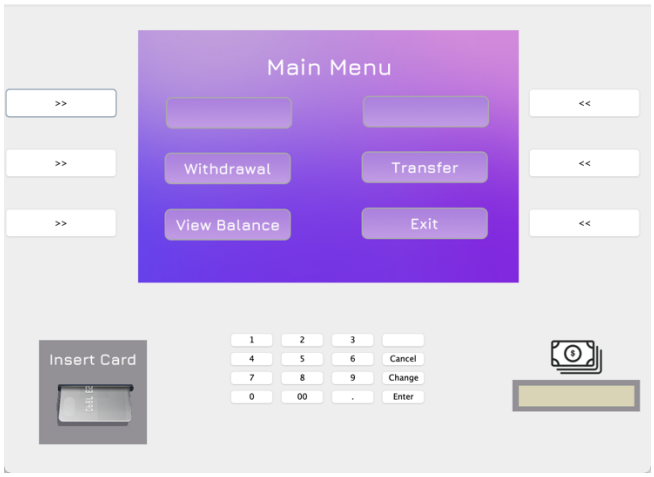
After unsuccess withdrawal account balance form above	
Not sufficient cash in the ATM	
Press the invalid button	

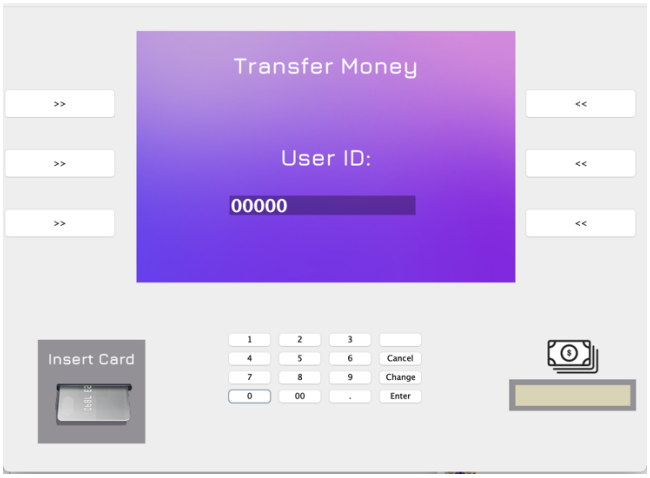
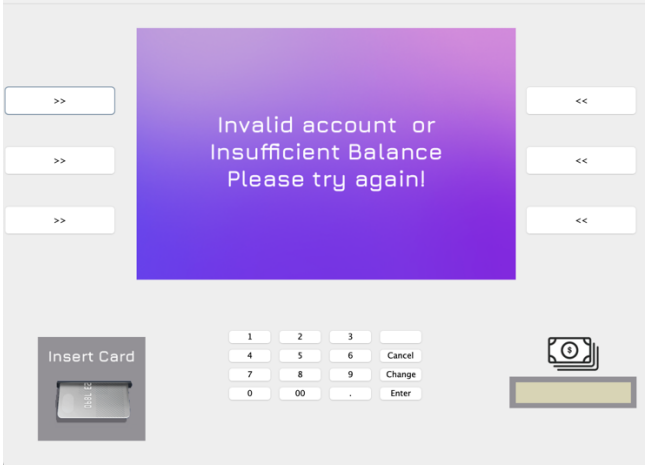
3. Transfer money

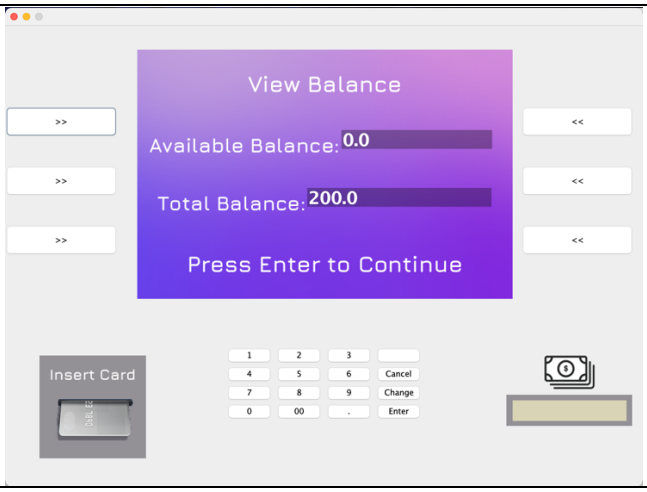
Case1: User re-enter account number when transfer	
When user input wrong amount, user can press the “change” to re-enter the account number. (Wrong account number:98766 Correct account number:98765)	
Screen with the empty text field	

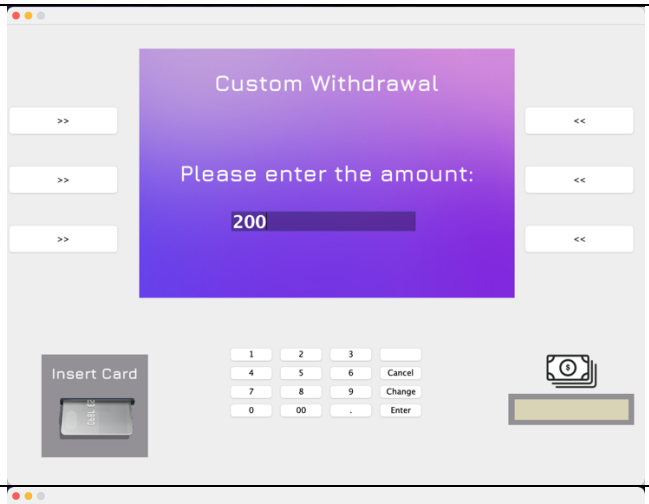
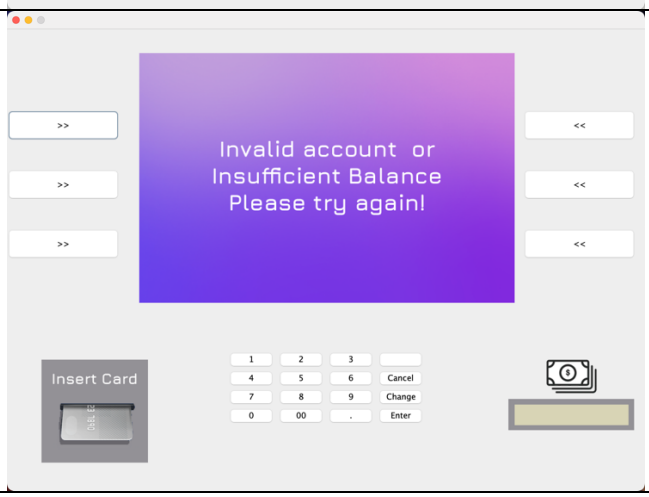
Case 2: User re-enter transfer amount when transfer	
When user input wrong amount, user can press the “change” to re-enter the transfer amount. (Wrong amount:2000 Correct amount:200)	

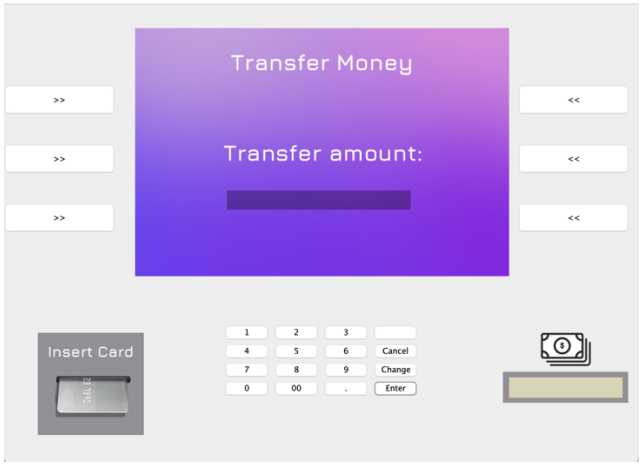
Screen with the empty text field	

Case 3: User cancelling the transaction:	
When a user wants to cancel the transfer, the user press cancel transaction.	
Return to the main menu	

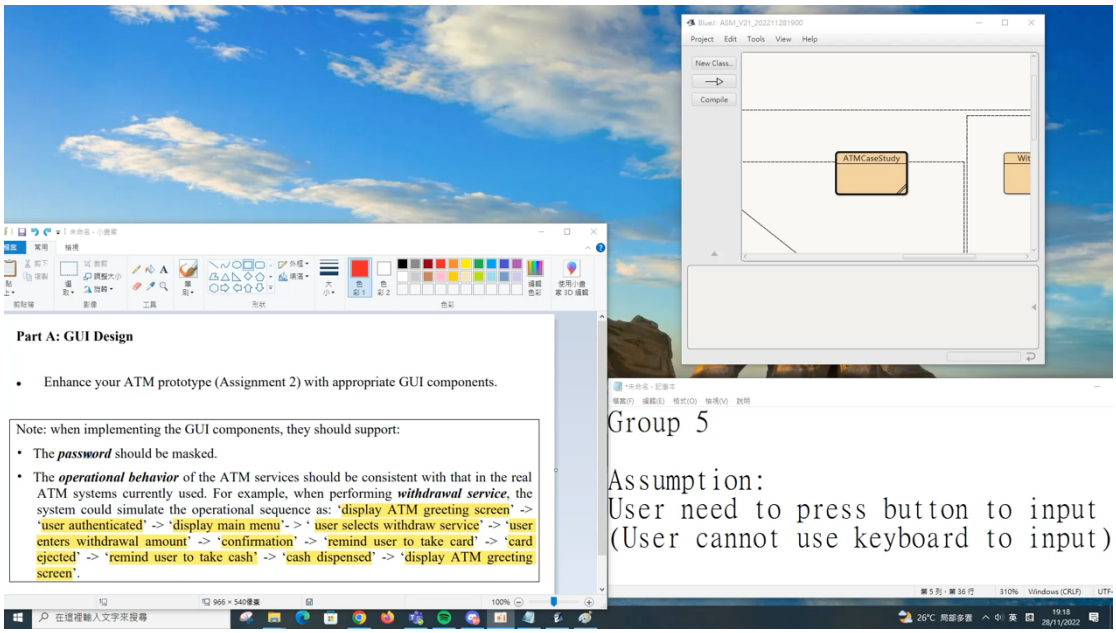
Case 4: The user's account number does not exist	
Enter the invalid account number	
User will receive an error message when the transfer account does not exist.	

Case 5: Not enough balance in user account	
The available balance and total balance of another client	

Enter the transfer amount	
User will receive an error message when the transfer account does not exist.	

Case 6: User input nothing	
When the user press “enter”, the program cannot run due to the insufficient value	

DEMONSTRATION OF THE ATM PROTOTYPE



https://youtu.be/LQUZUTbeUxQ

HTTPS://DRIVE.GOOGLE.COM/FILE/D/1U_T7CBKWKWUDRLRUCRRRAFJNGY1NBHM7/VIEW?USP=SHARING

DESCRIPTION OF TEAMWORK

THE DIVISION OF JOB DUTIES

Coding Team (Include Demo)	- Ng Ka Ki - Fung Cheuk Hin - Wong Hiu Chun
Report Team (Include GUI interface)	- Li Wing Sum - Fung Pui Kiu - Wan Hoi Nam

TIMELINE OF THE WORK DONE

First Meeting - Assignment Flow - Job Distribution	8/11/2022
Second Meeting - Job Reporting	10/11/2022
GUL confirming	12/11/2022
Finish GUI	15/11/2022
Third Meeting - GUI	17/11/2022
Coding Meeting	21/11/2022
Coding Edit	17- 28/11/2022
Reporting Edit	22- 28/11/2022
Final Check	27-28/11/2022

GROUP LEARNING EXPERIENCE

This is the second group assignment we've done together, so we're more in tune with our team. While our distribution is the same as the previous assignments, we always feel free to do as much as possible to help each other whenever we need support.

In the GUI design section, we first reviewed how we use the ATM machine, so we can have a basic idea of how we can design the ATM machine's interface and those functions. Next, we start our design for the coding team to have more images on their program.

For the program code of the ATM machine, we must confirm the GUI interface with different functions and designs with our teammates, and then we can edit our program. In the process of editing programs, we always face the illusion of pngs that cannot be connected to classes and interfaces. Also, our main difficulty is using the ATM to perform different functions. After several days of many tests, we finally completed the smooth coding system for the ATM.

After completing this assignment, we have more ideas on how to do the GUI of the interface and the connection of different classes in the program. In this assignment, we have a lot of hands-on time with Java coding and can make great progress in writing Java code and programming systems. We hope that through this experience, we can do better in our next projects.