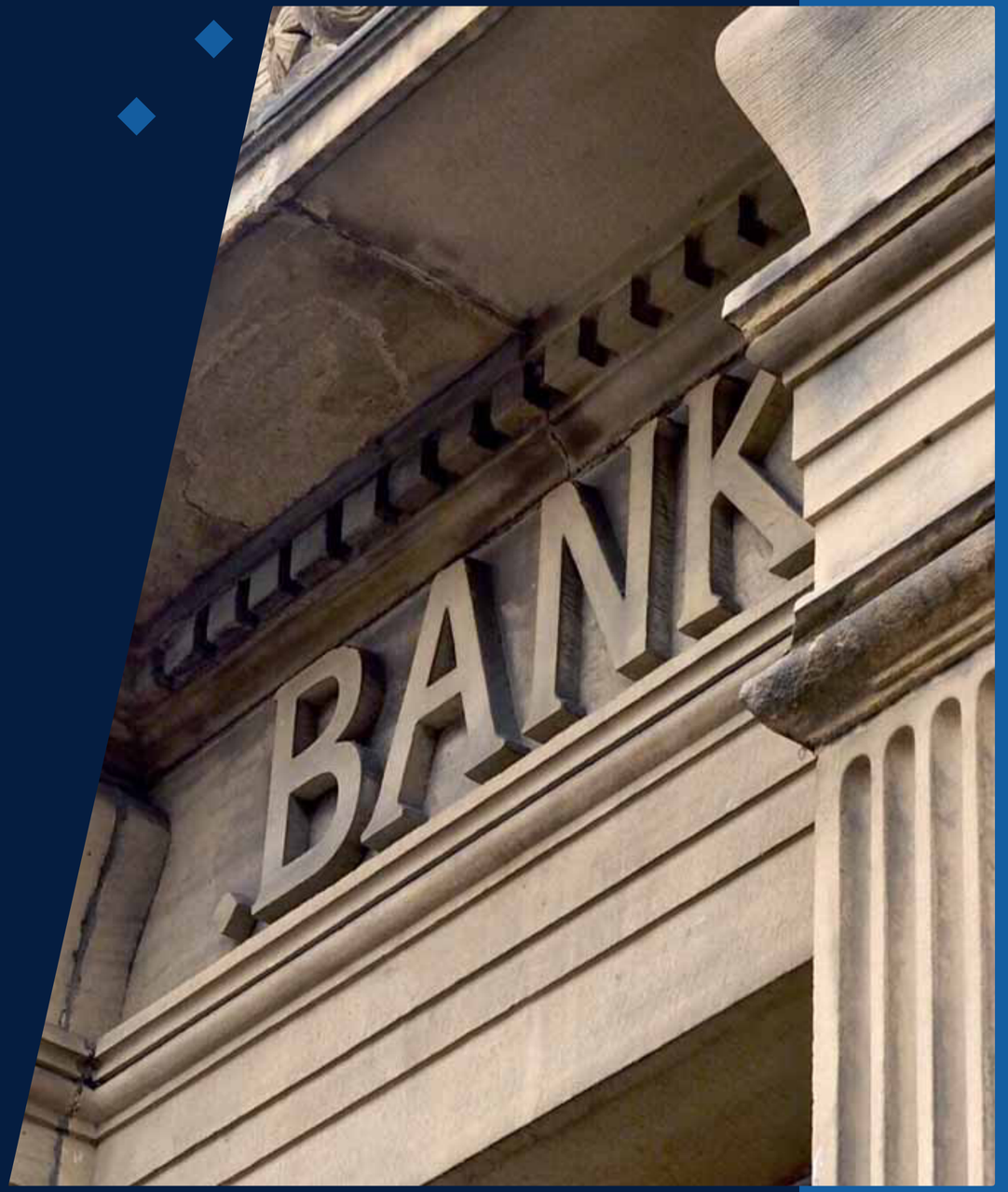
DDJK BANKING INSTITUTION



OVERVIEW

01
Our Team

02
Introduction

03
Flowchart

04
Scenarios

05
UML Diagram

06
OOP Principles

OUR TEAM

Daniel Zavala-Palafox

Donovan Hall

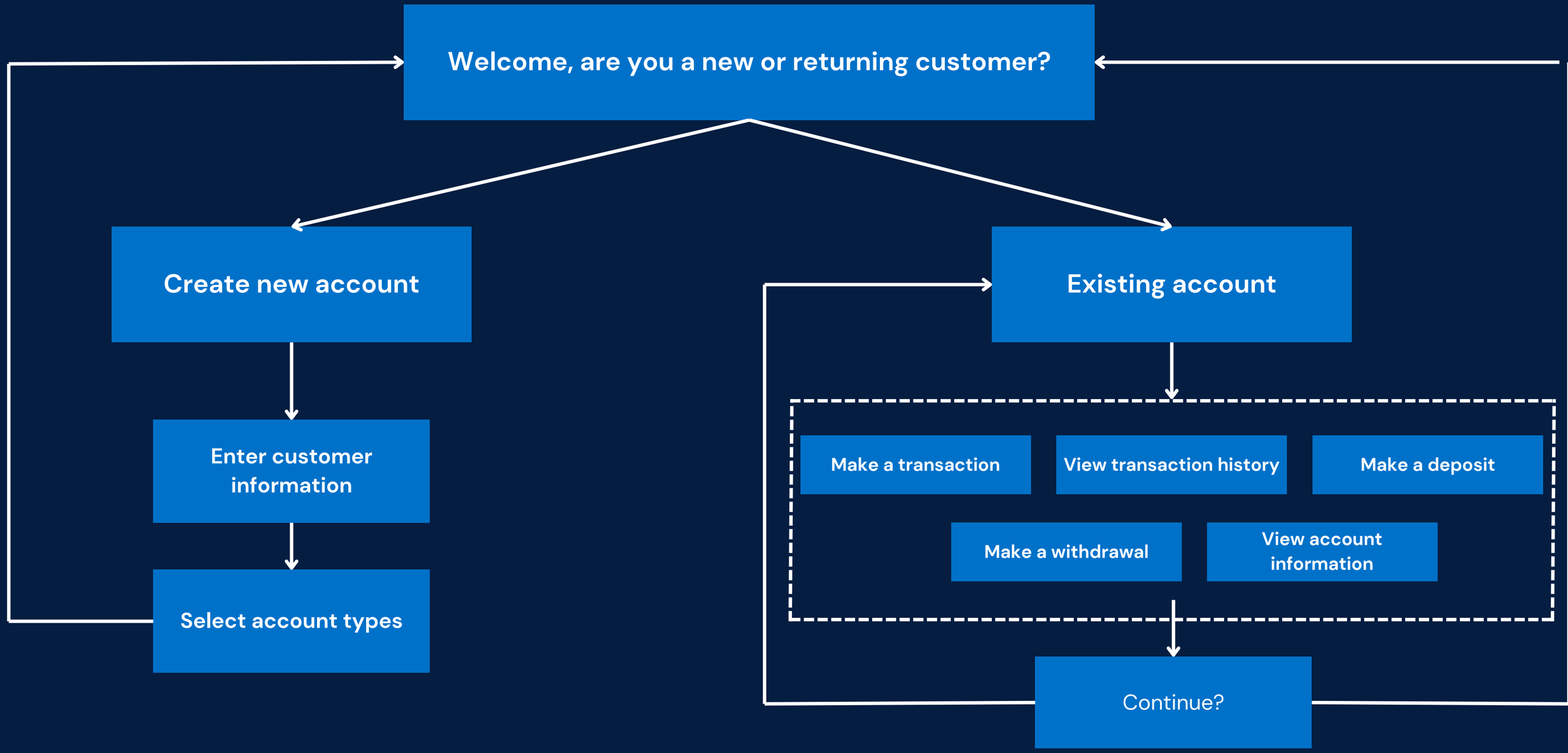
Kush Patel

Jordan Eccleston

DDJK BANK

Welcome to the DDJK Banking Institution, where your financial well-being is our top priority! Whether you're looking for flexible savings accounts with a convenient depositing and withdrawing process or reliable checking accounts to make and manage your transactions, we're here to make your banking experience secure, transparent, and effortless.

OVERVIEW FLOWCHART



SCENARIO 1: KIRBY SMART OPENS AN ACCOUNT

After his big win in the National Championship, Kirby Smart heads over to DDJK Bank to deposit his newly awarded bonus! He first provides his name, birthday, and social security number into the database.

Kirby then realizes he needs to create a password that fits the bank's specific criteria. His obvious choice is: Georgia>Bama99

Kirby decides to open both a Savings and Checking account as he wants to spend a bit of his bonus but also plans to save most of it for the future. He chooses to deposit $500,000 to his savings.

To deposit his earnings into his checking account, Kirby creates a PIN number: 6507. He then deposits $5,000 into the account. With this process completed, Kirby Smart is now an official customer of DDJK Bank!

Step 1

Input Required Information

Step 2

Create Password

Requirements:
10 Characters, 1 Uppercase, 1 Lowercase, 1 digit, **1 special character**

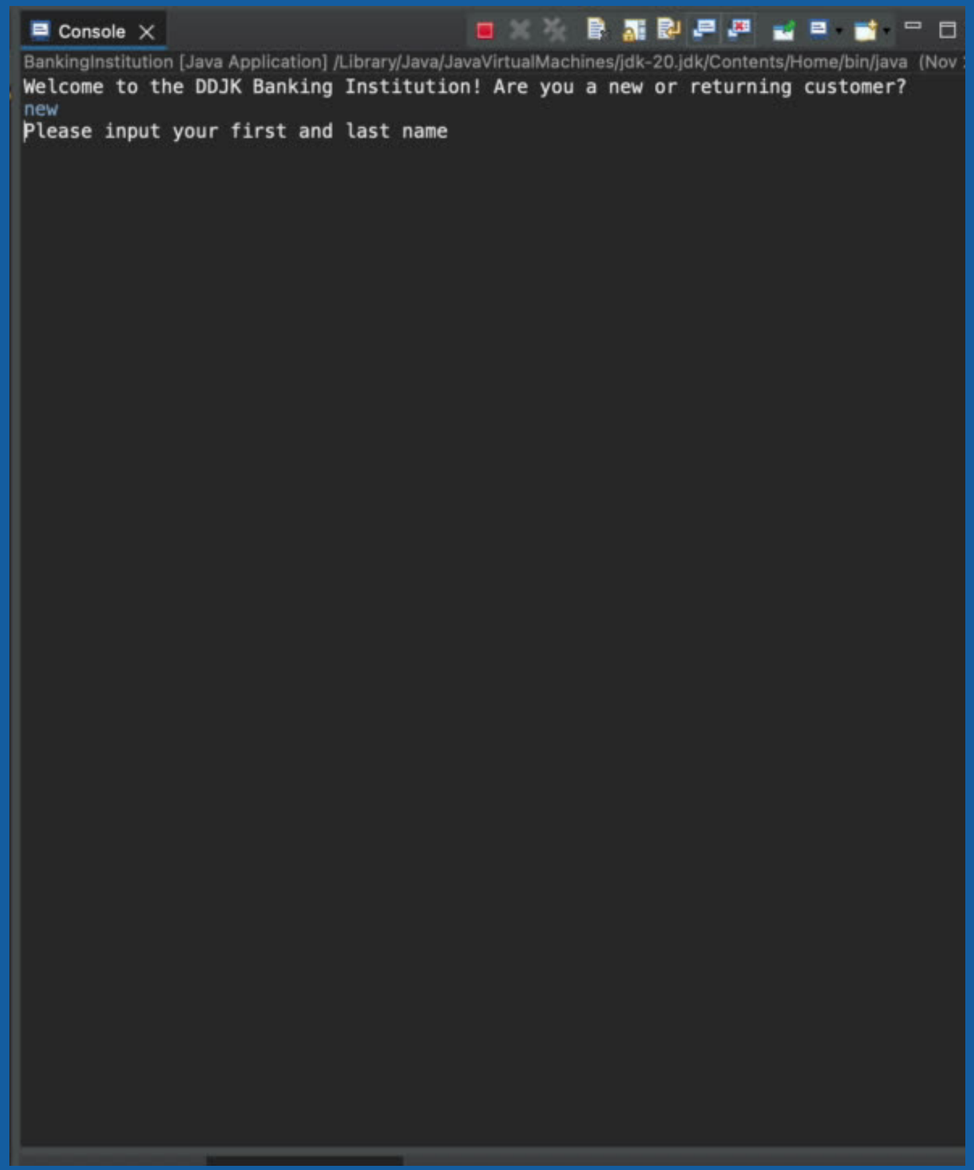
Step 3

Open Savings & Deposit

Step 4

Open Checking & Deposit

LET'S RUN THROUGH SOME CODE!



SCENARIO 2: KIRBY SMART MAKES A TRANSACTION

Now that Kirby has his accounts set up properly, it's time for him to start spending! He starts by signing into his account he recently made and selects "Make a Transaction".

Kirby decides he wants to adopt a bulldog to celebrate his win. The adoption fees add up to $3000. He enters the amount and the system lets him know how much money he has remaining. Since he has some extra cash, he buys his new dog some treats, a new collar, and other supplies for $200.

After he finishes shopping, Kirby wants to make sure that all of his purchases went through properly. He navigates to the menu and selects "View transactions history" and checks over his purchases.

Since everything looked right, he navigated back to the main menu and signed out of his account. Now time to celebrate the win with the team!

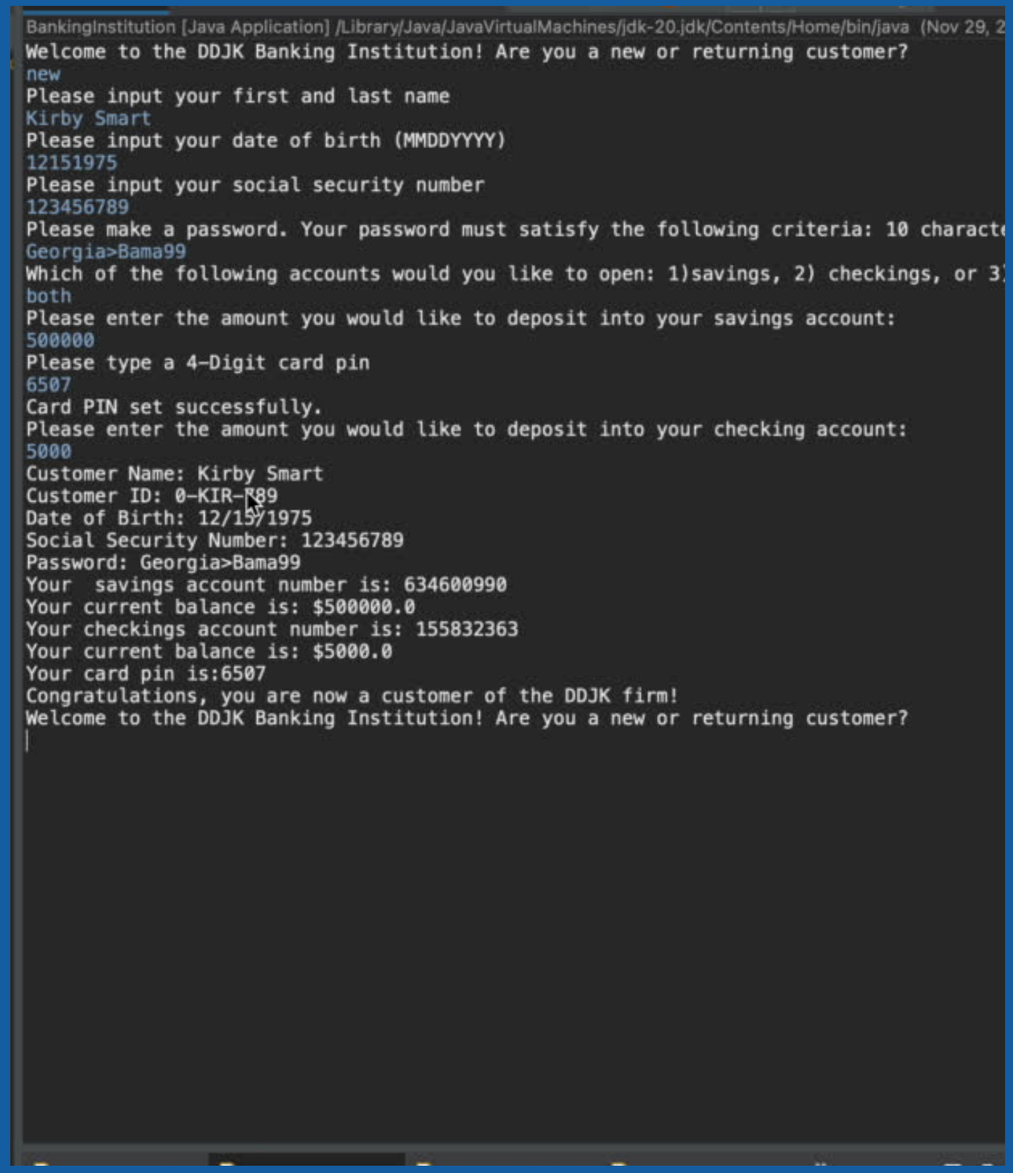
Step 1 **Sign On**

Step 2 **Make Transactions**

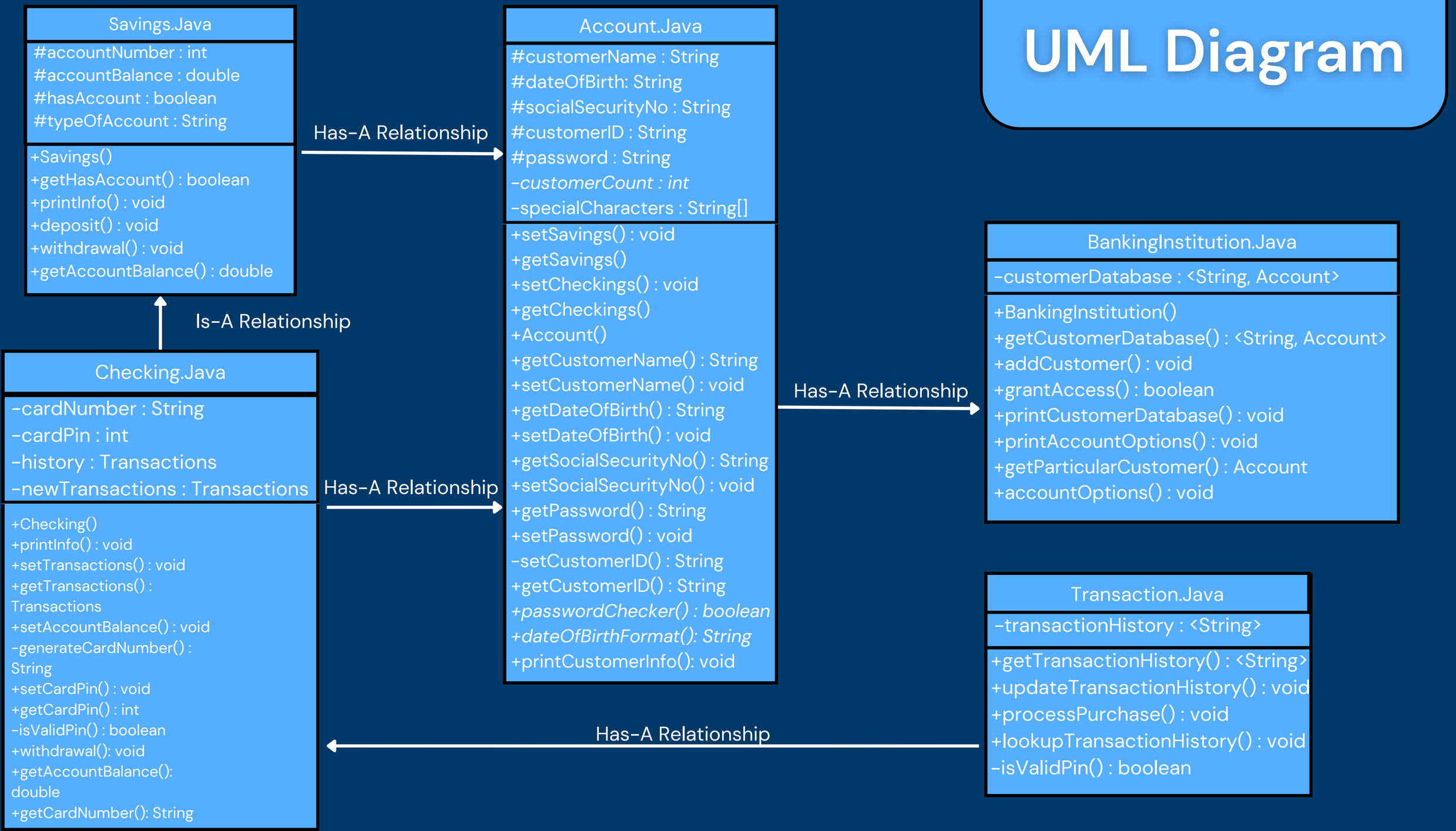
Step 3 **View Transaction History**

Step 4 **Sign Out**

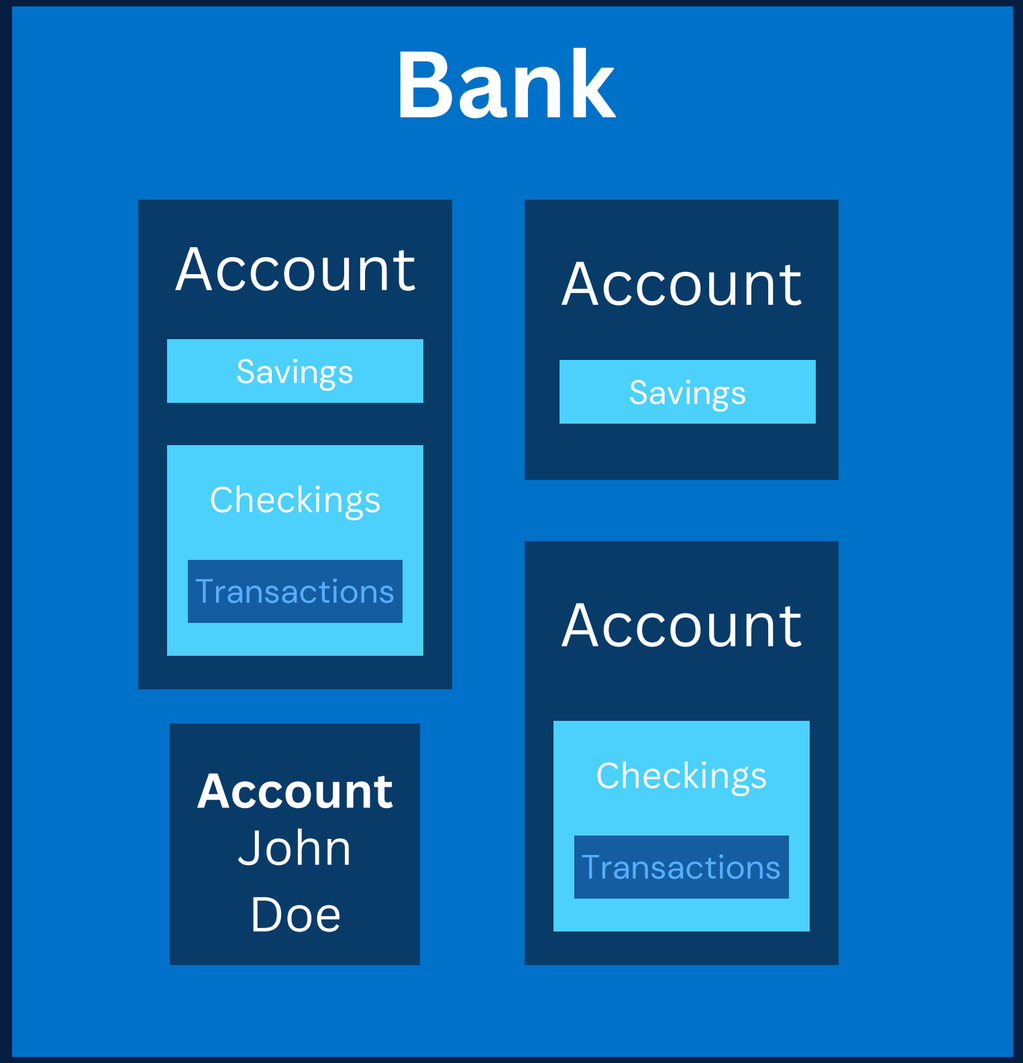
LET'S RUN THROUGH SOME CODE!



UML Diagram


Savings.Java
#accountNumber : int
#accountBalance : double
#hasAccount : boolean
#typeOfAccount : String
+Savings()
+getHasAccount() : boolean
+printInfo() : void
+deposit() : void
+withdrawal() : void
+getAccountBalance() : double
Has-A Relationship
Account.Java
#customerName : String
#dateOfBirth: String
#socialSecurityNo : String
#customerID : String
#password : String
-customerCount : int
-specialCharacters : String[]
+setSavings() : void
+getSavings()
+setCheckings() : void
+getCheckings()
+Account()
+getCustomerName() : String
+setCustomerName() : void
+getDateOfBirth() : String
+setDateOfBirth() : void
+getSocialSecurityNo() : String
+setSocialSecurityNo() : void
+getPassword() : String
+setPassword() : void
-setCustomerID() : String
+getCustomerID() : String
+passwordChecker() : boolean
+dateOfBirthFormat(): String
+printCustomerInfo(): void
Is-A Relationship
Checking.Java
-cardNumber : String
-cardPin : int
-history : Transactions
-newTransactions : Transactions
+Checking()
+printInfo() : void
+setTransactions() : void
+getTransactions() : Transactions
+setAccountBalance() : void
-generateCardNumber() : String
+setCardPin() : void
+getCardPin() : int
-isValidPin() : boolean
+withdrawal(): void
+getAccountBalance(): double
+getCardNumber(): String
Has-A Relationship
BankingInstitution.Java
-customerDatabase : <String, Account>
+BankingInstitution()
+getCustomerDatabase() : <String, Account>
+addCustomer() : void
+grantAccess() : boolean
+printCustomerDatabase() : void
+printAccountOptions() : void
+getParticularCustomer() : Account
+accountOptions() : void
Has-A Relationship
Transaction.Java
-transactionHistory : <String>
+getTransactionHistory() : <String>
+updateTransactionHistory() : void
+processPurchase() : void
+lookupTransactionHistory() : void
-isValidPin() : boolean

OOP Design



OOP Concepts

Inheritance

```
public class Checking extends Savings {

    Random random = new Random();

    private String cardNumber;
    private int cardPin;
```

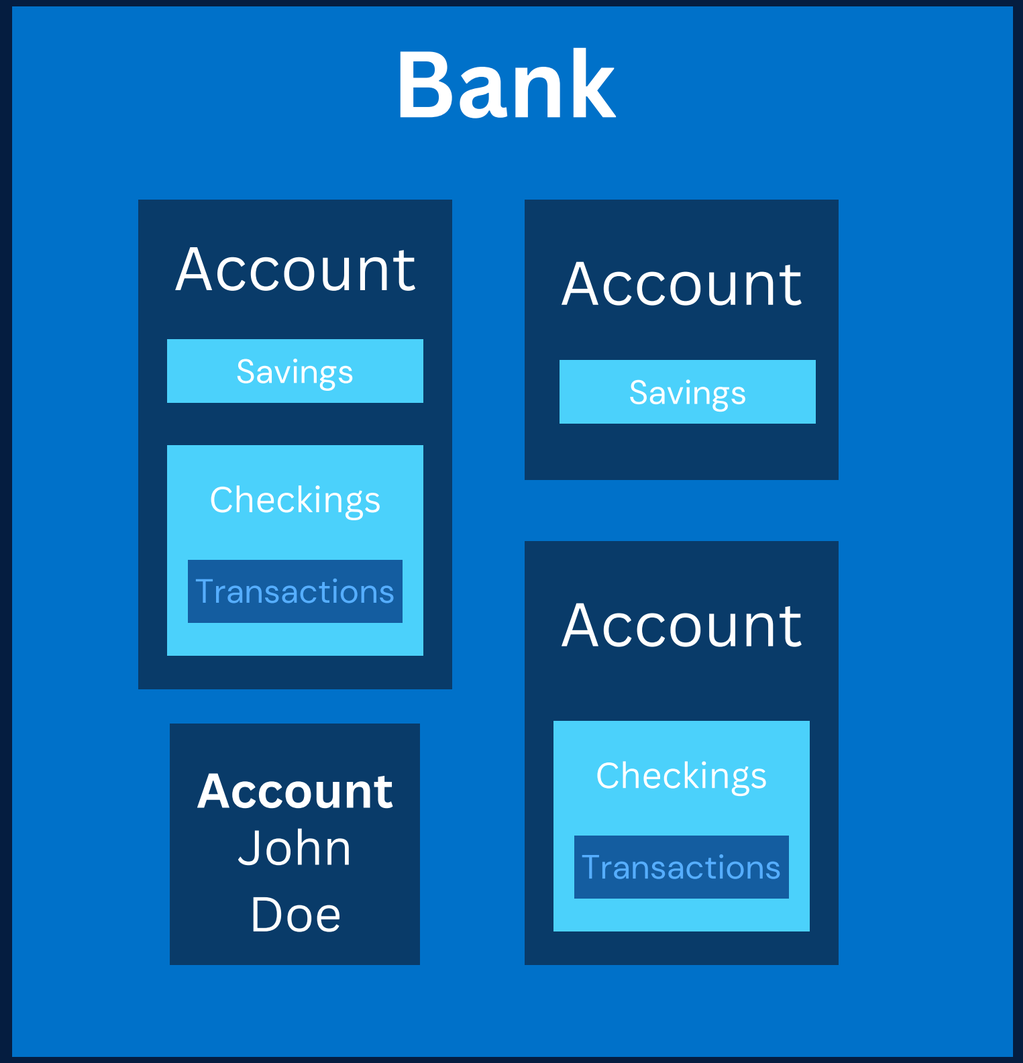
Checking class inherits attributes like accountNumber and accountBalance from the Savings class, but adds other options like cards and transactions.

Composition

```
private Transactions history;
private Transactions newTransactions;
```

Checking class uses objects to reference the Transactions class, creating a **Has-a relationship.** This allows for better code organization because the entire transaction process can be established in its own class rather in than the Checking class.

OOP Design



OOP Concepts

Encapsulation

```
private String cardNumber;
private int cardPin;
```

```
// Method to set the 4-digit card PIN
public void setCardPin(int pin) {
    if (isValidPin(pin)) {
        this.cardPin = pin;
        System.out.println("Card PIN set successfully.");
    } else {
        System.out.println("Invalid PIN format. Please enter a 4-digit PIN.");
    }
}

// Method to get the card PIN - for Transaction
public int getCardPin() {
    return this.cardPin;
}
```

The checking class has the private variable ‘cardPin’ and gains access to it through the public methods ‘setCardPin’ and ‘getCardPin’.

Polymorphism

```
public void printInfo() {

    if (getHasAccount()) {
        System.out.println("Your " + typeOfAccount +  " account number is: " + accountNumber);
        System.out.println("Your current balance is: $" + accountBalance);
    } else {
        System.out.println("You have " + typeOfAccount+ " account");
    }


}
```

```
@Override
public void printInfo() {
    super.printInfo();
    if (hasAccount) {
        System.out.println("Your card pin is:" + getCardPin());
    }

}
```

The savings and checking class both have the method ‘printInfo()’ but the checking class overrides the method for its own implementation. When they are called, they do different things based on the object they are referring to.

Collections

Array

```
private static String[] specialCharacters = {"~", "`", "!", "@", "#", "$", "%", "^", "&",
        "*", "(", ")", "-", "_", "+", "=", "{", "[", "}", "]", "|", "\\", ":", ";", "\"",
        "\'", "<", ",", ">", ".", "?", "/"};
```

```
        }

        for (String character: specialCharacters ) {
            if (character.charAt(0) == letter) {
                specialCharacter = true;
            }
        }
    }

    }
```

located in passwordChecker method

- Stores special characters
- passwordChecker method - enhance security
  - For each loop

```
Please make a password. Your password must satisfy the following criteria: 10 characters long, no
password
Password does not meet criteria, please try again
pa$$word
Password does not meet criteria, please try again
pa$$w0rd
Password does not meet criteria, please try again
pa$$w0rd!!!
Password does not meet criteria, please try again
Pa$$w0rd!!!
Which of the following accounts would you like to open: 1)savings, 2) checkings, or 3)both ?
```

ArrayList

```
private ArrayList<String> transactionHistory = new ArrayList<String>();

// Method to retrieve transaction history
public ArrayList<String> getTransactionHistory() {
    return this.transactionHistory;
}

// Method to update transaction history
public void updateTransactionHistory(String transaction) {
    this.transactionHistory.add(transaction);
}
```

- Stores a string of information regarding transactions created under checkings - differentiated by DATETIME data

```
Making a Transaction...
Please enter the amount:
56.78
Enter your 4-digit PIN to proceed with the purchase: 1234
Purchase successful. Your new balance is: $407.87
Do you want to go back to the options menu? (yes/no)
yes
Select one of the following options down below:
1. Make a transaction
2. View transactions history
3. Make a deposit
4. Make a withdrawl
5. View account information
6. Close account
2
Viewing Transaction History...
Transaction History for Account ID: 0-DAN-789
Purchase: -$45.99 on Wed Nov 29 21:40:58 EST 2023
Purchase: -$489.36 on Wed Nov 29 21:41:16 EST 2023
Purchase: -$56.78 on Wed Nov 29 21:42:08 EST 2023
Do you want to go back to the options menu? (yes/no)
```

Collections

HashMap

```
// creates customerID based on information regarding the person
private String setCustomerID(int customerCount, String name, String SSN) {

    String userID = this.customerCount + "-";
    userID = userID + name.substring(0, 3).toUpperCase() + "-";
    userID = userID + SSN.substring(SSN.length() - 3, SSN.length() );
    return userID;

}

public String getCustomerID() {
    return customerID;
}
```

customer ID:
(# of customer) - (first3lettersofFirstName)-
(last3#ofSSN)

```
private HashMap<String, Account> customerDatabase;
private HashSet<String> uniqueSSN;



//Default constructor
public BankingInstitution() {
    customerDatabase = new HashMap<String, Account>();
    uniqueSSN = new HashSet<String>();
}



// Getter
public HashMap<String, Account> getCustomerDatabase(){
    return customerDatabase;
}
```

Official database (HashMap) of the bank, customerID , Account

```
// create new customer object
public void addCustomer(String name, String DOB, String SSN, String password, String type, Scanner keyboard)
{
    if (uniqueSSN.add(SSN)) {
        Account newCustomer = new Account(name, DOB, SSN, password, type, keyboard);
        customerDatabase.put(newCustomer.getCustomerID(), newCustomer);
        System.out.println("Congratulations, you are now a customer of the DDJK firm!");
    }
    else {
        System.out.println("SSN already exists. Cannot add customer.");
        System.exit(0);

    }


}
```

adds customer to HashMap - each account is associated wlth their key value, customer id

Collections

HashMap (continued)

```
Console
BankingInstitution [Java Application] /Library/Java/JavaVirtualMachines/jdk-
4. Make a withdrawl
5. View account information
6. Close account
100
Customer Name: Jordan Eccleston
Customer ID: 3-JOR-333
Date of Birth: 04/16/2003
Social Security Number: 333333333
Password: J0rdAniscool!
Your  savings account number is: 766471111
Your current balance is: $3000.0
Your checkings account number is: 28363717
Your current balance is: $30.0
Your card pin is:1234
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Customer Name: Donovan Hall
Customer ID: 2-DON-222
Date of Birth: 06/01/2003
Social Security Number: 222222222
Password: Don0v@niscool!
You have no savings account
Your checkings account number is: 887280441
Your current balance is: $1000.0
Your card pin is:1234
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Customer Name: Kirby Smart
Customer ID: 0-KIR-789
Date of Birth: 12/15/1975
Social Security Number: 123456789
Password: Georgia>Bama99
Your  savings account number is: 29124469
Your current balance is: $500000.0
Your checkings account number is: 816345354
Your current balance is: $1800.0
Your card pin is:6507
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Customer Name: Kush Patel
Customer ID: 4-KUS-333
Date of Birth: 04/01/2003
Social Security Number: 123333333
Password: Upward3ound!
Your  savings account number is: 39578060
Your current balance is: $2000.0
You have no checkings account
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Customer Name: Daniel Zavala-Palafox
Customer ID: 1-DAN-111
Date of Birth: 05/20/2003
Social Security Number: 111111111
Password: Upward3ound!
Your  savings account number is: 477945895
Your current balance is: $9.9999999E7
You have no checkings account
```

results of HashMap + other collections - resilient database!

HashSet

```
    System.out.println("Please input your social security number");
    String SSN = keyboard.next();

    if (!uniqueSSN.add(SSN)) {
        System.out.println("SSN already exists. Cannot add customer.");
        System.exit(0);
    }
```

```
// create new customer object
public void addCustomer(String name, String DOB, String SSN, String password, String type, Scanner keyboard)
{
    if (uniqueSSN.add(SSN)) {
        Account newCustomer = new Account(name, DOB, SSN, password, type, keyboard);
        customerDatabase.put(newCustomer.getCustomerID(), newCustomer);
        System.out.println("Congratulations, you are now a customer of the DDJK firm!");
    }
    else {
        System.out.println("SSN already exists. Cannot add customer.");
        System.exit(0);

    }


}
```

creates account object IF SSN does NOT exist in HashSet

```
Please input your first and last name
Jordan Eccleston
Please input your date of birth (MMDDYYYY)
04162003
Please input your social security number
123456789
Please make a password. Your password must sati
Atljulio1125$
```

```
Please input your first and last name
Mark Cuban
Please input your date of birth (MMDDYYYY)
04151977
Please input your social security number
123456789
SSN already exists. Cannot add customer.
```

THANK YOU!

We appreciate your time and attention!