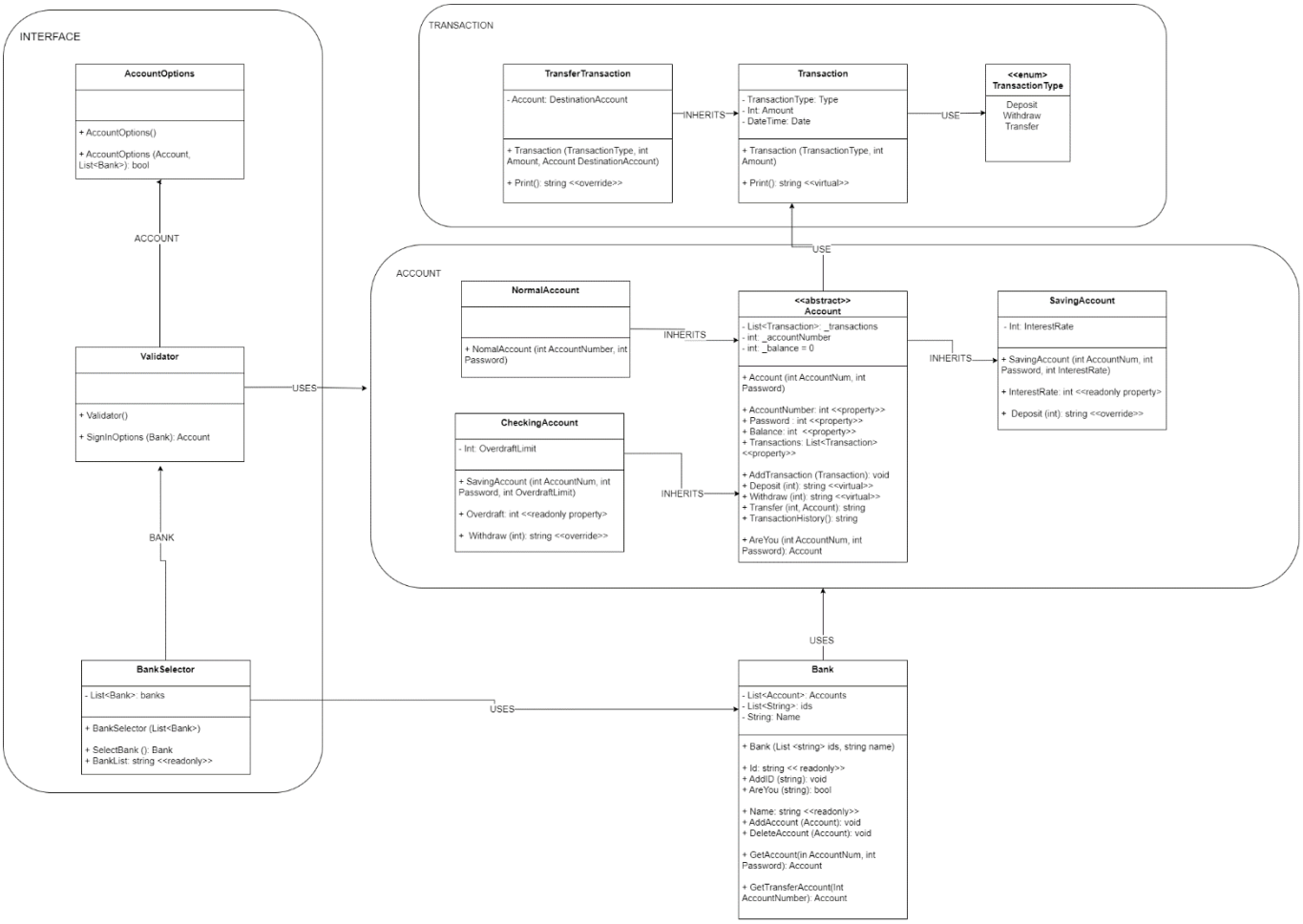
Code Printout And UML: ATM Project

UML



TransactionType

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace ATM
{
    public enum TransactionType
    {
        Deposit,
        Withdraw,
        Transfer
    }
}
```

TransferTransaction

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
```

```
namespace ATM
{
    public class TransferTransaction : Transaction
    {
        public Account DestinationAccount {  get; private set; }

        public TransferTransaction(TransactionType type, int amount, Account
destinationAccount)
            : base(type, amount)
        {
            this.DestinationAccount = destinationAccount;
        }

        public override string Print()
        {
            return $"{Type} of ${Amount} to {DestinationAccount.AccountNumber}
on {Date}";
        }

    }
}
```

Transaction

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace ATM
{
    public class Transaction
    {
        public TransactionType Type { get; private set; }
        public int Amount { get; private set; }
        public DateTime Date { get; private set; }

        public Transaction(TransactionType type, int amount)
        {
            Type = type;
            Amount = amount;
            Date = DateTime.Now;
        }

        public virtual string Print()
        {
            return $"{Type} of ${Amount} on {Date}";
        }
    }
}
```

CheckingAccount

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace ATM
{
    public class CheckingAccount : Account
```

```
    {
        public int OverdraftLimit { get; private set; }

        public CheckingAccount(int accountNumber, int password, int
overdraftLimit)
            : base(accountNumber, password)
        {
            OverdraftLimit = overdraftLimit;
        }


        public override string Withdraw(int amount)
        {
            if (amount <= Balance)
            {
                base.Withdraw(amount);
                Transaction transaction = new
Transaction(TransactionType.Withdraw, amount);
                AddTransaction(transaction);
                return $"You have withdrawn ${amount}. Your new balance is
${Balance}";
            }
            else if (amount <= Balance + OverdraftLimit)
            {
                int overdraft = amount - Balance;
                OverdraftLimit -= overdraft;
                base.Withdraw(Balance);
                Transaction transaction = new
Transaction(TransactionType.Withdraw, amount);
                AddTransaction(transaction);
                return $"You have withdrawn ${amount}. Your new balance is
${Balance}. You have used ${overdraft} of your overdraft limit. Your Overdraft
limit is ${OverdraftLimit} left.";
            }
            else
            {
                return null;
            }
        }


    }
}
```

SavingAccount

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace ATM
{
    public class SavingsAccount : Account
    {
        private int _interestRate;
        public int InterestRate
        {
            get { return _interestRate; }
            private set
            {
                if (value == 3 || value == 5)
                {
                    _interestRate = value;
                }
```

```
                else
                {
                    throw new ArgumentException("Interest rate must be either 3
or 5.");
                }
            }
        }

        public SavingsAccount(int accountNumber, int password, int
interestRate)
            : base(accountNumber, password)
        {
            InterestRate = interestRate;
        }

        public override string Deposit(int amount)
        {
            double extra = amount * (InterestRate / 100.0);
            Console.WriteLine(extra);
            int total = amount + (int)extra;
            base.Deposit(total);
            Transaction transaction = new Transaction(TransactionType.Deposit,
amount);
            AddTransaction(transaction);
            return ($"Interest applied at {InterestRate}% for deposit of
${amount}. Now your balance is ${Balance}");
        }

    }
}
```

NormalAccount

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace ATM
{
    public class NormalAccount : Account
    {
        public NormalAccount(int accountNumber, int password)
            : base(accountNumber, password)
        {
        }

        public override string Withdraw(int amount)
        {
            if (amount > Balance)
            {
                return null;
            }
            else
            {
                Balance -= amount;
                Transaction transaction = new
Transaction(TransactionType.Withdraw, amount);
                AddTransaction(transaction);
                return $"You have withdrawn ${amount}. Your new balance is
${Balance}";
            }
        }

        public override string Deposit(int amount)
```

```
        {
            Balance += amount;
            Transaction transaction = new Transaction(TransactionType.Deposit,
amount);
            AddTransaction(transaction);
            return $"You have deposited ${amount}. Your new balance is
${Balance}";
        }
    }
}
```

Account

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace ATM
{
    public abstract class Account
    {
        List<Transaction> _transactions = new List<Transaction>();
        int _accountNumber;
        int _password;
        int _balance = 0;

        public Account(int accountNumber, int password)
        {
            _accountNumber = accountNumber;
            _password = password;
        }

        public int AccountNumber
        {
            get { return _accountNumber; }
            set { _accountNumber = value; }
        }

        public int Password
        {
            get { return _password; }
            set { _password = value; }
        }

        public int Balance
        {
            get { return _balance; }
            set { _balance = value; }
        }

        public List<Transaction> Transactions
        {
            get { return _transactions; }
            set { _transactions = value; }
        }

        public void AddTransaction(Transaction transaction)
        {
            _transactions.Add(transaction);
        }

        public virtual string Deposit(int amount)
        {
            Balance += amount;
```

```
            Transaction transaction = new Transaction(TransactionType.Deposit,
amount);
            AddTransaction(transaction);
            return $"You have deposited ${amount}. Your new balance is
${Balance}";
        }

        public virtual string Withdraw(int amount)
        {
            if (amount > Balance)
            {
                return null;
            }
            else
            {
                Balance -= amount;
                Transaction transaction = new
Transaction(TransactionType.Withdraw, amount);
                AddTransaction(transaction);
                return $"You have withdrawn ${amount}. Your new balance is
${Balance}";
            }
        }

        public string Transfer(int amount, Account destinationAccount)
        {
            if (amount > Balance)
            {
                return null;
            }
            else
            {
                Balance -= amount;
                destinationAccount.Deposit(amount);
                TransferTransaction transaction = new
TransferTransaction(TransactionType.Transfer, amount, destinationAccount);
                AddTransaction(transaction);
                return $"You have transferred ${amount} to account number
{destinationAccount.AccountNumber}. Your new balance is ${Balance}";
            }

        }


        public string TransactionHistory()
        {
            int order = 1;
            string history = "";
            foreach (Transaction transaction in _transactions)
            {
                history += order + ". " + transaction.Print() + "\n";
                order++;
            }
            return history;
        }

        public Account? AreYou(int InputAccountNum, int InputPassword)
        {
            if (InputAccountNum == _accountNumber && InputPassword ==
_password)
            {
                return this;
            }
            else
            {
                return null;
            }
        }
```

```
    }
}
```

Utility

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Security;
using System.Text;
using System.Threading;
using System.Threading.Tasks;

namespace ATM
{
    public class Utility
    {
        public static string GetSecretInput(string prompt)
        {
            Console.Write(prompt + "\n");
            StringBuilder input = new StringBuilder();

            while (true)
            {
                ConsoleKeyInfo inputKey = Console.ReadKey(true);

                if (inputKey.Key == ConsoleKey.Enter)
                {
                    break;
                }

                if (inputKey.Key == ConsoleKey.Backspace)
                {
                    if (input.Length > 0)
                    {
                        input.Remove(input.Length - 1, 1);
                        // Move the cursor back, overwrite the last asterisk
with a space, and move the cursor back again
                        Console.Write("\b \b");
                    }
                }
                else
                {
                    input.Append(inputKey.KeyChar);
                    Console.Write("*");
                }
            }
            Console.WriteLine(); // Move to the next line after input is done
            return input.ToString();
        }

        public static void PrintColoredMessage(string message, ConsoleColor
color)
        {
            Console.ForegroundColor = color;
            Console.WriteLine(message);
            Console.ResetColor();
        }

        public static void PressEnterToContinue()
        {
            Console.WriteLine("\nPress Enter to continue...");
            while (Console.ReadKey(true).Key != ConsoleKey.Enter)
            {
            }
```

```
        }

        public static void SleepWithDots(int milliseconds)
        {
            int dotsCount = 0;
            while (milliseconds > 0)
            {
                Console.Write(".");
                Thread.Sleep(500); // Sleep for 500 milliseconds for each dot
                milliseconds -= 500;
                dotsCount++;
                if (dotsCount % 10 == 0) // Print newline after every 10 dots
                {
                    Console.WriteLine();
                }
            }
            Console.WriteLine(); // Ensure newline after sleeping
        }
    }
}
```

AccountOptions

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading;
using System.Threading.Tasks;

namespace ATM
{
    public class AccountOptions
    {
        public AccountOptions()
        {
        }

        public bool AccountOptionsMenu(Account account, List<Bank>
banksDatabase)
        {
            bool isSignedOut = false;
            Console.Clear();
            Console.WriteLine("Please select an option: \n1. Deposit \n2.
Withdraw \n3. Transfer \n4. Check balance \n5. Check transactions history\n6.
Sign out\n\nYour Option:");
            string Input;
            int InputOption;

            while (true)
            {
                Input = Console.ReadLine().Trim();

                if (int.TryParse(Input, out InputOption) && InputOption > 0 &&
InputOption < 7)
                {
                    break;
                }
                else
                {
                    Utility.PrintColoredMessage("Invalid input. Please enter a
valid option.", ConsoleColor.Red);
                    Utility.SleepWithDots(1000);
                    return AccountOptionsMenu(account, banksDatabase);
                }
            }
```

```
            switch (InputOption)
            {
                case 1:

                    Console.WriteLine("Please enter the amount you would like
to deposit: ");

                    int DepositAmount;
                    while (true)
                    {
                        string input = Console.ReadLine().Trim();
                        if (int.TryParse(input, out DepositAmount) &&
DepositAmount > 0)
                        {
                            break;
                        }
                        else
                        {

                            Utility.PrintColoredMessage("Invalid input. Please
enter a valid positive number.", ConsoleColor.Red);
                            Utility.SleepWithDots(1000);
                            return AccountOptionsMenu(account, banksDatabase);
                        }
                    }


                    Console.Clear();
                    Console.WriteLine(account.Deposit(DepositAmount));
                    Utility.PressEnterToContinue();
                    return AccountOptionsMenu(account, banksDatabase);

                case 2:

                    Console.WriteLine("Please enter the amount you would like
to withdraw: ");

                    int WithdrawAmount;
                    while (true)
                    {
                        string input = Console.ReadLine().Trim();
                        if (int.TryParse(input, out WithdrawAmount) &&
WithdrawAmount > 0)
                        {
                            break;
                        }
                        else
                        {
                            Utility.PrintColoredMessage("Invalid input. Please
enter a valid positive number.", ConsoleColor.Red);
                            Utility.SleepWithDots(1000);
                            return AccountOptionsMenu(account, banksDatabase);
                        }
                    }

                    string withdrawalResult = account.Withdraw(WithdrawAmount);
                    if (withdrawalResult == null)
                    {
                        Utility.PrintColoredMessage("Insufficient funds. Please
try again.", ConsoleColor.Red);
                        Utility.SleepWithDots(1000);
                        return AccountOptionsMenu(account, banksDatabase);
                    }
                    else
                    {
                        Console.Clear();
                        Console.WriteLine(withdrawalResult);
                        Utility.PressEnterToContinue();
```

```
                        return AccountOptionsMenu(account, banksDatabase);
                    }

                case 3:

                    Console.WriteLine("Please enter the amount you would like
to transfer: ");

                    int TransferAmount;
                    while (true)
                    {
                        string input = Console.ReadLine().Trim();
                        if (int.TryParse(input, out TransferAmount) &&
TransferAmount > 0)
                        {
                            break;
                        }
                        else
                        {
                            Utility.PrintColoredMessage("Invalid input. Please
enter a valid positive number.", ConsoleColor.Red);
                            Utility.SleepWithDots(1000);
                            return AccountOptionsMenu(account, banksDatabase);
                        }
                    }

                    BankSelector bankSelector = new
BankSelector(banksDatabase);
                    Bank selectedBank = null;

                    while (selectedBank == null)
                    {
                        selectedBank = bankSelector.SelectBank();
                        if (selectedBank == null)
                        {
                            Utility.PrintColoredMessage("Please select a valid
bank by either abbreviated name or number", ConsoleColor.Red);
                        }

                    }

                    Console.WriteLine("\nPlease enter the account number you
would like to transfer to (6 digits): ");

                    int DestinationAccountNum;
                    while (true)
                    {
                        string input = Console.ReadLine().Trim();
                        if (int.TryParse(input, out DestinationAccountNum) &&
input.Length == 6)
                        {
                            break;
                        }
                        else
                        {
                            Utility.PrintColoredMessage("Invalid input. Please
enter a valid 6-digit account number.", ConsoleColor.Red);
                            Utility.SleepWithDots(1000);
                            return AccountOptionsMenu(account, banksDatabase);
                        }
                    }

                    Account? destinationAccount =
selectedBank.GetTransferAccount(DestinationAccountNum);
                    if (destinationAccount == null)
                    {

                        Utility.PrintColoredMessage("Account not found. Please
try again.", ConsoleColor.Red);
```

```
                        Utility.SleepWithDots(1000);
                        return AccountOptionsMenu(account, banksDatabase);
                    }
                    else if (TransferAmount > account.Balance)
                    {
                        Utility.PrintColoredMessage("Insufficient funds. Please
try again.", ConsoleColor.Red);
                        Utility.SleepWithDots(1000);
                        return AccountOptionsMenu(account, banksDatabase);
                    }
                    else
                    {
                        Console.Clear();
                        Console.WriteLine(account.Transfer(TransferAmount,
destinationAccount));
                        Utility.PressEnterToContinue();
                        return AccountOptionsMenu(account, banksDatabase);
                    }



                case 4:
                    Console.Clear();
                    Console.WriteLine("Your current balance is $" +
account.Balance);
                    Utility.PressEnterToContinue();
                    return AccountOptionsMenu(account, banksDatabase);


                case 5:
                    Console.Clear();
                    Console.WriteLine("Transaction History: ");
                    Console.WriteLine(account.TransactionHistory());
                    Utility.PressEnterToContinue();
                    return AccountOptionsMenu(account, banksDatabase);


                case 6:
                    Console.Clear();
                    Console.WriteLine("You have signed out successfully.");
                    Utility.PressEnterToContinue();
                    isSignedOut = true;
                    return isSignedOut;


                default:

                    Console.WriteLine("Invalid option. Please try again.");
                    Utility.SleepWithDots(1000);
                    return AccountOptionsMenu(account, banksDatabase);
            }
        }
    }
}
```

Validator

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading;
using System.Threading.Tasks;
```

```
namespace ATM
{
  public class Validator
  {

    public Validator()
    {
    }

    public Account? SignInOptions(Bank bank)
    {
      Console.Clear();
      Console.WriteLine("Please select an option: \n1. Sign in \n2. Create a normal
account\n3. Create a saving account\n4. Create a checking account\n5.
Back\n\nYour Option:");

      int InputOption;
      while (true)
      {
        string input = Console.ReadLine().Trim();

        // Check if the input is an integer and within the range
        if (int.TryParse(input, out InputOption) && InputOption >= 1 && InputOption <=
5)
        {
          break; // Exit the loop if the input is valid
        }
        else
        {
          Utility.PrintColoredMessage("Invalid option. Please try again.",
ConsoleColor.Red);
          Utility.SleepWithDots(1000); // Pause for 1 seconds to allow the user to see
the error message
          return SignInOptions(bank);
        }
      }

      if (InputOption == 5)
      {
        return null;
      }

      switch (InputOption)
      {
        case 1:
          Console.WriteLine("Please enter your account number (6 digits): ");
          string inputAccountNumStr = Console.ReadLine().Trim();
          if (!int.TryParse(inputAccountNumStr, out int inputAccountNum) ||
inputAccountNumStr.Length != 6)
          {
            Utility.PrintColoredMessage("Invalid account number. Please try again.",
ConsoleColor.Red);
```

```
                Utility.SleepWithDots(1000); // Pause for 1 seconds to allow the user to
see the error message
                return SignInOptions(bank);
            }

            string inputPasswordStr = Utility.GetSecretInput("Please enter your
password (4 digits): ");
            if (!int.TryParse(inputPasswordStr, out int inputPassword) ||
inputPasswordStr.Length != 4)
            {
                Utility.PrintColoredMessage("Invalid password. Please try again.",
ConsoleColor.Red);
                Utility.SleepWithDots(1000); // Pause for 1 seconds to allow the user to
see the error message
                return SignInOptions(bank);
            }

            if (bank.GetAccount(inputAccountNum, inputPassword) == null)
            {
                Utility.PrintColoredMessage("Account not found. Please try again.",
ConsoleColor.Red);
                Utility.SleepWithDots(1000); // Pause for 1 seconds to allow the user to
see the error message
                return SignInOptions(bank);
            }
            else
            {
                Console.Clear();
                Utility.PrintColoredMessage("You have logged in successfully. Welcome
back " + bank.GetAccount(inputAccountNum, inputPassword).AccountNumber,
ConsoleColor.Green);
                Utility.PressEnterToContinue();
                return bank.GetAccount(inputAccountNum, inputPassword);
            }

            break;

        case 2:
            int NewAccountNum;
            Console.WriteLine("\nPlease enter your new account number (6 digits): ");
            string accountNumInput = Console.ReadLine();
            if (!int.TryParse(accountNumInput, out NewAccountNum) ||
accountNumInput.Length != 6)
            {
                Utility.PrintColoredMessage("Invalid account number format. Please enter
a 6-digit number.", ConsoleColor.Red);
                Utility.SleepWithDots(1000); // Pause for 1 seconds to allow the user to
see the error message
                return SignInOptions(bank);
            }

            int NewPassword;
```

```
            string passwordInput = Utility.GetSecretInput("Please enter your preferred
password (4 digits): ");
            if (!int.TryParse(passwordInput, out NewPassword) || passwordInput.Length
!= 4)
            {
                Utility.PrintColoredMessage("Invalid password format. Please enter a 4-
digit number.", ConsoleColor.Red);
                Utility.SleepWithDots(1000); // Pause for 1 seconds to allow the user to
see the error message
                return SignInOptions(bank);
            }

            Account newAccount = new NormalAccount(NewAccountNum,
NewPassword);
            bank.AddAccount(newAccount);

            Console.Clear();
            Utility.PrintColoredMessage("Account created successfully!",
ConsoleColor.Green);
            Utility.PressEnterToContinue();
            return SignInOptions(bank);


            break;


        case 3:
            int NewAccountNum2;
            Console.WriteLine("\nPlease enter your new account number (6 digits): ");
            string accountNumInput2 = Console.ReadLine();
            if (!int.TryParse(accountNumInput2, out NewAccountNum2) ||
accountNumInput2.Length != 6)
            {
                Utility.PrintColoredMessage("Invalid account number format. Please enter
a 6-digit number.", ConsoleColor.Red);
                Utility.SleepWithDots(1000); // Pause for 1 seconds to allow the user to
see the error message
                return SignInOptions(bank);
            }

            int NewPassword2;
            string passwordInput2 = Utility.GetSecretInput("Please enter your preferred
password (4 digits): ");
            if (!int.TryParse(passwordInput2, out NewPassword2) ||
passwordInput2.Length != 4)
            {
                Utility.PrintColoredMessage("Invalid password format. Please enter a 4-
digit number.", ConsoleColor.Red);
                Utility.SleepWithDots(1000); // Pause for 1 seconds to allow the user to
see the error message
                return SignInOptions(bank);
            }
```

```
            int InputInterestRate;
            Console.WriteLine("Please select your desired interest rate: \n1. 3%\n2.
5%");
            string interestRateInput = Console.ReadLine().Trim();
            if (!int.TryParse(interestRateInput, out InputInterestRate) ||
(InputInterestRate != 1 && InputInterestRate != 2))
            {
              Utility.PrintColoredMessage("Invalid interest rate choice. Please enter
either 1 or 2.", ConsoleColor.Red);
              Utility.SleepWithDots(1000); // Pause for 1 seconds to allow the user to
see the error message
              return SignInOptions(bank);
            }

            if (InputInterestRate == 1)
            {
              SavingsAccount Account = new SavingsAccount(NewAccountNum2,
NewPassword2, 3);
              bank.AddAccount(Account);
            }
            else
            {
              SavingsAccount Account = new SavingsAccount(NewAccountNum2,
NewPassword2, 5);
              bank.AddAccount(Account);
            }

            Utility.PrintColoredMessage("Account created successfully!",
ConsoleColor.Green);
            Utility.PressEnterToContinue();
            return SignInOptions(bank);


            break;

          case 4:
            int NewAccountNum3;
            Console.WriteLine("\nPlease enter your new account number (6 digits): ");
            string accountNumInput3 = Console.ReadLine();
            if (!int.TryParse(accountNumInput3, out NewAccountNum3) ||
accountNumInput3.Length != 6)
            {
              Utility.PrintColoredMessage("Invalid account number format. Please enter
a 6-digit number.", ConsoleColor.Red);
              Utility.SleepWithDots(1000); // Pause for 1 seconds to allow the user to
see the error message
              return SignInOptions(bank);
            }

            int NewPassword3;
```

```
            string passwordInput3 = Utility.GetSecretInput("Please enter your preferred
password (4 digits): ");
            if (!int.TryParse(passwordInput3, out NewPassword3) ||
passwordInput3.Length != 4)
            {
              Utility.PrintColoredMessage("Invalid password format. Please enter a 4-
digit number.", ConsoleColor.Red);
              Utility.SleepWithDots(1000); // Pause for 1 seconds to allow the user to
see the error message
              return SignInOptions(bank);
            }

            int InputOverdraftLimit;
            Console.WriteLine("Please select your desired overdraft limit: \n1. $100\n2.
$200");
            string overdraftLimitInput = Console.ReadLine().Trim();
            if (!int.TryParse(overdraftLimitInput, out InputOverdraftLimit) ||
(InputOverdraftLimit != 1 && InputOverdraftLimit != 2))
            {
              Utility.PrintColoredMessage("Invalid overdraft limit choice. Please enter
either 1 or 2.", ConsoleColor.Red);
              Utility.SleepWithDots(1000); // Pause for 1 seconds to allow the user to
see the error message
              return SignInOptions(bank);
            }

            CheckingAccount accountToCreate;
            if (InputOverdraftLimit == 1)
            {
              accountToCreate = new CheckingAccount(NewAccountNum3,
NewPassword3, 100);
            }
            else
            {
              accountToCreate = new CheckingAccount(NewAccountNum3,
NewPassword3, 200);
            }
            bank.AddAccount(accountToCreate);
            Utility.PrintColoredMessage("Account created successfully!",
ConsoleColor.Green);
            Utility.PressEnterToContinue();
            return SignInOptions(bank);
            break;


          default:

            Utility.PrintColoredMessage("Invalid option. Please try again.",
ConsoleColor.Red);
            return SignInOptions(bank);
        }
    }
```

```
  }
}
```

BankSelector

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace ATM
{
    public class BankSelector
    {
        internal List<Bank> _banks = new List<Bank>();
        public BankSelector(List<Bank> banks)
        {
            _banks = banks;
            foreach (Bank bank in _banks)
            {
                int num = _banks.IndexOf(bank) + 1;
                bank.AddId(num.ToString());
            }
        }

        public Bank? SelectBank()
        {
            Console.WriteLine("Please select a bank: \n" + BankList + "\nYour
Option:");
            string InputBank = Console.ReadLine().ToLower().Trim();

            foreach (Bank bank in _banks)
            {
                if (bank.AreYou(InputBank))
                {
                    return bank;
                }
            }
            return null;
        }

        public string BankList
        {
            get
            {
                int i = 1;
                string bankList = "";
                foreach (Bank bank in _banks)
                {
                    bankList += i + ". " + bank.Name + " (" + bank.Id + ")\n";
                    i++;
                }
                return bankList;
            }
        }
    }
}
```

Bank

```
using System;
using System.Collections.Generic;
using System.Linq;
```

```
using System.Text;
using System.Threading.Tasks;

namespace ATM
{
    public class Bank
    {
        private List<Account> _accounts = new List<Account>();
        private List<string> _ids = new List<string>();
        private string _name;
        public Bank(List<string> ids, string name)
        {
            foreach (string id in ids)
            {
                _ids.Add(id.ToLower());
                _name = name;
            }
        }

        public string Id
        {
            get { return _ids[0].ToUpper(); }
        }

        public void AddId(string id)
        {
            _ids.Add(id.ToLower());
        }

        public bool AreYou(string id)
        {
            if (_ids.Contains(id.ToLower()))
            {
                return true;
            }
            else
            {
                return false;
            }
        }

        public string Name
        {
            get { return _name; }
        }

        public void AddAccount(Account account)
        {
            _accounts.Add(account);
        }

        public void DeleteAccount(Account account)
        {
            _accounts.Remove(account);
        }

        public Account? GetAccount(int accountNumber, int password)
        {
            foreach (Account account in _accounts)
            {
                if (account.AccountNumber == accountNumber && account.Password
== password)
                {
                    return account;
                }
            }
            return null;
        }
```

```
        public Account? GetTransferAccount(int accountNumber)
        {
            foreach (Account account in _accounts)
            {
                if (account.AccountNumber == accountNumber)
                {
                    return account;
                }
            }
            return null;
        }
    }
}
```

Program

```
using System;
using System.Collections.Generic;
using System.Diagnostics.Eventing.Reader;
using System.Linq;
using System.Net;
using System.Security.Cryptography.X509Certificates;
using System.Text;
using System.Threading;
using System.Threading.Tasks;

namespace ATM
{
    public class Program
    {
        static void Main(string[] args)
        {
            //Set up banks and accounts
            Bank cba = new Bank(new List<string> {"CBA"}, "Commonwealth");
            Account account1 = new NormalAccount(123456, 1234);
            SavingsAccount account3 = new SavingsAccount(123457, 1234, 3);
            cba.AddAccount(account1);
            cba.AddAccount(account3);
            Bank westpac = new Bank(new List<string> {"WBC"}, "Westpac");
            CheckingAccount account2 = new CheckingAccount(654321, 4321, 100);
            westpac.AddAccount(account2);
            Bank nab = new Bank(new List<string> { "NAB" }, "National Australia
Bank");
            Bank anz = new Bank(new List<string> { "ANZ" }, "Australia and New
Zealand Banking Group");
            List<Bank> banksDatabase = new List<Bank> { cba, westpac, nab, anz
};

            // Set the console title and text color
            Console.Title = "My ATM Application";
            Console.ForegroundColor = ConsoleColor.White;

            // Display the welcome message
            Utility.PrintColoredMessage("\n\n---------------------------------
-Welcome to the ATM----------------------------------\n", ConsoleColor.Green);

            //Sign in or create an account to sign in
            BankSelector bankSelector = new BankSelector(banksDatabase);
            Bank selectedBank = null;

            while (selectedBank == null)
            {
                selectedBank = bankSelector.SelectBank();
                if (selectedBank == null)
                {
```

```
                    Utility.PrintColoredMessage("Please select a valid bank by
either abbreviated name or number", ConsoleColor.Red);
                }

            }

            Console.Clear();
            Utility.PrintColoredMessage("Welcome to " + selectedBank.Name +
".", ConsoleColor.Green);
            Utility.SleepWithDots(3000);

            Validator validator = new Validator();
            Account signedInAccount = validator.SignInOptions(selectedBank);

            //This option is for when user selects "back" in the sign in
options.
            while (signedInAccount == null)
            {
                Console.Clear();
                selectedBank = bankSelector.SelectBank();
                signedInAccount = validator.SignInOptions(selectedBank);
            }


            //Display account options
            AccountOptions accountOptions = new AccountOptions();
            bool isSignedOut =
accountOptions.AccountOptionsMenu(signedInAccount, banksDatabase);

            //For the sign out option
            while (isSignedOut)
            {
                Console.Clear();
               signedInAccount = validator.SignInOptions(selectedBank);
               while (signedInAccount == null)
                {
                    Console.Clear();
                    selectedBank = bankSelector.SelectBank();
                    signedInAccount = validator.SignInOptions(selectedBank);
                }
               isSignedOut = accountOptions.AccountOptionsMenu(signedInAccount,
banksDatabase);
            }




        }
    }
}
```