UNIVERSIDAD EUROPEA DEL ATLÁNTICO

GRADO EN

Ingeniería Informática

Programación II

Documentación Proyecto Final

(Cajero Automático)

Trabajo realizado por

Madeline Trejos (developer)

Sara Patiño (reviewer)

Brandon Samayoa (developer)

Profesor
Elder Bol

SANTANDER – 9, Junio, 2023

Índice:

Información general

Idea principal del proyecto:

El proyecto se centra en el desarrollo de un cajero automático que proporciona una amplia gama de funciones y servicios a los titulares de cuentas bancarias. La principal característica del cajero automático es su capacidad para llevar a cabo un proceso de autenticación seguro, garantizando que solo los usuarios autorizados tengan acceso a las funcionalidades.

Para garantizar la seguridad, el cajero automático requerirá que los usuarios ingresen su identificación personal, que puede ser un usuario y contraseña o un número PIN (Personal Identification Number). Esta medida de seguridad ayuda a prevenir el acceso no autorizado a la cuenta bancaria y garantiza que solo el titular de la cuenta pueda llevar a cabo transacciones.

En resumen, el proyecto se enfoca en desarrollar un cajero automático completo y seguro que brinde a los usuarios la capacidad de realizar operaciones financieras como depósitos, transferencias, pagos y consultas. Esto no solo proporciona comodidad y accesibilidad a los titulares de cuentas bancarias, sino que también promueve la eficiencia en la gestión de transacciones y la seguridad de los fondos.

Implementación:

El proyecto implementó programación orientada a objetos (POO) para lograr un diseño eficiente y extensible. Se utilizaron clases abstractas para definir propiedades y métodos comunes, como "Transacción", con clases concretas heredando de ellas. Se emplearon métodos estáticos para funcionalidades globales, como autenticación. Las interfaces definieron métodos que deben ser implementados, permitiendo flexibilidad en el comportamiento. La herencia posibilitó la reutilización de código y organización jerárquica, mientras que el polimorfismo facilitó la manipulación uniforme de objetos de diferentes clases.

¿Qué hace?

Esta iniciativa se fundamenta en un dispositivo de autoservicio financiero que ofrece múltiples prestaciones. Podrás seleccionar tu entidad bancaria, llevar a cabo transacciones, abonar facturas pendientes u otros pagos específicos, verificar el saldo y revisar tus gastos mensuales. Además, el cajero automático brinda la opción de realizar depósitos, transferencias, retiros y pagos electrónicos de facturas.

¿Para que se realiza?

Se creó este dispensador automático con el propósito de acelerar las transacciones, los retiros, los pagos y las demás operaciones disponibles en un cajero automático, con el fin de alcanzar una mayor eficiencia y rapidez para el usuario al llevar a cabo múltiples operaciones en un solo dispositivo, superando así la velocidad de otros cajeros automáticos.

¿Qué necesidad será resuelta?

Satisface múltiples requerimientos, destacando principalmente la posibilidad de realizar retiros y/o depósitos de efectivo, efectuar transferencias y pagar facturas, todo ello sin requerir la presencia física del cliente en la institución bancaria. Esta característica también agiliza la movilidad de las personas, ya que se puede llevar a cabo desde cualquier ordenador sin necesidad de desplazarse del lugar.

Funcionalidades

1. **Creación de usuarios:** Se pueden crear usuarios en cada banco al momento de crear una nueva cuenta en dicho banco.

```
 Bienvenido/a al banco BBVA

Seleccione una opción:
1) Abrir una cuenta nueva
2) Ingresar al cajero
3) Salir
1
Ingrese su dni: Y8260607X
Ingrese su email: madeline.trejos@alumnos.uneatlantico.es
Ingrese su Nombre: Madeline
Ingrese sus Apellidos: Trejos
Ingrese su pin personal: 1431
Ingrese un alias para su cuenta: M.trejos
```

2. **Autentificación de Usuario:** Se autentifica a un usuario de acuerdo a su dni o id único de usuario generado al momento de su creación y un pin que se encuentra encriptado.

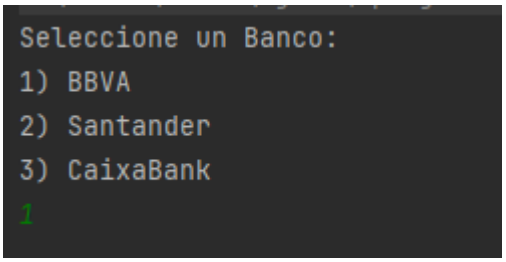
```
Ingrese su pin personal: 1431
Ingrese un alias para su cuenta: M.trejos

Su ID de usuario es: 379030


Resumen de cuentas de: Madeline Trejos
1) 9899670302 : $0.00 : M.trejos

 Madeline Trejos, que quiere hacer
   1) Ver Historial de Transacciones
   2) Añadir una nueva cuenta bancaria
   3) Retirar Dinero
   4) Depositar
   5) Realizar Transferencia
   6) Pagar servicios
   7) Salir
```

3. **Implementación de una interfaz para Varios bancos:** Se pueden tener diferentes bancos para que el usuario elija a cual quiere ingresar dependiendo de en cúal tiene cuenta. También se permite que el usuario ingrese a un banco para crear una nueva cuenta.



4. **Retiros de efectivo:** se pueden realizar retiros de cada una de las cuentas que tiene el usuario en un banco dependiendo del monto que tiene disponible. En ningún caso se permite sobregirar la cuenta.

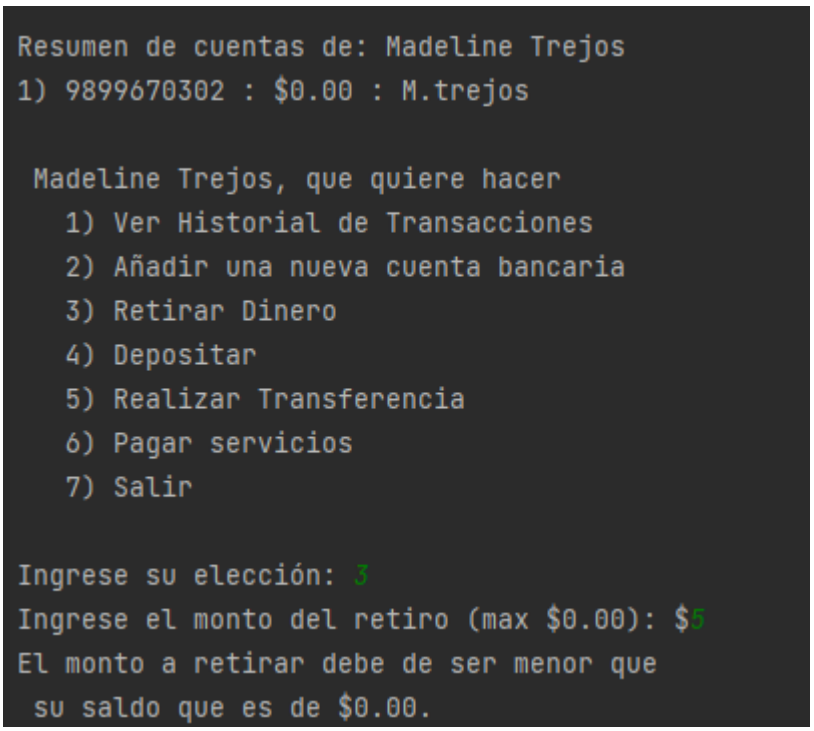


Diagrama de clases

Código UML

```
class classes.ATMManagement implements interfaces.IManagement{
  ~ ArrayList<Entity> entities
  ~ ArrayList<Account> accounts
  ~ ArrayList<User> users
  ~ ArrayList<Transaction> transactions

  + ATMManagement()
  - void loadData()
}

class classes.User implements interfaces.IUser{
  - String uuid
  - String dni
  - String email
  - String firstName
  - String lastName
  - byte pinHash[]
  - ArrayList<Account> accounts

  + User(String dni, String email, String firstName, String lastName, String pin, Entity
theBank)
}

class classes.Account{
  - String uuid
  - String name
  - double avalibleMoney
  - User holder
  - Entity entity
  - ArrayList<Transaction> transactions

  + Account (String name, User holder, Entity theBank)
  + String getUUID()
  + double getBalance()
  + double getAvalibleMoney()
  + Entity getEntity()
  + void setAvalibleMoney(double newAvalibleMoney)
  + String getSummaryLine()
  + void printTransHistory()
  + void addTransaction (Transaction newTrans)
}
```

class classes.Entity{
  - String name
  - ArrayList<User> users
  - ArrayList<Account> accounts
  - ArrayList<Transaction> transactions

  + Entity (String name)
  + String createUserUUID()
  + String createAccountUUID()
  + String createTransactionUUID()
  + void addAccount(Account onAcct)
  + void addUser(User newUser)
  + void addTransaction(Transaction newTransaction)
  + User userLogin(String userID, String pin)
  + String getName()
  + ArrayList<User> getUsers()
  + ArrayList<Account> getAccounts()
  + ArrayList<Transaction> getTransactions()
}

abstract class classes.transactions.Transaction implements interfaces.ITransaction{
   # String uuid
   # ETransactionTypes transactionType
   # double amount
   # Date transactionDate
   # String description
   # Account inAccount

   + Transaction(double amount,Account inAccount, Entity entity, ETransactionTypes transactionType)
   + Transaction (double amount, String description, Account inAccount, Entity entity, ETransactionTypes transactionType)
   + Transaction(String uuid, double amount, Date transactionDate, String description, Account inAccount, ETransactionTypes transactionType)
}

class classes.transactions.Deposit extends classes.transactions.Transaction {
   + Deposit(double amount, Account inAccount, Entity entity, ETransactionTypes transactionType)
}

class classes.transactions.Transfer extends classes.transactions.Transaction {
  + Transfer(double amount, Account inAccount, Account destinationAccount, Entity entity, ETransactionTypes transactionType)
  + Transfer(double amount, String description, Account inAccount, Entity entity, ETransactionTypes transactionType)
}

class classes.transactions.Withdrawal extends classes.transactions.Transaction {
  + Withdrawal(double amount, Account inAccount, Entity entity, ETransactionTypes transactionType)
}

abstract class classes.transactions.servicePayment.ServicePayment extends classes.transactions.Transaction implements interfaces.IServicePayment{
  # double montToPay
  # Date maxDateToPay

  + ServicePayment(double amount,Account inAccount, Entity entity, ETransactionTypes transactionType)
  + void verifyInvoice()
  + void payService()
}

class classes.transactions.servicePayment.PayWater extends classes.transactions.servicePayment.ServicePayment {
  + PayWater(double amount,Account inAccount, Entity entity, ETransactionTypes transactionType)
}

class classes.transactions.servicePayment.PayLight extends classes.transactions.servicePayment.ServicePayment{
  + PayLight (double amount,Account inAccount, Entity entity, ETransactionTypes transactionType)
}

class classes.transactions.servicePayment.PayGas extends classes.transactions.servicePayment.ServicePayment{
  + PayGas (double amount,Account inAccount, Entity entity, ETransactionTypes transactionType)
}

class classes.transactions.servicePayment.PayPhone extends classes.transactions.servicePayment.ServicePayment{
  + PayPhone (double amount,Account inAccount, Entity entity, ETransactionTypes transactionType)
}

class client.Menu{
  + {static} Entity ATMMainView(ATMManagement atmMg, Scanner sc)
  + {static} void adminView(ATMManagement atmMg, Scanner sc)
  + {static} void createEntity(ATMManagement atmMg, Scanner sc)
  + {static} User bankMainView(Entity entity, Scanner sc)
  + {static} User createAccount(Entity entity, Scanner sc)

```
  + {static} User loginView(Entity entity, Scanner sc)
  + {static} void userMenuView(ATMManagement atmMg, User user, Entity entity, Scanner
sc)
  + {static} void addAccount(User user,Entity entity, Scanner sc)
  + {static} void transHistoryView(User user, Scanner sc)
  + {static} void transferView(ATMManagement atmMg,User user, Entity entity, Scanner sc)
  + {static} void personalTransferView(User user, Entity entity, Scanner sc)
  + {static} void interbankTransferView(ATMManagement atmMg,User user, Entity entity,
Scanner sc)
  + {static} void withdrawlView(User user, Entity entity, Scanner sc)
  + {static} void depositView(User user, Entity entity, Scanner sc)
  + {static} void payServices(ATMManagement atmMg,User user, Entity entity, Scanner sc)
}

class common.Files{
  + {static} List<String> loadFile(String filePath)
}
class common.Utils{
  + {static} String[] clearString( String[] tokens, String token, String replacement)
}

enum enums.ETransactionTypes{
  deposit
  withdrawal
  personalTransfer
  interbankTransfer
  servicesPayment
}

interface interfaces.IManagement{
 + void start()
 + void addEntity(Entity e)
 + Account foundAccountInEntitiesForUUID(String UUID)
 + List<Entity> getEntities()
}

interface interfaces.IServicePayment extends interfaces.ITransaction{
  # void verifyInvoice();
  # void payService();
  # boolean createTransaction();
}

interface interfaces.ITransaction{
  # String getUUID();
  # double getAmount();
  # String getTransactionType();
  # String getSummaryLine();
```

```
   # {abstract} boolean createTransaction();
}

interface interfaces.IUser{
  + String getUUID()
  + String getFirstName()
  + int getNumAccounts()
  + double getAcctBalance(int acctIdx)
  + String getAcctUUID(int acctIdx)
  + void addAccount(Account onAcct)
  + Account getAccount(int AccountIndex)
  + String getDNI()
  + boolean validatePin (String userPin)
  + void printAccountsSummary()
  + void printAcctTransHistory(int acctIdx)
}
@enduml
```

Diagrama

https://www.plantuml.com/plantuml/svg/pHXTR-8uy7dv5PRk9LLp_GDtgguhjhT8xQfIwUt3QX-coG1M4nlPfYr2lTz-iJE1SG870I7jl-Bv_XXdOazMWBKYFD5UtOplxq7154jKJiXoNeIL5L8vD1F8AfxHXcCr4E9VSMCCBE

Ubicación del repositorio

https://github.com/sara-patino/awesome-projects-prograii