INSTITUTO POLITECNICO NACIONAL
UNIDAD PROFESIONAL

UNIDAD PROFESIONAL INTERDISCIPLINARIA DE INGENIERIA Y CIENCIAS SOCIALES Y ADMINISTRATIVAS

INGENIERIA EN INFORMATICA

Programación Orientada a Objetos

SECUENCIA: 2NM42

Equipo 07

Porcentaje de Participación

De Santiago Landeros Edwin Uriel

Reveles Ramirez Jesus Emilio

Villegas Nolasco Elide Yolotzin

Herrera Cano Edwin Ruben

Palmieri Mondragron Gerardo

PROFESOR. Oviedo Galdeano Mario

Segundo Departamental: Proyecto Cajero ATM

Fecha de entrega:

Tabla de contenidos

Planteamiento del problema

Implementar una interfaz gráfica de usuario, para un modelo de cajero automático diseñado originalmente como una aplicación de consola.

A continuación, se especifican los requerimientos específicos para el ATM-00.

Cada usuario solo puede tener una cuenta en el banco. Los usuarios del ATM deben poder ver el saldo de su cuenta, retirar efectivo (es decir, sacar dinero de una cuenta) y depositar fondos (es 2 decir, meter dinero en una cuenta). El ATM tendrá un dispensador de efectivo que proporciona el efectivo al usuario, y una ranura de depósito que recibe billetes para depósitos del usuario.

El dispensador de efectivo comienza cada día cargado con 2,000 billetes de $100.00, 1,500 billetes de $200.00 y 1,000 billetes de $500.00. (Los retiros solamente podrán ser en múltiplos de $100.00). Una sesión con el ATM consiste en la autenticación de un usuario (es decir, proporcionar la identidad del usuario) con base en un número de cuenta y un número de identificación personal (NIP), seguida de la creación y la ejecución de transacciones financieras.
Así que el ATM pide al usuario que escriba su número de cuenta y su NIP. Para autenticar un usuario y realizar transacciones, el ATM debe interactuar con su base de datos de información sobre las cuentas del banco (se utilizarán arreglos para esta información que será la base de datos del banco, los datos del arreglo deberán cargarse desde el archivo de respaldo).

Para cada cuenta de banco, se almacena un número de cuenta, un NIP y un saldo que indica la cantidad de dinero en la cuenta. Se asumirá solo un ATM, por lo que no es necesario considerar que varios ATM se puedan acceder a esta información al mismo tiempo. Se supondrá que el banco no realizará modificaciones en la información que hay en su base de datos mientras un usuario accede al ATM.

Para simplificar el problema, no se considerarán elementos de seguridad que normalmente existen en un ATM real. Al utilizar al ATM (y suponiendo que nadie más lo hace en ese momento), el usuario deberá realizar la siguiente secuencia de eventos, las figuras citadas en esta secuencia de pasos se relacionan al documento ATM-D.pdf y las podrán considerar para su propio diseño de la GUI:

1. La pantalla muestra un mensaje de bienvenida y pide al usuario que introduzca un número de cuenta.
2. El usuario introduce un número de cuenta de cinco dígitos, mediante el uso del teclado.
3. En la pantalla aparece un mensaje, en el que se pide al usuario que introduzca su NIP (número de identificación personal) asociado con el número de cuenta especificado.
4. El usuario introduce un NIP de cinco dígitos mediante el teclado numérico

5. Si el usuario introduce un número de cuenta válido y el NIP correcto para esa cuenta, la pantalla muestra el menú principal (figura 2.18).

Si el usuario introduce un número de cuenta 4 invalido o un NIP incorrecto, la pantalla muestra un mensaje apropiado y después el ATM regresa al paso 1 para reiniciar el proceso de autenticación. Una vez que el ATM autentica al usuario, el menú principal (figura 2.18) debe contener una opción numerada para cada uno de los tres tipos de transacciones: solicitud de saldo (opción 1), retiro (opción 2) y depósito (opción 3). El menú principal también debe contener una opción para que el usuario pueda salir del sistema (opción 4). Después el usuario elegirá si desea realizar una transacción (oprimiendo 1, 2 ó 3) o salir del sistema (oprimiendo 4). Si el usuario oprime 1 para solicitar su saldo, la pantalla mostrará el saldo de esa cuenta bancaria. Para ello, el ATM deberá obtener el saldo de la base de datos del banco.

Descripción de la Solución

El modelo implementado dentro de este proyecto es el planteado en el libro “como programar en Java de Deitel & Deitel” donde se presenta un ATM con interfaz de consola, nosotros implementaremos una interfaz gráfica, la cual, permitirá al usuario una interacción más intuitiva y agradable.

También se modificó la forma en cómo se hacen las validaciones y modificaciones sobre las cuentas, pues la simulación de una base de datos inicialmente era un array que sólo se modificaba en tiempo de ejecución, sin embargo, al ejecutar el programa este arreglo se mantenía igual. Para reemplazar este apartado utilizamos una simulación de base de datos distinta, donde hicimos uso de archivos, para que las cuentas se modifiquen a la hora de realizar una transacción como un depósito, retiro o una consulta de saldo. Se utilizó un ArrayList que guarda los objetos de tipo Cuenta que se almacenan en el archivo, el ArrayList sirve para grabar las modificaciones que se hagan sobre un objeto en el archivo.

- La clase BaseDatosBanco será la encargada de la gestión de los archivos, dado que en ella se encontrarán los métodos para crear el archivo, insertar datos en el archivo y leer el archivo.

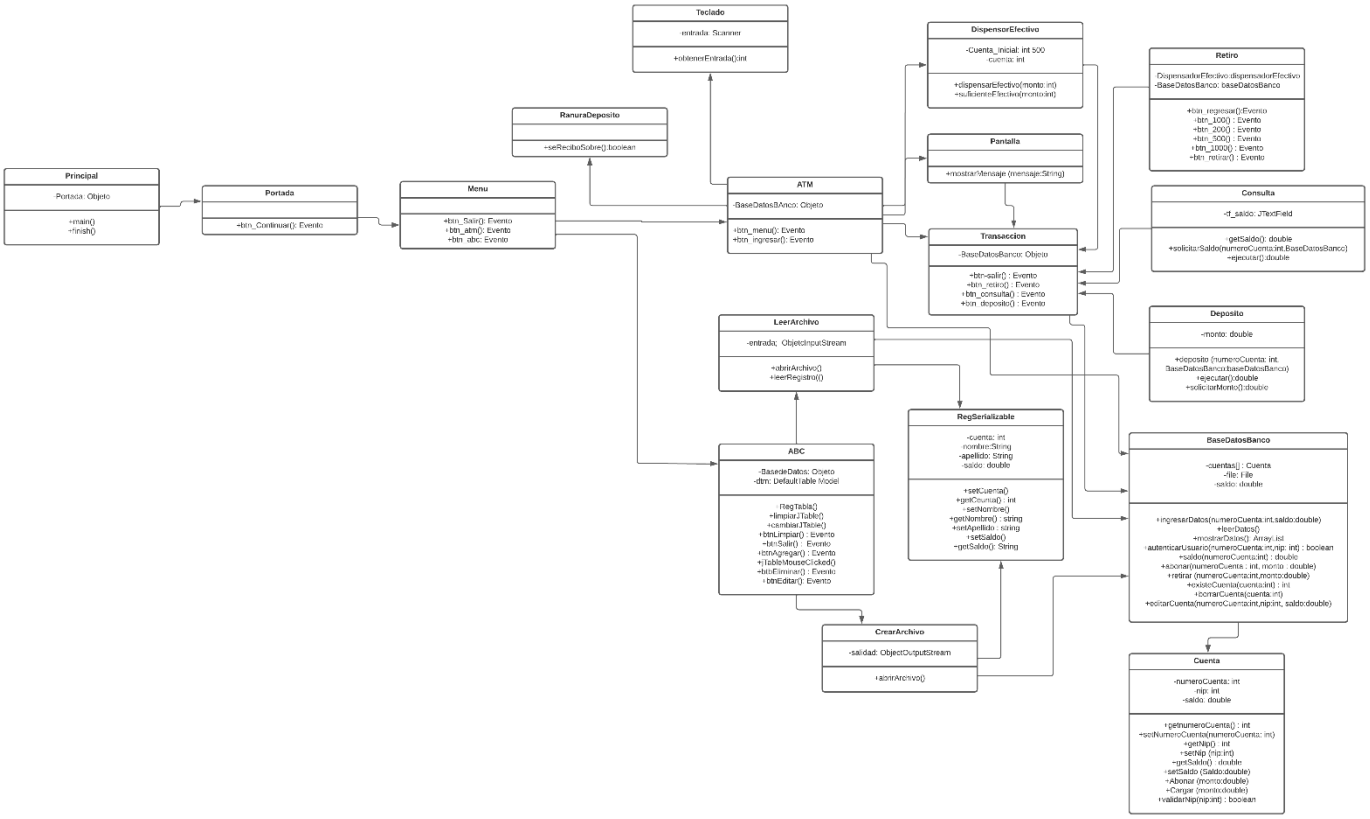
Además, también contendrá los métodos que le permitan al Ejecutivo de Cuenta dar de alta, eliminar o modificar los datos de una cuenta. Adicional a las transacciones que permite el ATM decidimos agregar una cuenta Ejecutivo que se encargará de hacer un ABC (Altas, Bajas y Cambios) de las cuentas.
En la pantalla del ejecutivo se muestra una tabla donde se le muestran todas las cuentas almacenadas en el archivo, el cual se actualiza en tiempo real de acuerdo con las modificaciones que se hagan en los campos de NIP y saldo, el campo del número de cuenta no podrá ser modificado dado que cada cuenta es única y ese campo es el equivalente a una llave primaria dentro de una Base de Datos.

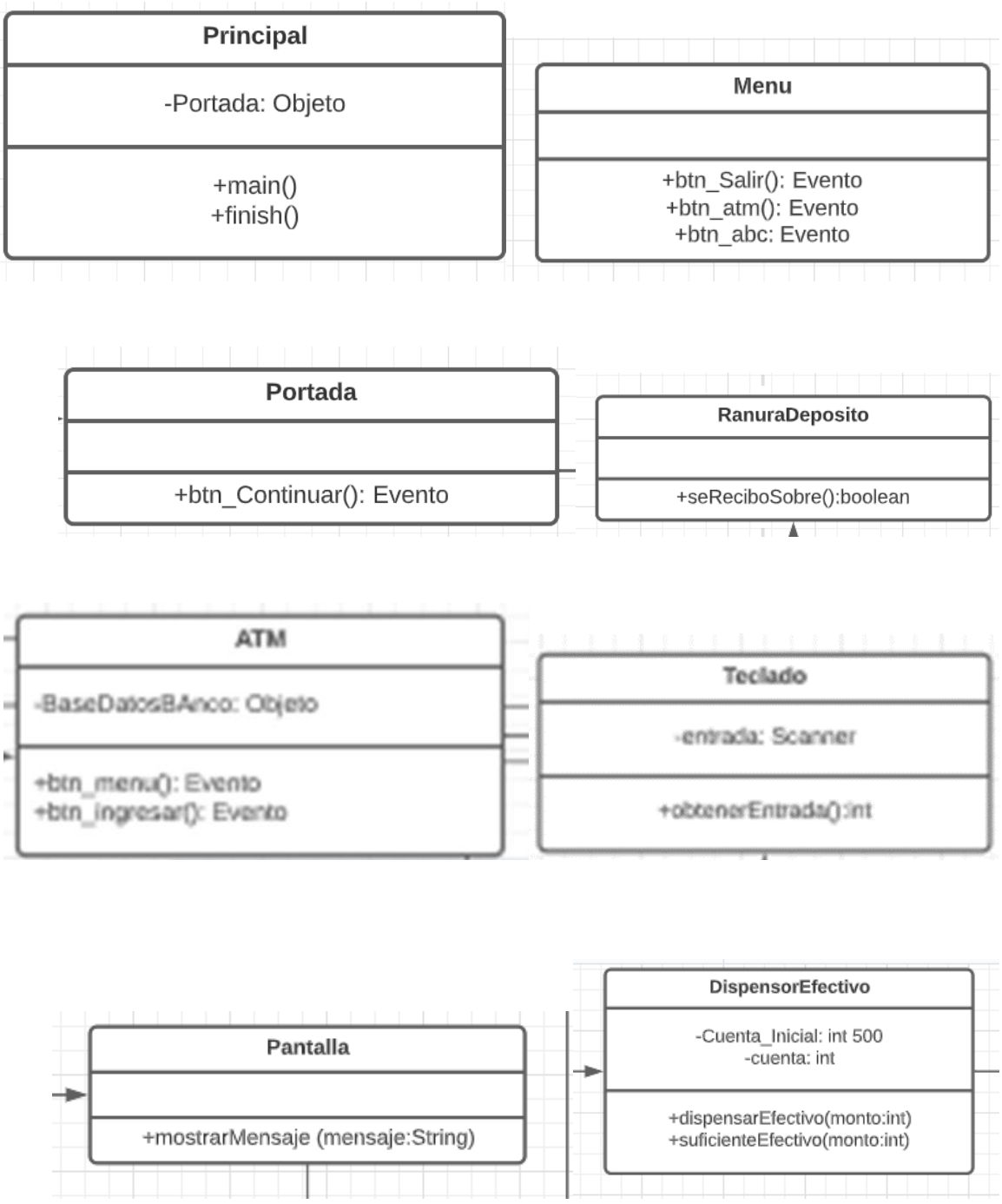
A la hora de que el cliente inicia su sesión se tiene una validación dentro del archivo de la base de datos para que le permita al usuario avanzar a las opciones de transacción sólo si su número de cuenta y su NIP son correctos, en caso contrario, niega la solicitud y se solicitan los datos nuevamente.
Para la consulta de saldo se accede a un método de la clase BaseDatosBanco que muestra el saldo del usuario en tiempo real, este saldo se actualiza de acuerdo con las transacciones que realice el usuario. Para el retiro, el ATM cuenta con unas opciones de monto predeterminadas y una opción para ingresar un monto que debe ser mayor a cero, en caso de que se solicite un monto mayor al saldo disponible se

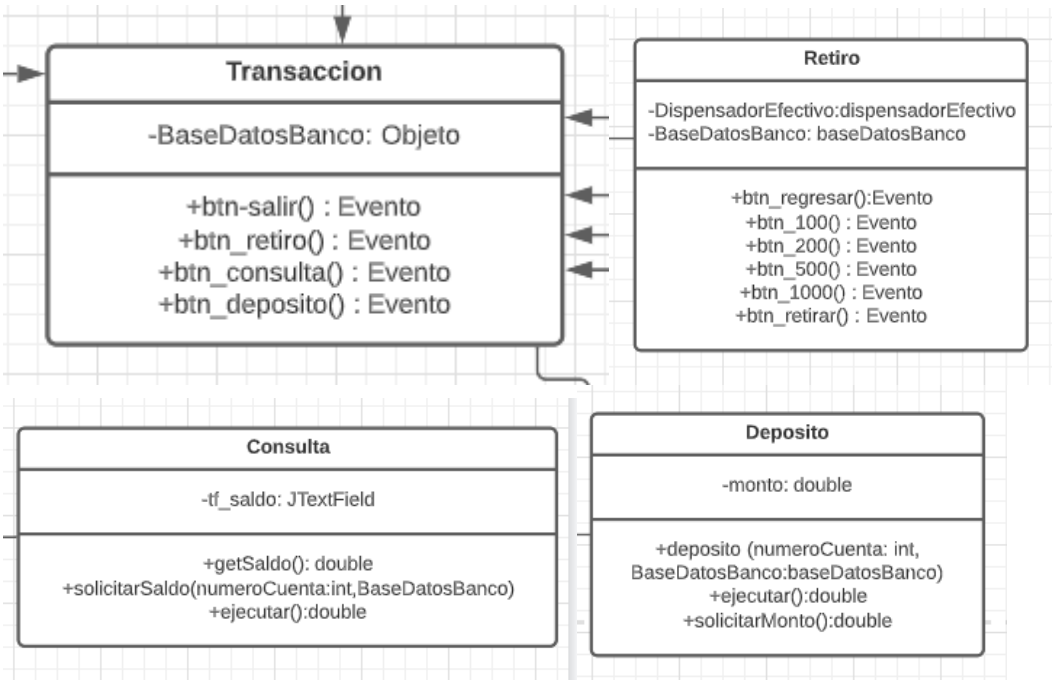
manda un mensaje y se niega la petición, en caso de que el saldo sea menor o igual al saldo total se debe presionar el botón del Dispensador Efectivo para confirmar el retiro, el ATM cuenta con una determinada cantidad de billetes, si el retiro es mayor a la cantidad disponible se muestra un mensaje que indica que el retiro no puede proceder por falta de efectivo.

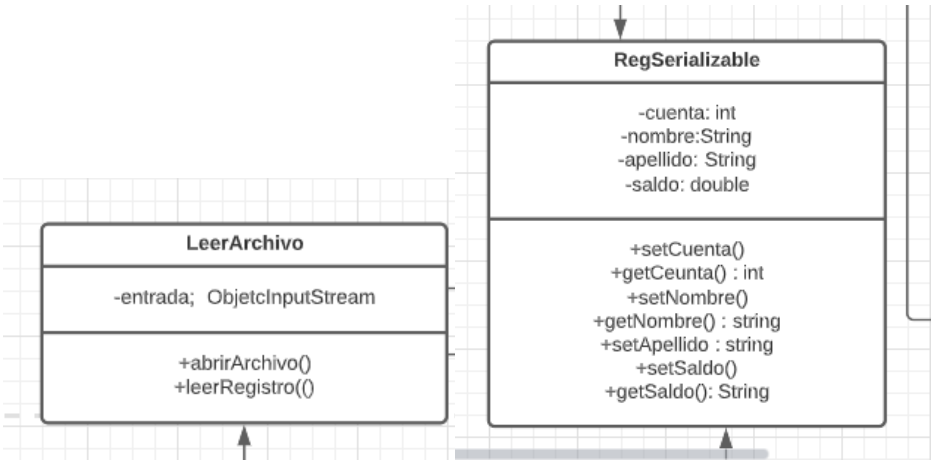
Para el depósito se debe ingresar una cantidad, después se le da un tiempo al usuario para que apriete el botón Ranura Deposito que simula el depósito del efectivo de acuerdo con la cantidad, si tarda más de diez segundos en presionar el botón se cancela la transacción, si lo hace dentro del tiempo otorgado se realiza la transacción y se modifica el saldo del usuario. Para esta validación de tiempo se utilizó una clase Timer que nos permite utilizar un contador

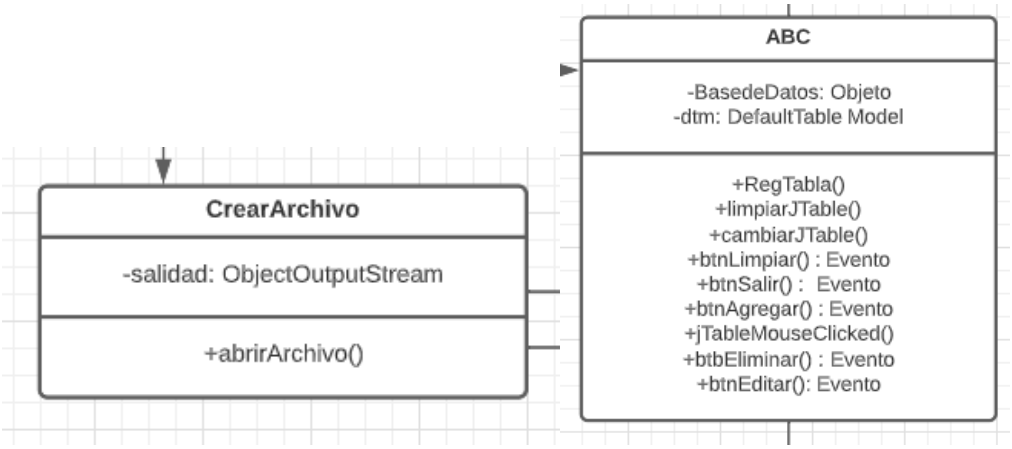
Diagramas de Clase

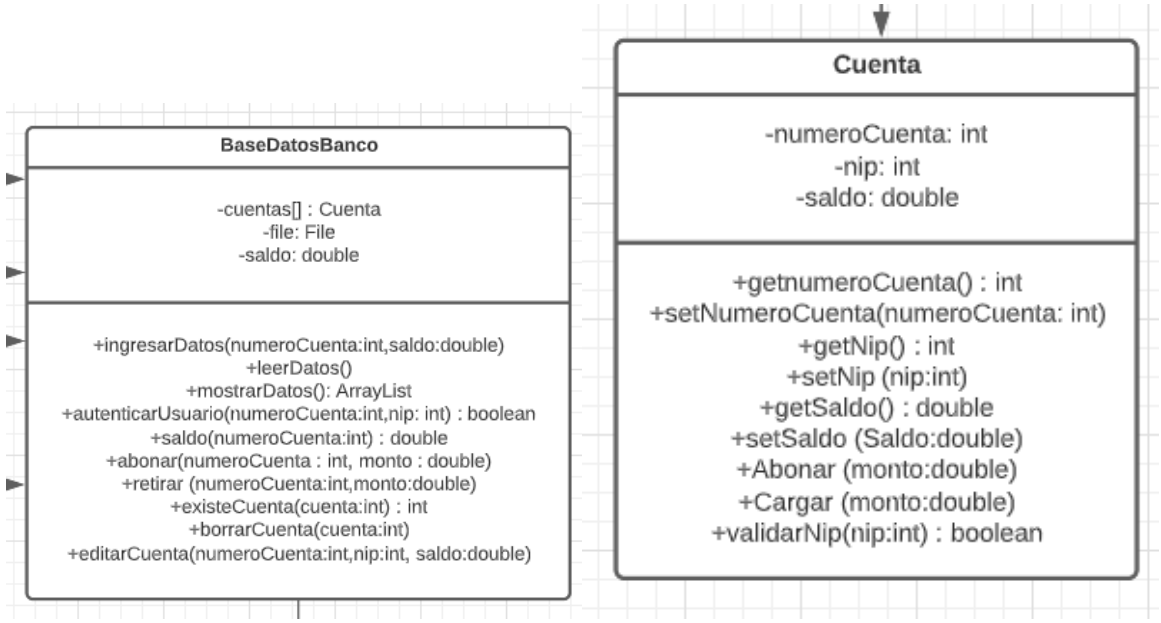



Principal
-Portada: Objeto
+main()
+finish()
Menu
+btn_Salir(): Evento
+btn_atm(): Evento
+btn_abc: Evento
Portada
+btn_Continuar(): Evento
RanuraDeposito
+seReciboSobre():boolean
ATM
-BaseDatosBAnco: Objeto
+btn_menu(): Evento
+btn_ingresar(): Evento
Teclado
-entrada: Scanner
+obtenerEntrada():int
Pantalla
+mostrarMensaje (mensaje:String)
DispensorEfectivo
-Cuenta_Inicial: int 500
-cuenta: int
+dispensarEfectivo(monto:int)
+suficienteEfectivo(monto:int)

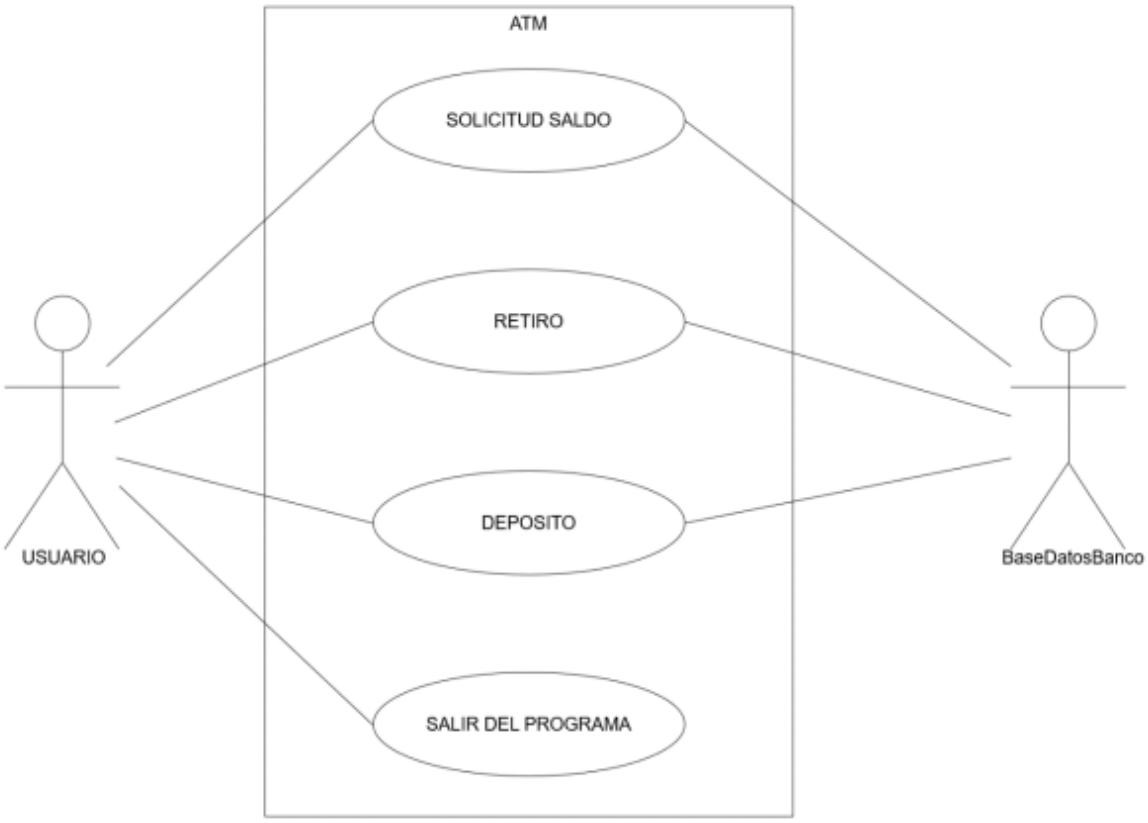

Transaccion
-BaseDatosBanco: Objeto
+btn-salir() : Evento
+btn_retiro() : Evento
+btn_consulta() : Evento
+btn_deposito() : Evento
Retiro
-DispensadorEfectivo:dispensadorEfectivo
-BaseDatosBanco: baseDatosBanco
+btn_regresar():Evento
+btn_100() : Evento
+btn_200() : Evento
+btn_500() : Evento
+btn_1000() : Evento
+btn_retirar() : Evento
Consulta
-tf_saldo: JTextField
+getSaldo(): double
+solicitarSaldo(numeroCuenta:int,BaseDatosBanco)
+ejecutar():double
Deposito
-monto: double
+deposito (numeroCuenta: int,
BaseDatosBanco:baseDatosBanco)
+ejecutar():double
+solicitarMonto():double

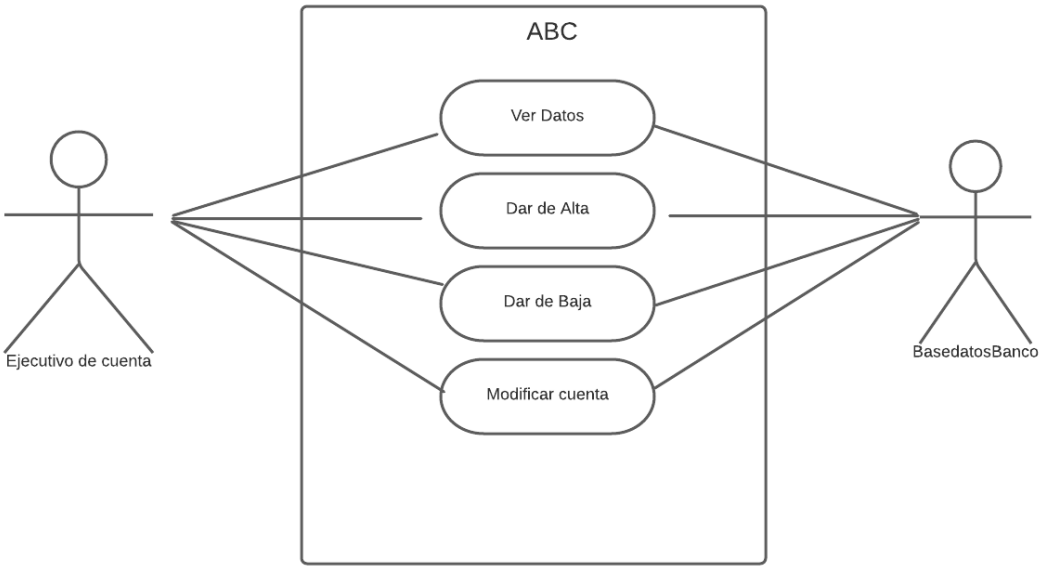

RegSerializable
-cuenta: int
-nombre:String
-apellido: String
-saldo: double
+setCuenta()
+getCeunta() : int
+setNombre()
+getNombre() : string
+setApellido : string
+setSaldo()
+getSaldo(): String
LeerArchivo
-entrada; ObjetcInputStream
+abrirArchivo()
+leerRegistro(()


ABC
-BasedeDatos: Objeto
-dtm: DefaultTable Model
+RegTabla()
+limpiarJTable()
+cambiarJTable()
+btnLimpiar() : Evento
+btnSalir() : Evento
+btnAgregar() : Evento
+jTableMouseClicked()
+btbEliminar() : Evento
+btnEditar(): Evento
CrearArchivo
-salidad: ObjectOutputStream
+abrirArchivo()


Cuenta
-numeroCuenta: int
-nip: int
-saldo: double
+getnumeroCuenta() : int
+setNumeroCuenta(numeroCuenta: int)
+getNip() : int
+setNip (nip:int)
+getSaldo() : double
+setSaldo (Saldo:double)
+Abonar (monto:double)
+Cargar (monto:double)
+validarNip(nip:int) : boolean
BaseDatosBanco
-cuentas[] : Cuenta
-file: File
-saldo: double
+ingresarDatos(numeroCuenta:int,saldo:double)
+leerDatos()
+mostrarDatos(): ArrayList
+autenticarUsuario(numeroCuenta:int,nip: int) : boolean
+saldo(numeroCuenta:int) : double
+abonar(numeroCuenta : int, monto : double)
+retirar (numeroCuenta:int,monto:double)
+existeCuenta(cuenta:int) : int
+borrarCuenta(cuenta:int)
+editarCuenta(numeroCuenta:int,nip:int, saldo:double)

Diagramas de caso de uso





Pseudocódigo de métodos

// atm metodos de la clase ATM-07

```
Sub Run():
!Enunciados
{
        mientras(verdadero)
                mientras(!usuarioAutenticado)
                        escribir("Bienvenidos");
                Fin mientras
                realizarTransacciones;
                UsuarioAutenticado <- falso;
                Numero de cuenta <- 0;
                escribir ("Gracias! Adios!");
        Fin mientras
}
Fin Sub


Sub AutenticarUsuario():
!Enunciados
{
```

```
        Escribir("Escriba su numero de cuenta");
        Entero numero de cuenta <- teclado.ObtenerEntrada();
        Escribir ("Escriba su NIP");
        Entero nip <- teclado.ObtenerEntrada();
        UsuarioAutenticad<-baseDatosBanco.autenticarUsuario(numero de cuenta,
        nip);
        si(UsuarioAutenticado)
                numeroCunetaActual <- numeroCuenta;
        Fin si
        sino
                Imprimir("Numero de cuenta o NIP invalido. Intente de nuevo");
        Fin sino
}
Fin Sub

Sub realizarTransacciones():
!Enunciados
{
        Transaccion transaccionActual <- nulo;
        Boleano usuarioSalio <- falso;
        mientras(!usuarioSalio)
                Entero seleccionMenuPricipal <- mostrarMenuPrincipal()
                Caso(seleccionMenuPricipal)
                        caso SOLICITUD_SALDO;
                        caso RETIRO;
                        caso DEPOSITO;
                        transaccionActual <-
                        crearTransaccion(seleccionMenuPricipal);
                        transaccionActual.ejecutar();
                        caso SALIR
                        Escribir("cerrando el sistema");
                        usuarioSalio<-verdadero;
                        Por defecto
                        Escribir("No introdujo una seleccion valida. Intente de nuevo");
                Fin cambio
        Fin mientras
}
Fin Sub

Sub mostrarMenuPrincipal():
!Enunciados
{
        Escribir ("Menu principal");
        Escribir ("1.- Ver mi saldo");
        Escribir ("2.- Retirar efectivo");
```

```
        Escribir ("3.- Depositar fondos");
        Escribir ("4.- Salir");
        Escribir ("Escriba una opcion");
        retornar teclado.obtenerEntrada();
}
Fin Sub

Sub Transaccion crearTransaccion (Entero tipo):
!Enunciados
{
        transaccion temp <- nulo;
        caso(tipo)
                caso SOLICITUD_SALDO;
                temp<- nuevo SolicitudSaldo (numeroCunetaActual, pantalla,
                baseDatosBanco)
                caso RETIRO;
                temp = nuevo Retiro( numeroCuentaActual, pantalla,
                baseDatosBanco, teclado, dispensadorEfectivo )
                caso DEPOSITO;
                 temp <- new Deposito( numeroCuentaActual, pantalla,
                baseDatosBanco, teclado, ranuraDeposito );
        Fin caso
        retornar temp;
}
Fin Sub

//BaseDatosBanco  métodos de la clase ATM-07

sub BaseDatosBanco ()
!Enunciados
{
   cuentas <- new Cuenta[ 2 ];
   cuentas[ 0 ] <- new Cuenta( 12345, 54321, 1000.0, 1200.0 );
   cuentas[ 1 ] <- new Cuenta( 98765, 56789, 200.0, 200.0 );
}
Fin sub

Sub obtenerCuenta(Entero numero de cuentaActual)
{
        para ( Cuenta cuentaActual : cuentas )
                si ( cuentaActual.obtenerNumeroCuenta() == numeroCuenta )
                        retorna cuentaActual;
                Fin si
        fin para
retorna nulo;
```

```
}
Fin sub

Sub autenticar usuario():
{
        Cuenta cuentaUsuario<-obtenerCuenta(numeroCuentaUsuario);
        si ( cuentaUsuario != null )
              retorna cuentaUsuario.validarNIP( nipUsuario );
            sino
              retorna falso
        Fin sino
}
fin sub

Sub obtenerSaldoDisponible( entero numeroCuentaUsuario )
!Enunciados
{
   retorna obtenerCuenta( numeroCuentaUsuario ).obtenerSaldoDisponible
}
Fin metodo obtenerSaldoTotal

Sub abonar( int numeroCuentaUsuario, doble monto )
!Enunciados
{
   obtenerCuenta( numeroCuentaUsuario ).abonar( monto );
}
Fin sub


Sub cargar (entero numeroCuentaUsuario, doble monto )
!Enunciados
{
   obtenerCuenta( numeroCuentaUsuario ).cargar( monto );
}
Fin sub


//Cuenta métodos de la clase ATM-07

Sub Cuenta  (entero numeroCuenta, entero nip, Doble saldoDisponible, doble
saldoTotal): cuenta
!Enunciados
{
        si ( nipUsuario == nip )
                retorna verdadero;
```

```
        fin si
        sino
                retorna falso;
        Fin sino
}
Fin sub

Sub obtenerSaldoDisponible
!Enunciados
{
        retorna saldoDisponible;
}
Fin sub

Sub obtenerSaldoTotal
!Enunciados
{
        retorna saldoTotal;
}
Fin sub

Sub abonar
!Enunciados
{
        saldoTotal += monto;
}
Fin sub

Sub cargar
!Enunciados
{
        saldoDisponible -= monto;
        saldoTotal -= monto;
}
Fin sub

Sub obtenerNumeroCuenta
!Enunciados
{
        retorna numeroCuenta;
}
Fin sub

//Deposito  métodos de la clase ATM-07
```

```
Sub Deposito( int numeroCuentaUsuario, Pantalla pantallaATM, BaseDatosBanco
baseDatosBanco, Teclado tecladoATM, RanuraDeposito ranuraDepositoATM ):
!Enunciados
{
   super( numeroCuentaUsuario, pantallaATM, baseDatosBanco );
   teclado <- tecladoATM;
   ranuraDeposito <- ranuraDepositoATM;
}
Fin sub

Sub ejecutar():
!Enunciados
{
   BaseDatosBanco baseDatosBanco <- obtenerBaseDatosBanco();
   Pantalla pantalla <- obtenerPantalla();

   monto <- pedirMontoADepositar();

   si ( monto != CANCELO )
      Escribir("Inserte un sobre que contenga " );
      Escribir( monto );
      Escribir( "." );

      seRecibioSobre <- ranuraDeposito.seRecibioSobre();

      si ( seRecibioSobre
         Escribir( "Se recibio su sobre de " + "deposito.\nNOTA: El dinero que acaba
de depositar no " + "estara disponible sino hasta que verifiquemos el monto del " +
"efectivo y cualquier cheque incluido." );

         baseDatosBanco.abonar( obtenerNumeroCuenta(), monto );
      Fin si
      sino
         Escribir( "No inserto un sobre de " +  "deposito, por lo que el ATM ha
cancelado su transaccion." );
      Fin sino
   Fin si
   Sino
      Escribir( "Cancelando transaccion..." );
   fin sino
  }
fin sub

Sub pedirMontoADepositar()
!Enunciados
```

```
{
   Pantalla pantalla <- obtenerPantalla();
   Escribir( "Introduzca un monto a depositar en " + "CENTAVOS (o 0 para
cancelar): " );
   entrada <. teclado.obtenerEntrada();

   SI ( entrada == CANCELO )
      retornar CANCELO;
sino
retornar ( double ) entrada / 100;
fin sino
Fin si
Fin sub
```

//DispensadorEfectivo metodos de la clase ATM-07

```
Sub DispensadorEfectivo():
!Enunciados
{
        cuenta <- CUENTA_INICIAL;
}
Fin Sub

Sub DispensarEfectivo(Entero monto):
!Enunciados
{
        Entero billetesRequeridos <- monto / 20;
        cuenta -<- billetesRequeridos;
}
Fin sub

Sub haySudicienteEfectivoDisponible():
!Enunciados
{
        Entero billetesRequeridos <- monto;
        si(cuenta >= billetesRequeridos)
                retornar verdadero;
        Fin si
        sino
                retornar falso;
        Fin sino
}
Fin sub
```

//Ejemplo practico métodos de la clase ATM-07

```
Sub main(args)
!Enunciados
{
        ATM elATM<-nuevo ATM;
        elATM.run();
}
Fin sub

//Pantalla métodos de la clase ATM-07

Sub mostrarMensaje():
!Entorno
{
        Escribir(mensaje);
}
Fin Sub

Sub mostrarLineaMensaje():
!Entorno
{
        Escribir(mensaje);
}
Fin Sub

Sub mostrarMontoDolares():
!Entorno
{
        Escribir(monto);
}
Fin Sub

//RanuraDeposito metodos de la clase ATM-07

Sub seRecibioSobre():
!Entorno
{
        retornar verdadero;
}
Fin Sub

//SolicitudSaldo  métodos de la clase ATM-07

Sub SolicitudSaldo( int numeroCuentaUsuario, Pantalla pantallaATM,
BaseDatosBanco baseDatosBanco ):
```

```
!Enunciados
{
        super( numeroCuentaUsuario, pantallaATM, baseDatosBanco );
}
Fin sub

Sub ejecutar()
!Enunciados
{
        BaseDatosBanco baseDatosBanco <- obtenerBaseDatosBanco();
        Pantalla pantalla <- obtenerPantalla();
        saldoDisponible <- baseDatosBanco.obtenerSaldoDisponible(
        obtenerNumeroCuenta());

        double saldoTotal <- baseDatosBanco.obtenerSaldoTotal(
        obtenerNumeroCuenta() );

   Escribir( "\nInformacion de saldo:" );
   Escribir( " - Saldo disponible: " );
   Escribir( saldoDisponible );
   Escribir( "\n - Saldo total:     " );
   Escribir( saldoTotal );
   Escribir( "" );
}
Fin sub

//Teclado  métodos de la clase ATM-07

Sub Teclado():
{
        entrada <- new Scanner( System.in );
}
Fin sub

Sub obtenerEntrada():
{
   retorna entrada.nextInt();
}
Fin sub

//Transacción  métodos de la clase ATM-07

Sub Transaccion( int numeroCuentaUsuario, Pantalla
pantallaATM,BaseDatosBanco baseDatosBancoATM ):
!Enunciados
```

```
{
        numeroCuenta <- numeroCuentaUsuario;
        pantalla <- pantallaATM;
        baseDatosBanco <- baseDatosBancoATM;
}
Fin sub

Sub obtenerNumeroCuenta():
!Enunciados
{
   retornar numeroCuenta;
}
Fin sub

Sub Pantalla obtenerPantalla():
!Enunciados
{
        retornarpantalla;
}
Fin sub

Sub BaseDatosBanco obtenerBaseDatosBanco():
!Enunciados
{
   return baseDatosBanco;
}
Fin sub

//Retiro métodos de la clase ATM-07

Sub  Retiro (int numeroCuentaUsuario, Pantalla pantallaATM,
     BaseDatosBanco baseDatosBanco, Teclado tecladoATM,
     DispensadorEfectivo dispensadorEfectivoATM ):

teclado = tecladoATM;
     dispensadorEfectivo = dispensadorEfectivoATM;
Fin constructor Retiro
sub ejecutar
!Enunciados
{
        boleano efectivo dispensado<-falso
        doble saldoDisponible
        BaseDatosBanco baseDatosBanco <- obtenerBaseDatosBanco();
        Pantalla pantalla <- obtenerPantalla();
        hacer
```

```
                monto <- mostrarMenuDeMontos();
                si ( monto != CANCELO )
                        saldoDisponible <- baseDatosBanco.obtenerSaldoDisponible(
                        obtenerNumeroCuenta() );
                        si ( monto <= saldoDisponible )
                                si(
                                dispensadorEfectivo.haySuficienteEfectivoDisponible(
                                monto ))
                                baseDatosBanco.cargar( obtenerNumeroCuenta(),
                                monto );
                                dispensadorEfectivo.dispensarEfectivo( monto );
                                efectivoDispensado <- true;
                                escribir ("Tome ahora su efectivo." );
                                sino
                                        Escribir("No hay suficientes fondos en su cuenta."
                                        + "Seleccione un monto menor." )
                                fin sino
                                fin si
                        fin si
                fin si
                sino
                        Escribir( "Cancelando transaccion..." );
                fin sino
        mientras ( !efectivoDispensado );
}
fin sub


Sub mostrarMenuDeMontos()
!Enunciados
{
   opcionUsuario <- 0;
   Pantalla pantalla <- obtenerPantalla();
   montos[] <- { 0, 20, 40, 60, 100, 200 };

   Mientras ( opcionUsuario == 0 )

      Escribir( "\nMenu de retiro:" );
      Escribir( "1 - $20" );
      Escribir( "2 - $40" );
      Escribir( "3 - $60" );
      Escribir( "4 - $100" );
      Escribir( "5 - $200" );
      Escribir( "6 - Cancelar transaccion" );
      Escribir( "Seleccione un monto a retirar: " );
```

```
        entrada = teclado.obtenerEntrada();
        caso ( entrada )
           caso 1:
           caso 2:
           caso 3:
           caso 4:
           caso 5:
opcionUsuario <- montos[ entrada ]; // guarda la elecci□n del usuario
           caso CANCELO:
opcionUsuario <- CANCELO;
           default:
pantalla.mostrarLineaMensaje( Seleccion invalida. Intente de nuevo." );
        Fin caso
    Fin mientras

     retornar opcionUsuario;
}
Fin sub
```

Listado de los mÓdulos

Método ATM.java

Método Main()

```
9           public int token;
10          public boolean pass = false;
11          BaseDatosBanco baseDatosBanco = new BaseDatosBanco();
12
13          private Pantalla pantalla;
14          private boolean usuarioAutenticado;
15          private int cuenta;
16          private String key;
17
18          public ATM(){
19                  pantalla = new Pantalla();
20                  pantalla.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
21                  pantalla.setResizable(false);
22                  pantalla.setLocationRelativeTo(null);
23                  pantalla.Portada();
24
25                  while(true){
26                          while(true){
27                                  pass = pantalla.getPass();
28                                  System.out.println("pass of portada: "+pass);  // Sin este print no sirve
29                                  if(pass){
                                            clean("ingreso");
31                                          System.out.println("ingreso");
32                                          while(true){
                                                    if(pantalla.getChoose() == "user"){
34                                                          key = pantalla.getKey();
35                                                          pantalla.user.setText(key);
36                                                  }
                                                    if(pantalla.getChoose() == "password"){
```

```
19                  pantalla = new Pantalla();
20                  pantalla.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
21                  pantalla.setResizable(false);
22                  pantalla.setLocationRelativeTo(null);
23                  pantalla.Portada();
24
25                  while(true){
26                          while(true){
27                                  pass = pantalla.getPass();
28                                  System.out.println("pass of portada: "+pass);  // Sin este print no sirve
29                                  if(pass){
                                            clean("ingreso");
31                                          System.out.println("ingreso");
32                                          while(true){
                                                    if(pantalla.getChoose() == "user"){
34                                                          key = pantalla.getKey();
35                                                          pantalla.user.setText(key);
36                                                  }
                                                    if(pantalla.getChoose() == "password"){
38                                                          key = pantalla.getKey();
39                                                          pantalla.password.setText(key);
40                                                  }
41                                                  pass = pantalla.getPass();
42                                                  System.out.println("pass of ingreso: "+pass);
43                                                  if(pass){
44                                                          cuenta = pantalla.getCount();
45                                                          break;
46                                                  }
47                                          }
48                                          break;
```

Método limpiar()

```
136         public void clean(String function, double sd, double sf){
137                 pantalla.getContentPane().removeAll();
138                 pantalla.teclado();
                    if(function == "saldo"){
140                         pantalla.Saldo(sd, sf);
141                 }
142                 pantalla.revalidate();
143                 pantalla.repaint();
144         }
```

Modulo BaseDatosBanco.java

Método Cuenta()

```
5     private Cuenta cuentas[];
6
7     public BaseDatosBanco()
8     {
9        cuentas = new Cuenta[ 2 ];
10       cuentas[ 0 ] = new Cuenta( 12345, 54321, 1000.0, 1200.0 );
11       cuentas[ 1 ] = new Cuenta( 98765, 56789, 200.0, 200.0 );
12    }
13
14    private Cuenta obtenerCuenta( int numeroCuenta )
15    {
16       for ( Cuenta cuentaActual : cuentas )
17       {
18
19          if ( cuentaActual.obtenerNumeroCuenta() == numeroCuenta )
20             return cuentaActual;
21       }
22
23       return null;
24    }
25
26    public boolean autenticarUsuario( int numeroCuentaUsuario, int nipUsuario )
27    {
28       Cuenta cuentaUsuario = obtenerCuenta( numeroCuentaUsuario );
29
30       if ( cuentaUsuario != null )
31          return cuentaUsuario.validarNIP( nipUsuario );
32       else
33          return false;
34    }
```

Modulo Cuenta.java

Método cuenta ()

```
10      public Cuenta( int elNumeroDeCuenta, int elNIP,
11          double elSaldoDisponible, double elSaldoTotal )
12      {
13          numeroCuenta = elNumeroDeCuenta;
14          nip = elNIP;
15          saldoDisponible = elSaldoDisponible;
16          saldoTotal = elSaldoTotal;
17      }
```

Modulo Deposito.java

Método Deposito ()

```
4   public class Deposito extends Transactions{
5
6           public void ejecutar(int cuenta, double monto){
7                   System.out.println(cuenta + " " + monto);
8                   baseDatosBanco.abonar(cuenta, monto);
9           }
10  }
```

Modulo Patalla.java

Método Portada ()

```
23      public void Portada(){
24              // Label
25              lblBienvenida= new Label("Bienvenido");
26              lblBienvenida.setBounds(250,30,100,30);
27              lblBienvenida.setFont(new Font("Consolas", 1, 13));
28              add(lblBienvenida);
29
30              // Label
31              lblmensaje= new Label("Este es tu cajero automatico");
32              lblmensaje.setBounds(200,130,200,30);
33              lblmensaje.setFont(new Font("Consolas", 1, 13));
34              add(lblmensaje);
35
36              // Button
37              btnSig = new Button("Continuar");
38              btnSig.setBounds(250, 210, 100, 30);
39              add(btnSig);
40              btnSig.addActionListener(this);
41
42              // General settings
43              setTitle("Portada");
44              setLayout(null);
45              setVisible(true);
46      }
47
```

Método Ingreso ()

```
48          public void Ingreso(){
49                  pass = false; // false para misma variable
50
51                  // Label
52                  lblWlk= new Label("Bienvenido");
53                  lblWlk.setBounds(250,30,100,30);
54                  lblWlk.setFont(new Font("Consolas", 1, 13));
55                  add(lblWlk);
56
57                  // User
58                  lblUsr= new Label("Usuario");
59                  lblUsr.setBounds(100,100,75,30);
60                  lblUsr.setFont(new Font("Consolas", 1, 13));
61                  add(lblUsr);
62
63                  user = new JTextField();
64                  user.setBounds(200,100,200,30);
65                  user.addFocusListener(new FocusListener(){
66
67                          @Override
                    public void focusGained(FocusEvent fe) {
69                              choose = "user";
70                              key = "";
71                  }
72
73                          @Override
                    public void focusLost(FocusEvent fe) {
75                              choose = "user";
76                  }
77                  });
78                  add(user);
```

Método Transaction()

```
114
115         public void Transactions(){
116                 lblMessage= new Label("Selecciona tu transaction");
117                 lblMessage.setBounds(250,30,200,30);
118                 lblMessage.setFont(new Font("Consolas", 1, 13));
119                 add(lblMessage);
120
121                 // Botones
122                 btnSaldo = new Button("Saldo");
123                 btnSaldo.setBounds(50, 110, 100, 30);
124                 add(btnSaldo);
125                 btnSaldo.addActionListener(this);
126
127                 btnDeposito = new Button("Deposito");
128                 btnDeposito.setBounds(250, 110, 100, 30);
129                 add(btnDeposito);
130                 btnDeposito.addActionListener(this);
131
132                 btnRetirar = new Button("Retiro");
133                 btnRetirar.setBounds(450, 110, 100, 30);
134                 add(btnRetirar);
135                 btnRetirar.addActionListener(this);
136
137                 btnCancelar = new Button("Salir");
138                 btnCancelar.setBounds(250, 210, 100, 30);
139                 add(btnCancelar);
140                 btnCancelar.addActionListener(this);
141
142                 setTitle("Menu");
143
144                 setLayout(null);
```

Método Saldo()

```
148     public void Saldo(double saldoDisponible, double saldoTotal){
149             lblMensaje= new Label("Saldo disponible: " + saldoDisponible); // Instancia de la variable de ti
150             lblMensaje.setBounds(230,30,200,30); // Esta funcion establece las medidas
151             lblMensaje.setFont(new Font("Consolas", 1, 13));  // Fuente y tamaño del texto
152             add(lblMensaje);  // Se tiene que añadir a la ventana actual con add()
153
154             lblMessage= new Label("Saldo total: " + saldoTotal);
155             lblMessage.setBounds(230,100,200,30);
156             lblMessage.setFont(new Font("Consolas", 1, 13));
157             add(lblMessage);
158
159             btnSalir = new Button("Salir");
160             btnSalir.setBounds(250, 210, 100, 30);
161             add(btnSalir);
162             btnSalir.addActionListener(this);
163
164             setTitle("Saldo");
165             setLayout(null);
166             setVisible(true);
167     }
```

Método Deposito()

```
169         public void Deposito(){
170                 lblMessage= new Label("Monto a depositar");
171                 lblMessage.setBounds(250,30,120,30);
172                 lblMessage.setFont(new Font("Consolas", 1, 13));
173                 add(lblMessage);
174
175                 tfMonto = new TextField();
176                 tfMonto.setBounds(250,100,100,30);
177                 tfMonto.setFont(new Font("Consolas", 1, 13));
178                 add(tfMonto);
179
180                 btnMonto = new Button("Depositar");
181                 btnMonto.setBounds(200, 210, 100, 30);
182                 add(btnMonto);
183                 btnMonto.addActionListener(this);
184
185                 btnSalir = new Button("Salir");
186                 btnSalir.setBounds(310, 210, 100, 30);
187                 add(btnSalir);
188                 btnSalir.addActionListener(this);
189
190                 setTitle("Deposito");
191                 setLayout(null);
192                 setVisible(true);
193         }
```

Método Retiro()

```
195         public void Retiro(){
196                 lblMessage= new Label("¿Cuanto desea retirar?");
197                 lblMessage.setBounds(240,30,150,30);
198                 lblMessage.setFont(new Font("Consolas", 1, 13));
199                 add(lblMessage);
200
201                 tfMonto = new TextField();
202                 tfMonto.setBounds(250,100,100,30);
203                 tfMonto.setFont(new Font("Consolas", 1, 13));
204                 add(tfMonto);
205
206                 btnRetiro = new Button("Retirar");
207                 btnRetiro.setBounds(200, 210, 100, 30);
208                 add(btnRetiro);
209                 btnRetiro.addActionListener(this);
210
211                 btnSalir = new Button("Salir");
212                 btnSalir.setBounds(310, 210, 100, 30);
213                 add(btnSalir);
214                 btnSalir.addActionListener(this);
215
216                 setTitle("Retiro");
217                 setLayout(null);
218                 setVisible(true);
219         }
```

Modulo Retiro.java

Método Retiro()

```
4   public class Retiro extends Transactions{
5
7           public void ejecutar(int cuenta, double monto){
8                   System.out.println(cuenta + " " + monto);
9                   baseDatosBanco.cargar(cuenta, monto);
10          }
11  }
```

Modulo Saldo.java

Método SaldoDisponible()

```
4   public class Saldo extends Transactions{
5
6           public double getSaldoDisponible(int cuenta){
7                   return baseDatosBanco.obtenerSaldoDisponible( cuenta );
8           }
9
10          public double getSaldoTotal(int cuenta){
11                  return baseDatosBanco.obtenerSaldoTotal( cuenta );
12          }
13
14  }
15
```

Modulo Teclado.java

Método Teclado()

```
2
3   import java.awt.*;
4   import java.awt.event.*;
5   import javax.swing.*;
6
7   public class Teclado extends JFrame implements ActionListener{
8
9           public Button uno, dos, tres, cuatro, cinco, seis, siete, ocho, nueve, cero, borrar;
10          public JTextField ranura;
11          public String key = "";
12
13          public Teclado(){
14                  teclado();
15          }
16
17          public void teclado(){
18
19                  // Uno
20                  uno = new Button("1");
21                  uno.setBounds(245, 400, 30, 30);
22                  add(uno);
23                  uno.addActionListener(this);
24
25                  // Dos
26                  dos = new Button("2");
27                  dos.setBounds(285, 400, 30, 30);
28                  add(dos);
29                  dos.addActionListener(this);
30
```

Método Transactions.java

Método Transactions()

```
public class Transactions{

        public BaseDatosBanco baseDatosBanco = new BaseDatosBanco();
        public double saldoDisponible;
        public double saldoTotal;
        public double[] saldos;

        public double[] saldos(int cuenta){
                saldoDisponible = baseDatosBanco.obtenerSaldoDisponible( cuenta );
                saldoTotal = baseDatosBanco.obtenerSaldoTotal( cuenta );
                saldos = new double[]{saldoDisponible, saldoTotal};
                return saldos;
        }

        public void ejecutar(int cuenta, double monto){
                //return saldos;
        }
}
```

atm d

Modulo BaseDatosBanco.java
Método BaseDatosBanco()

```
1   package atm.d;
2
3   // BaseDatosBanco.java
4   // Representa a la base de datos de informaci□n de cuentas bancarias
5
6   public class BaseDatosBanco
7   {
8      private Cuenta cuentas[]; // arreglo de objetos Cuenta
9
10     // el constructor sin argumentos de BaseDatosBanco inicializa a cuentas
11     public BaseDatosBanco()
12     {
13        cuentas = new Cuenta[ 2 ]; // s□lo 2 cuentas para probar
14        cuentas[ 0 ] = new Cuenta( 12345, 54321, 1000.0, 1200.0 );
15        cuentas[ 1 ] = new Cuenta( 98765, 56789, 200.0, 200.0 );
16     } // fin del constructor sin argumentos de BaseDatosBanco
17
18     // obtiene el objeto Cuenta que contiene el n□mero de cuenta especificado
19     private Cuenta obtenerCuenta( int numeroCuenta )
20     {
21        // itera a trav□s de cuentas, buscando el n□mero de cuenta que coincida
22        for ( Cuenta cuentaActual : cuentas )
23        {
24           // devuelve la cuenta actual si encuentra una coincidencia
25           if ( cuentaActual.obtenerNumeroCuenta() == numeroCuenta )
26              return cuentaActual;
27        } // fin de for
28
29        return null; // si no se enontr□ una cuenta que coincida, devuelve null
30     } // fin del m□todo obtenerCuenta
31
32     // determina si el n□mero de cuenta y el NIP especificados por el usuario coinciden
33     // con los de una cuenta en la base de datos
34     public boolean autenticarUsuario( int numeroCuentaUsuario, int nipUsuario )
```

```
35      {
36         // trata de obtener la cuenta con el n□mero de cuenta
37         Cuenta cuentaUsuario = obtenerCuenta( numeroCuentaUsuario );
38
39         // si la cuenta existe, devuelve el resultado del m□todo validarNIP de Cuenta
40         if ( cuentaUsuario != null )
41            return cuentaUsuario.validarNIP( nipUsuario );
42         else
43            return false; // no se encontr□ el n□mero de cuenta, por lo que devuelve false
44      } // fin del m□todo autenticarUsuario
45
46      // devuelve el saldo disponible de la Cuenta con el n□mero de cuenta especificado
47      public double obtenerSaldoDisponible( int numeroCuentaUsuario )
48      {
49         return obtenerCuenta( numeroCuentaUsuario ).obtenerSaldoDisponible();
50      } // fin del m□todo obtenerSaldoDisponible
51
52      // devuelve el saldo total de la Cuenta con el n□mero de cuenta especificado
53      public double obtenerSaldoTotal( int numeroCuentaUsuario )
54      {
55         return obtenerCuenta( numeroCuentaUsuario ).obtenerSaldoTotal();
56      } // fin del m□todo obtenerSaldoTotal
57
58      // abona un monto a la Cuenta a trav□s del n□mero de cuenta especificador
59      public void abonar( int numeroCuentaUsuario, double monto )
60      {
61         obtenerCuenta( numeroCuentaUsuario ).abonar( monto );
62      } // fin del m□todo abonar
63
64      // carga un monto a la Cuenta con el n□mero de cuenta especificado
65      public void cargar( int numeroCuentaUsuario, double monto )
66      {
67         obtenerCuenta( numeroCuentaUsuario ).cargar( monto );
68      } // fin del m□todo cargar
```

Salida

Modulo ATM.java

Método ATM ()

```java
package atm.d;

// ATM.java
// Representa a un cajero autom□tico

public class ATM
{
   private boolean usuarioAutenticado; // indica si el usuario es autenticado
   private int numeroCuentaActual; // current user's account number
   private Pantalla pantalla; // pantalla del ATM
   private Teclado teclado; // teclado del ATM
   private DispensadorEfectivo dispensadorEfectivo; // dispensador de efectivo del ATM
   private RanuraDeposito ranuraDeposito; // ranura de dep□sito del ATM
   private BaseDatosBanco baseDatosBanco; //  base de datos de informaci□n de las cuentas

   // constantes correspondientes a las opciones del men□ principal
   private static final int SOLICITUD_SALDO = 1;
   private static final int RETIRO = 2;
   private static final int DEPOSITO = 3;
   private static final int SALIR = 4;

   // el constructor sin argumentos de ATM inicializa las variables de instancia
   public ATM()
   {
      usuarioAutenticado = false; // al principio, el usuario no est□ autenticado
      numeroCuentaActual = 0; // al principio, no hay n□mero de cuenta
      pantalla = new Pantalla(); // crea la pantalla
      teclado = new Teclado(); // crea el teclado
      dispensadorEfectivo = new DispensadorEfectivo(); // crea el dispensador de efectivo
      ranuraDeposito = new RanuraDeposito(); // crea la ranura de dep□sito
      baseDatosBanco = new BaseDatosBanco(); // crea la base de datos de informaci□n de cuentas
   } // fin del constructor sin argumentos de ATM

   // inicia el ATM
```

```
34      // inicia el ATM
35      public void run()
36      {
37         // da la bienvenida al usuario y lo autentica; realiza transacciones
38         while ( true )
39         {
40            // itera mientras el usuario no haya sido autenticado
41            while ( !usuarioAutenticado )
42            {
43               pantalla.mostrarLineaMensaje( "\nBienvenido!" );
44               autenticarUsuario(); // autentica el usuario
45            } // fin de while
46
47            realizarTransacciones(); // ahora el usuario est□ autenticado
48            usuarioAutenticado = false; // restablece antes de la siguiente sesi□n con el ATM
49            numeroCuentaActual = 0; // restablece antes de la siguiente sesi□n con el ATM
50            pantalla.mostrarLineaMensaje( "\nGracias! Adios!" );
51         } // fin de while
52      } // fin del m□todo run
53
54      // trata de autenticar al usuario en la base de datos
55      private void autenticarUsuario()
56      {
57         pantalla.mostrarMensaje( "\nEscriba su numero de cuenta: " );
58         int numeroCuenta = teclado.obtenerEntrada(); // recibe como entrada el n□mero de cuenta
59         pantalla.mostrarMensaje( "\nEscriba su NIP: " ); // pide el NIP
60         int nip = teclado.obtenerEntrada(); // recibe como entrada el NIP
61
62         // establece usuarioAutenticado con el valor booleano devuelto por la base de datos
63         usuarioAutenticado =
64            baseDatosBanco.autenticarUsuario( numeroCuenta, nip );
65
66         // verifica si la autenticaci□n tuvo □xito
67         if ( usuarioAutenticado )
```

```
67         if ( usuarioAutenticado )
68         {
69            numeroCuentaActual = numeroCuenta; // guarda el # de cuenta del usuario
70         } // fin de if
71         else
72            pantalla.mostrarLineaMensaje(
73               "Numero de cuenta o NIP invalido. Intente de nuevo." );
74      } // fin del m□todo autenticarUsuario
75
76      // muestra el men□ principal y realiza transacciones
77      private void realizarTransacciones()
78      {
79         // variable local para almacenar la transacci□n que se procesa actualmente
80         Transaccion transaccionActual = null;
81
82         boolean usuarioSalio = false; // el usuario no ha elegido salir
83
84         // itera mientras que el usuario no haya elegido la opci□n para salir del sistema
85         while ( !usuarioSalio )
86         {
87            // muestra el men□ principal y obtiene la selecci□n del usuario
88            int seleccionMenuPrincipal = mostrarMenuPrincipal();
89
90            // decide c□mo proceder, con base en la opci□n del men□ seleccionada por el usuario
91            switch ( seleccionMenuPrincipal )
92            {
93               // el usuario eligi□ realizar uno de tres tipos de transacciones
94               case SOLICITUD_SALDO:
95               case RETIRO:
96               case DEPOSITO:
97
98                  // inicializa como nuevo objeto del tipo elegido
99                  transaccionActual =
100                    crearTransaccion( seleccionMenuPrincipal );
```

```
101
102                     transaccionActual.ejecutar(); // ejecuta la transacci□n
103                     break;
104                  case SALIR: // el usuario eligi□ terminar la sesi□n
105                     pantalla.mostrarLineaMensaje( "\nCerrando el sistema..." );
106                     usuarioSalio = true; // esta sesi□n con el ATM debe terminar
107                     break;
108                  default: // el usuario no introdujo un entero de 1 a 4
109                     pantalla.mostrarLineaMensaje(
110                        "\nNo introdujo una seleccion valida. Intente de nuevo." );
111                     break;
112               } // fin de switch
113            } // fin de while
114         } // fin del m□todo realizarTransacciones
115
116         // muestra el men□ principal y devuelve una selecci□n de entrada
117         private int mostrarMenuPrincipal()
118         {
119            pantalla.mostrarLineaMensaje( "\nMenu principal:" );
120            pantalla.mostrarLineaMensaje( "1 - Ver mi saldo" );
121            pantalla.mostrarLineaMensaje( "2 - Retirar efectivo" );
122            pantalla.mostrarLineaMensaje( "3 - Depositar fondos" );
123            pantalla.mostrarLineaMensaje( "4 - Salir\n" );
124            pantalla.mostrarMensaje( "Escriba una opcion: " );
125            return teclado.obtenerEntrada(); // devuelve la opcion seleccionada por el usuario
126         } // fin del m□todo mostrarMenuPrincipal
127
128         // devuelve un objeto de la subclase especificada de Transaccion
129         private Transaccion crearTransaccion( int tipo )
130         {
131            Transaccion temp = null; // variable temporal Transaccion
132
133            // determina qu□ tipo de Transaccion crear
134            switch ( tipo )
```

```
134            switch ( tipo )
135            {
136               case SOLICITUD_SALDO: // crea una nueva transacci□n SolicitudSaldo
137                  temp = new SolicitudSaldo(
138                     numeroCuentaActual, pantalla, baseDatosBanco );
139                  break;
140               case RETIRO: // crea una nueva transacci□n Retiro
141                  temp = new Retiro( numeroCuentaActual, pantalla,
142                     baseDatosBanco, teclado, dispensadorEfectivo );
143                  break;
144               case DEPOSITO: // crea una nueva transacci□n Deposito
145                  temp = new Deposito( numeroCuentaActual, pantalla,
146                     baseDatosBanco, teclado, ranuraDeposito );
147                  break;
148            } // fin de switch
149
150            return temp; // devuelve el obejto reci□n creado
151         } // fin del m□todo crearTransaccion
152      } // fin de la clase ATM
153
```

Modulo Cuenta.java

Método Cuanta ()

```
1   package atm.d;
2
3   // Cuenta.java
4   // Represents a bank account
5
6   public class Cuenta
7   {
8      private int numeroCuenta; // n□mero de cuenta
9      private int nip; // NIP para autenticaci□n
10     private double saldoDisponible; // fondos disponibles para retirar
11     private double saldoTotal; // fondos disponibles + dep□sitos pendientes
12
13     // el constructor de Cuenta inicializa los atributos
14     public Cuenta( int elNumeroDeCuenta, int elNIP,
15        double elSaldoDisponible, double elSaldoTotal )
16     {
17        numeroCuenta = elNumeroDeCuenta;
18        nip = elNIP;
19        saldoDisponible = elSaldoDisponible;
20        saldoTotal = elSaldoTotal;
21     } // fin del constructor de Cuenta
22
23     // determina si un NIP especificado por el usuario coincide con el NIP en la Cuenta
24     public boolean validarNIP( int nipUsuario )
25     {
26        if ( nipUsuario == nip )
27           return true;
28        else
29           return false;
30     } // fin del m□todo validarNIP
31
32     // devuelve el saldo disponible
33     public double obtenerSaldoDisponible()
34     {
```

```
34      {
35          return saldoDisponible;
36      } // fin de obtenerSaldoDisponible
37
38      // devuelve el saldo total
39      public double obtenerSaldoTotal()
40      {
41          return saldoTotal;
42      } // fin del m□todo obtenerSaldoTotal
43
44      // abona un monto a la cuenta
45      public void abonar( double monto )
46      {
47          saldoTotal += monto; // lo suma al saldo total
48      } // fin del m□todo abonar
49
50      // carga un monto a la cuenta
51      public void cargar( double monto )
52      {
53          saldoDisponible -= monto; // lo resta del saldo disponible
54          saldoTotal -= monto; // lo resta del saldo total
55      } // fin del m□todo cargar
56
57      // devuelve el n□mero de cuenta
58      public int obtenerNumeroCuenta()
59      {
60          return numeroCuenta;
61      } // fin del m□todo obtenerNumeroCuenta
62  } // fin de la clase Cuenta
63
```

Modulo Deposito.java

Método Deposito ()

```java
package atm.d;
// Deposito.java
// Representa una transacci□n de dep□sito en el AT
public class Deposito extends Transaccion
{
   private double monto; // monto a depositar
   private Teclado teclado; // referencia al teclado
   private RanuraDeposito ranuraDeposito; // referencia a la ranura de dep□sito
   private final static int CANCELO = 0; // constante para la opci□n de cancelar

   // constructor de Deposito
   public Deposito( int numeroCuentaUsuario, Pantalla pantallaATM,
      BaseDatosBanco baseDatosBanco, Teclado tecladoATM,
      RanuraDeposito ranuraDepositoATM )
   {
      // inicializa las variables de la superclase
      super( numeroCuentaUsuario, pantallaATM, baseDatosBanco );

      // inicializa las referencias al teclado y la ranura de dep□sito
      teclado = tecladoATM;
      ranuraDeposito = ranuraDepositoATM;
   } // fin del constructor de Deposito

   // realiza la transacci□n
   public void ejecutar()
   {
      BaseDatosBanco baseDatosBanco = obtenerBaseDatosBanco(); // obtiene la referencia
      Pantalla pantalla = obtenerPantalla(); // obtiene la referencia

      monto = pedirMontoADepositar(); // obtiene el monto a depositar del usuario

      // comprueba si el usuario introdujo un monto a depositar o cancel□
      if ( monto != CANCELO )
      {
```

```
34          {
35             // solicita un sobre de dep□sito que contenga el monto especificado
36             pantalla.mostrarMensaje(
37                "\nInserte un sobre que contenga " );
38             pantalla.mostrarMontoDolares( monto );
39             pantalla.mostrarLineaMensaje( "." );
40
41             // recibe el sobre de dep□sito
42             boolean seRecibioSobre = ranuraDeposito.seRecibioSobre();
43
44             // comprueba si se recibi□ el sobre de dep□sito
45             if ( seRecibioSobre )
46             {
47                pantalla.mostrarLineaMensaje( "\nSe recibio su sobre de " +
48                   "deposito.\nNOTA: El dinero que acaba de depositar no " +
49                   "estara disponible sino hasta que verifiquemos el monto del " +
50                   "efectivo y cualquier cheque incluido." );
51
52                // hace un abono a la cuenta para reflejar el dep□sito
53                baseDatosBanco.abonar( obtenerNumeroCuenta(), monto );
54             } // fin de if
55             else // no se recibi□ el sobre de dep□sito
56             {
57                pantalla.mostrarLineaMensaje( "\nNo inserto un sobre de " +
58                   "deposito, por lo que el ATM ha cancelado su transaccion." );
59             } // fin de else
60          } // fin de if
61          else // el usuario cancel□ en vez de introducir el monto
62          {
63             pantalla.mostrarLineaMensaje( "\nCancelando transaccion..." );
64          } // fin de else
65       } // fin del m□todo ejecutar
66
67       // pide al usuario que introduzca un monto a depositar en centavos
```

```
67       // pide al usuario que introduzca un monto a depositar en centavos
68       private double pedirMontoADepositar()
69       {
70          Pantalla pantalla = obtenerPantalla(); // obtiene referencia a la pantalla
71
72          // muestra el indicador
73          pantalla.mostrarMensaje( "\nIntroduzca un monto a depositar en " +
74             "CENTAVOS (o 0 para cancelar): " );
75          int entrada = teclado.obtenerEntrada(); // recibe la entrada del monto de dep□sito
76
77          // comprueba si el usuario cancel□ o introdujo un monto v□lido
78          if ( entrada == CANCELO )
79             return CANCELO;
80          else
81          {
82             return ( double ) entrada / 100; // devuelve el monto en d□lares
83          } // fin de else
84       } // fin del m□todo pedirMontoADepositar
85    } // fin de la clase Deposito
86
87
88
```

Modulo DispensadorEfectivo.java

Método DispensadorEfectivo ()

```java
package atm.d;
// DispensadorEfectivo.java
// Representa al dispensador de efectivo del AT
public class DispensadorEfectivo
{
   // el n□mero inicial predeterminado de billetes en el dispensador de efectivo
   private final static int CUENTA_INICIAL = 500;
   private int cuenta; // n□mero restante de billetes de $20

   // el constructor sin argumentos de DispensadorEfectivo inicializa cuenta con el valor predeterminado
   public DispensadorEfectivo()
   {
      cuenta = CUENTA_INICIAL; // establece el atributo cuenta al valor predeterminado
   } // fin del constructor de DispensadorEfectivo

   // simula la acci□n de dispensar el monto especificado de efectivo
   public void dispensarEfectivo( int monto )
   {
      int billetesRequeridos = monto / 20; // n□mero de billetes de $20 requeridos
      cuenta -= billetesRequeridos; // actualiza la cuenta de billetes
   } // fin del m□todo dispensarEfectivo

   // indica si el dispensador de efectivo puede dispensar el monto deseado
   public boolean haySuficienteEfectivoDisponible( int monto )
   {
      int billetesRequeridos = monto / 20; // n□mero de billetes de $20 requeridos

      if ( cuenta >= billetesRequeridos  )
         return true; // hay suficientes billetes disponibles
      else
         return false; // no hay suficientes billetes disponibles
   } // fin del m□todo haySuficienteEfectivoDisponible
} // fin de la clase DispensadorEfectivo
```

Modulo EjemploPracticoATM.java

Método EjemploPracticoATM ()

```java
package atm.d;


import atm.d.ATM;

// EjemploPracticoATM.java
// Programa controlador para el ejemplo pr□ctico del ATM

public class EjemploPracticoATM
{
   // el m□todo main crea y ejecuta el ATM
   public static void main( String[] args )
   {
      ATM elATM = new ATM();
      elATM.run();
   } // fin de main
} // fin de la clase EjemploPracticoATM


```

Modulo Pantalla.java

Método Pantalla ()

```java
package atm.d;

// Pantalla.java
// Representa a la pantalla del ATM

public class Pantalla
{
   // muestra un mensaje sin un retorno de carro
   public void mostrarMensaje( String mensaje )
   {
      System.out.print( mensaje );
   } // fin del m□todo mostrarMensaje

   // muestra un mensaje con un retorno de carro
   public void mostrarLineaMensaje( String mensaje )
   {
      System.out.println( mensaje );
   } // fin del m□todo mostrarLineaMensaje

   // muestra un monto en d□lares
   public void mostrarMontoDolares( double monto )
   {
      System.out.printf( "$%,.2f", monto );
   } // fin del m□todo mostrarMontoDolares
} // fin de la clase Pantalla
```

Modulo RanuraDeposito.java

Método RanuraDeposito ()

```java
package atm.d;

// RanuraDeposito.java
// Represents the deposit slot of the ATM

public class RanuraDeposito
{
   // indica si se recibi□ el sobre (siempre devuelve true, ya que □sta
   // es s□lo una simulaci□n de software de una ranura de dep□sito real)
   public boolean seRecibioSobre()
   {
      return true; // se recibi□ el sobre
   } // fin del m□todo seRecibioSobre
} // fin de la clase RanuraDeposito
```

Modulo Retiro.java

Método Retiro ()

```
1   package atm.d;
2
3
4   import atm.d.Pantalla;
5
6   // Retiro.java
7   // Representa una transacci□n de retiro en el ATM
8
9   public class Retiro extends Transaccion
10  {
11     private int monto; // monto a retirar
12     private Teclado teclado; // referencia al teclado
13     private DispensadorEfectivo dispensadorEfectivo; // referencia al dispensador de efe
14
15     // constante que corresponde a la opci□n del men□ a cancelar
16     private final static int CANCELO = 6;
17
18     // constructor de Retiro
19     public Retiro( int numeroCuentaUsuario, Pantalla pantallaATM,
20        BaseDatosBanco baseDatosBanco, Teclado tecladoATM,
21        DispensadorEfectivo dispensadorEfectivoATM )
22     {
23        // inicializa las variables de la superclase
24        super( numeroCuentaUsuario, pantallaATM, baseDatosBanco );
25
26        // inicializa las referencias al teclado y al dispensador de efectivo
27        teclado = tecladoATM;
28        dispensadorEfectivo = dispensadorEfectivoATM;
29     } // fin del constructor de Retiro
30
31     // realiza la transacci□n
32     public void ejecutar()
33     {
34        boolean efectivoDispensado = false; // no se ha dispensado a□n el efectivo
```

```
34          boolean efectivoDispensado = false; // no se ha dispensado a□n el efectivo
35          double saldoDisponible; // monto disponible para retirar
36
37          // obtiene referencias a la base de datos del banco y la pantalla
38          BaseDatosBanco baseDatosBanco = obtenerBaseDatosBanco();
39          Pantalla pantalla = obtenerPantalla();
40
41          // itera hasta que se dispense el efectivo o que cancele el usuario
42          do
43          {
44             // obtiene un monto de retiro elegido por el usuario
45             monto = mostrarMenuDeMontos();
46
47             // comprueba si el usuario eligi□ un monto de retiro o si cancel□
48             if ( monto != CANCELO )
49             {
50                // obtiene el saldo disponible de la cuenta implicada
51                saldoDisponible =
52                   baseDatosBanco.obtenerSaldoDisponible( obtenerNumeroCuenta() );
53
54                // comprueba si el usuario tiene suficiente dinero en la cuenta
55                if ( monto <= saldoDisponible )
56                {
57                   // comprueba si el dispensador de efectivo tiene suficiente dinero
58                   if ( dispensadorEfectivo.haySuficienteEfectivoDisponible( monto ) )
59                   {
60                      // actualiza la cuenta implicada para reflejar el saldo
61                      baseDatosBanco.cargar( obtenerNumeroCuenta(), monto );
62
63                      dispensadorEfectivo.dispensarEfectivo( monto ); // dispensar efectivo
64                      efectivoDispensado = true; // se dispens□ el efectivo
65
66                      // instruye al usuario que tome efectivo
67                      pantalla.mostrarLineaMensaje(
```

```
67                      pantalla.mostrarLineaMensaje(
68                         "\nTome ahora su efectivo." );
69                   } // fin de if
70                   else // el dispensador no tiene suficiente efectivo
71                      pantalla.mostrarLineaMensaje(
72                         "\nNo hay suficiente efectivo disponible en el ATM." +
73                         "\n\nSeleccione un monto menor." );
74                } // fin de if
75                else // no hay suficiente dinero disponible en la cuenta del usuario
76                {
77                   pantalla.mostrarLineaMensaje(
78                      "\nNo hay suficientes fondos en su cuenta." +
79                      "\n\nSeleccione un monto menor." );
80                } // fin de else
81             } // fin de if
82             else // el usuario eligi□ la opci□n cancelar del men□
83             {
84                pantalla.mostrarLineaMensaje( "\nCancelando transaccion..." );
85                return; // regresa al men□ principal porque el usuario cancel□
86             } // fin de else
87          } while ( !efectivoDispensado );
88
89       } // fin del m□todo ejecutar
90
91       // muestra un men□ de montos de retiro y la opci□n para cancelar;
92       // devuelve el monto elegido o 0 si el usuario elije cancelar
93       private int mostrarMenuDeMontos()
94       {
95          int opcionUsuario = 0; // variable local para almacenar el valor de retorno
96
97          Pantalla pantalla = obtenerPantalla(); // obtiene referencia a la pantalla
98
99          // arreglo de montos que corresponde a los n□meros del men□
100         int montos[] = { 0, 20, 40, 60, 100, 200 };
```

```
100         int montos[] = { 0, 20, 40, 60, 100, 200 };
101
102         // itera mientras no se haya elegido una opci□n v□lida
103         while ( opcionUsuario == 0 )
104         {
105            // muestra el men□
106            pantalla.mostrarLineaMensaje( "\nMenu de retiro:" );
107            pantalla.mostrarLineaMensaje( "1 - $20" );
108            pantalla.mostrarLineaMensaje( "2 - $40" );
109            pantalla.mostrarLineaMensaje( "3 - $60" );
110            pantalla.mostrarLineaMensaje( "4 - $100" );
111            pantalla.mostrarLineaMensaje( "5 - $200" );
112            pantalla.mostrarLineaMensaje( "6 - Cancelar transaccion" );
113            pantalla.mostrarMensaje( "\nSeleccione un monto a retirar: " );
114
115            int entrada = teclado.obtenerEntrada(); // obtiene la entrada del usuario mediante el teclado
116
117            // determina c□mo proceder con base en el valor de la entrada
118            switch ( entrada )
119            {
120               case 1: // si el usuario eligi□ un monto de retiro
121               case 2: // (es decir, si eligi□ la opci□n 1, 2, 3, 4 o 5), devolver
122               case 3: // el monto correspondiente del arreglo montos
123               case 4:
124               case 5:
125                  opcionUsuario = montos[ entrada ]; // guarda la elecci□n del usuario
126                  break;
127               case CANCELO: // el usuario eligi□ cancelar
128                  opcionUsuario = CANCELO; // guarda la elecci□n del usuario
129                  break;
130               default: // el usuario no introdujo un valor del 1 al 6
131                  pantalla.mostrarLineaMensaje(
132                     "\nSeleccion invalida. Intente de nuevo." );
133            } // fin de switch
134         } // fin de while
135
136         return opcionUsuario; // devuelve el monto de retiro o CANCELO
137      } // fin del m□todo mostrarMenuDeMontos
138   } // fin de la clase Retiro
139
140
```

Modulo SolicitudSaldo.java

Método SolicitudSaldo ()

```java
package atm.d;

import atm.d.Pantalla;

// SolicitudSaldo.java
// Representa una transacci□n de solicitud de saldo en el ATM

public class SolicitudSaldo extends Transaccion
{
   // constructor de SolicitudSaldo
   public SolicitudSaldo( int numeroCuentaUsuario, Pantalla pantallaATM,
      BaseDatosBanco baseDatosBanco )
   {
      super( numeroCuentaUsuario, pantallaATM, baseDatosBanco );
   } // fin del constructor de SolicitudSaldo


   // realiza la transacci□n
   public void ejecutar()
   {
      // obtiene referencias a la base de datos del banco y la pantalla
      BaseDatosBanco baseDatosBanco = obtenerBaseDatosBanco();
      Pantalla pantalla = obtenerPantalla();

      // obtiene el saldo disponible para la cuenta implicada
      double saldoDisponible =
         baseDatosBanco.obtenerSaldoDisponible( obtenerNumeroCuenta() );

      // obtiene el saldo total para la cuenta implicada
      double saldoTotal =
         baseDatosBanco.obtenerSaldoTotal( obtenerNumeroCuenta() );

      // muestra la informaci□n del saldo en la pantalla
      pantalla.mostrarLineaMensaje( "\nInformacion de saldo:" );
      pantalla.mostrarMensaje( " - Saldo disponible: " );
      pantalla.mostrarMontoDolares( saldoDisponible );
      pantalla.mostrarMensaje( "\n - Saldo total:     " );
      pantalla.mostrarMontoDolares( saldoTotal );
      pantalla.mostrarLineaMensaje( "" );
   } // fin del m□todo ejecutar
} // fin de la clase SolicitudSaldo
```

Modulo Teclado.java

Método Teclado ()

```java
package atm.d;

// Teclado.java
// Representa el teclado del ATM
import java.util.Scanner; // el programa usa a Scanner para obtener la entrada del usuario

public class Teclado
{
   private Scanner entrada; // lee datos de la l□nea de comandos

   // el constructor sin argumentos inicializa el objeto Scanner
   public Teclado()
   {
      entrada = new Scanner( System.in );
   } // fin del constructor sin argumentos de Teclado

   // devuelve un valor entero introducido por el usuario
   public int obtenerEntrada()
   {
      return entrada.nextInt(); // suponemos que el usuario introduce un entero
   } // fin del m□todo obtenerEntrada
} // fin de la clase Teclado
```

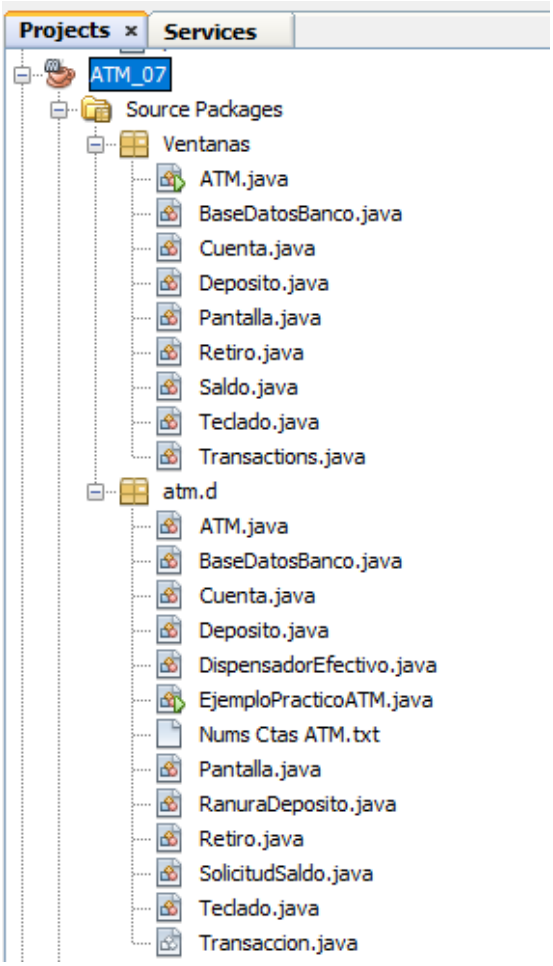
Modulo Transaccion.java

Método Transaccion ()

```java
package atm.d;


import atm.d.Pantalla;

// Transaccion.java
// La superclase abstracta Transaccion representa una transacci□n con el ATM

public abstract class Transaccion
{
   private int numeroCuenta; // indica la cuenta implicada
   private Pantalla pantalla; // pantalla del ATM
   private BaseDatosBanco baseDatosBanco; // base de datos de informaci□n de cuentas

   // el constructor de Transaccion es invocado por las subclases mediante super()
   public Transaccion( int numeroCuentaUsuario, Pantalla pantallaATM,
      BaseDatosBanco baseDatosBancoATM )
   {
      numeroCuenta = numeroCuentaUsuario;
      pantalla = pantallaATM;
      baseDatosBanco = baseDatosBancoATM;
   } // fin del constructor de Transaccion

   // devuelve el n□mero de cuenta
   public int obtenerNumeroCuenta()
   {
      return numeroCuenta;
   } // fin del m□todo obtenerNumeroCuenta

   // devuelve una referencia a la pantalla
   public Pantalla obtenerPantalla()
   {
      return pantalla;
   } // fin del m□todo obtenerPantalla

   // devuelve una referencia a la base de datos del banco
   public BaseDatosBanco obtenerBaseDatosBanco()
   {
      return baseDatosBanco;
   } // fin del m□todo obtenerBaseDatosBanco

   // realiza la transacci□n (cada subclase sobrescribe este m□todo)
   abstract public void ejecutar();
} // fin de la clase Transaccion
```

Imagen del Navegador de Proyectos con paquetes y clases

desplegados



Copias de las ventanas de las corridas de prueba

Comentarios Finales

De Santiago Landeros Edwin Uriel

El reflejo de la concepción de errores en el equipo ayuda a que tomemos nuevas decisiones, En el tiempo que llevamos desarrollando el informe del proyecto el Profesor nos demuestra cómo se relacionan las clases, el rol de estas es de suma importancia entender debido al flujo que sigue y que no entendíamos en un principio, era fundamental trabajar por consolidar y destacar su importancia en lo que se refiere a la tarea del cajero automático.

Para mi todo fue muy importante, ya que gracias a esto nos dimos cuenta de todo lo que usted profesor realiza/comenta durante de trabajo en clase no solo de es como es realizar el informe sino de las cosas que hacen los alumnos en las que nos podemos equivocar previniéndonos de errores y entendiendo el por qué pasa.

Reveles Ramírez Jesús Emilio

En esta ocasión, buscamos ser un poco capaz de motivarnos, no era un trabajo sencillo, pero agradezco el apoyo del equipo en ayudarnos a comprender este logro, lograr objetivos conceptuales, procedimentales y actitudinales definidos nos ayudó a nosotros a que adquiramos estrategias que nos permitan continuar un aprendizaje. Sin embargo, equivocándonos aprendimos de los errores de compilación, modificaciones y validaciones que en un principio se modificaba en tiempo de ejecución.

Por último, la planificación didáctica que nos proporciona las pruebas y errores que nos enseñó el profesor nos ayudó en nuevos principios que no teníamos definidos y buscar desarrollar nuevas estrategias para nosotros en la realización del proyecto.

Villegas Nolasco Elide Yolotzin

Conforme fuimos realizando este proyecto nos fuimos percatando de muchas cosas que antes no habíamos considerado. Pudimos percatarnos como lo hemos venido mencionando de la importancia de saber qué es lo que tenía que hacer el cajero, pero también pudimos detectar algunos puntos clave para afianzar muchos

procesos dentro de los métodos y clases, tener una visión más clara de la funcionalidad y saber que existen situaciones reales para poder resolver problemas o tomar cierto tipo de decisiones de acuerdo a lo que el usuario decide hacer. Llevar a cabo un análisis detallado como el que se realizó en este proyecto incrementa en gran proporción las probabilidades de tener éxito ya que de ante mano se conoce lo que se quiere lograr y cómo se va a hacer para lograrlo.

Herrera Cano Edwin Rubén

Las ideas centrales del proceso del cajero es identificar los principales problemas, que el cliente puede tener al iniciar sesión, como validaciones y la cantidad de opciones en donde el usuario puede navegar, la planificación y el ordenamiento de los procesos del cajero es un desafío, debido que se debe promover un flujo donde el usuario pueda tener una libre y entendible fluidez al navegar en el cajero.

Nuevos conceptos y ejemplos proporcionados por el profesor nos ayudan a buscar como corregir los errores presentados desarrollar nuevas estrategias para nosotros dio como resultado en ponernos de acuerdo en cómo poder llegar a una solución de esta tarea como el ¿que realizar? el profesor permanentemente desde sus conocimientos nos apoyó para los objetivos determinados del cajero.

Palmieri Mondragón Gerardo

Personalmente me enfoque en una Estrategia, Organizar el espacio, donde utilizamos otras herramientas de trabajo para poner en orden en la realización del trabajo, Adaptando nuestros tiempos a las actividades del proyecto del cajero automático. Cree un espacio de conocimiento compartido donde nos ayudábamos como equipo a organizarnos para Facilitar la interacción y composición del proyecto decidiendo las versiones y los avances que íbamos teniendo.

Imaginamos cómo funcionaria el proyecto simulando depósitos o retiros, es un proceso mental que se realizado gracias al profesor y un resultado de ese proceso, es que el proyecto del cajero automático es un producto comunicable y analizable.