Progettazione Orientata agli Oggetti: Sportello Bancomat

Sportello Bancomat - Obiettivo

- Vogliamo progettare e implementare un semplice sistema che simula il funzionamento di un tipico sportello bancomat con operazioni di accesso conto, e operazioni di prelievo, versamento e lettura saldo.

- Il sistema deve avere un'interfaccia grafica che riproduca la tastiera e il display di un tipico sportello bancomat

Sportello Bancomat -- Requisiti

- Uno Sportello Bancomat è utilizzato dai clienti di una banca
- Un cliente ha
  - un conto corrente,
  - un libretto di deposito,
  - un codice cliente
  - un PIN

Sportello Bancomat -- Requisiti

- Un cliente
  - accede al sistema e
  - seleziona un conto
- Il saldo del conto viene visualizzato
- Il cliente può scegliere di depositare o prelevare contante
- Il processo si ripete finché il cliente non decide di terminare

Sportello Bancomat -- Requisiti

- Interfaccia grafica utente:
  - Keypad
  - Display
  - Pulsanti A, B, C
  - La funzione dei pulsanti dipende dallo stato dello sportello

Sportello Bancomat -- Requisiti

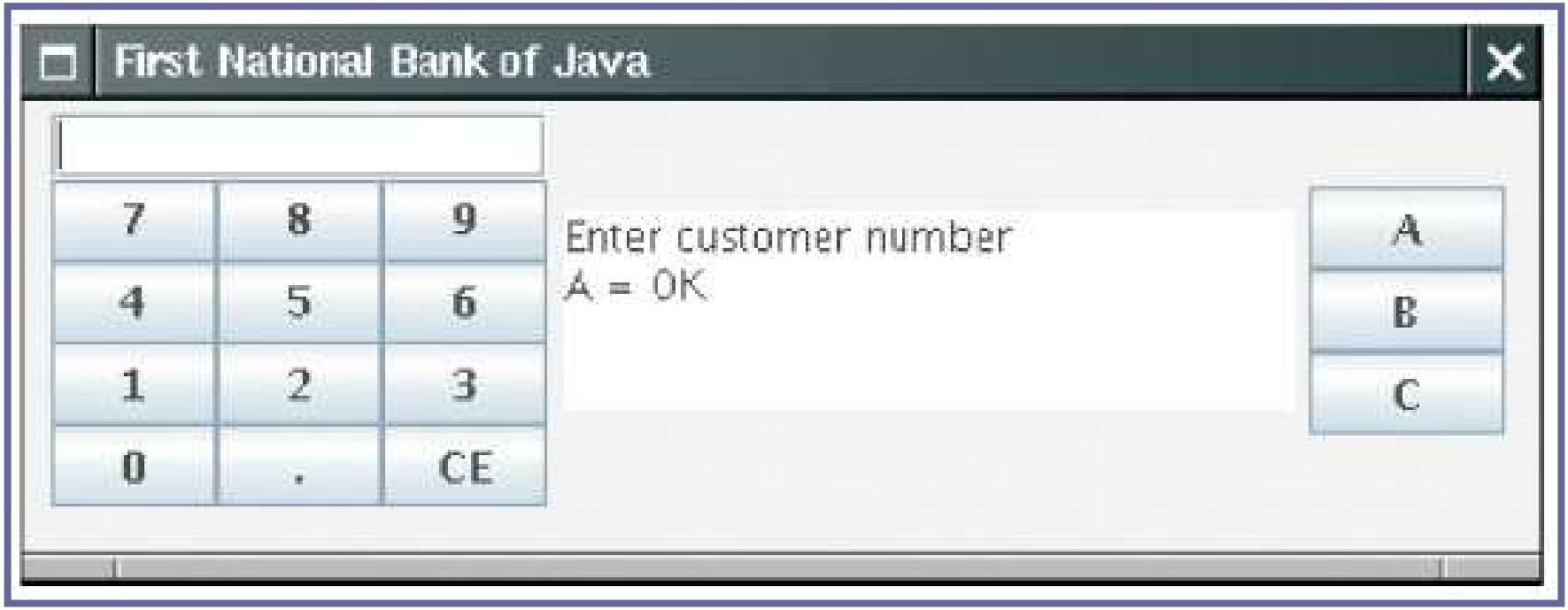


Figure 12:
User Interface of the Automatic Teller Machine

Sportello Bancomat -- Requisiti

- All'inizio il cliente deve:
  - Inserire il codice cliente
  - Pressare il pulsante A
  - Il display deve mostrare

```
Inserire il codice cliente
A = OK
```

Sportello Bancomat -- Requisiti

- Quindi, il cliente deve:
  - Inserire il PIN
  - Pressare il pulsante A
  - Il display deve mostrare

```
Iserire il PIN
A = OK
```

Sportello Bancomat -- Requisiti

- Ricercare il codice cliente e controllare se il PIN coincide
  - Se si trova un cliente della banca con i dati inseriti si procede
  - Altrimenti si deve tornare al menu di partenza

Sportello Bancomat -- Requisiti

- Se il cliente ha eseguito correttamente l'accesso
- Il display deve mostrare

```
Seleziona una scelta
A = Checking
B = Savings
C = Exit
```

Sportello Bancomat -- Requisiti

- Se il cliente sceglie C
  - Lo sportello torna al menu di partenza
- Se il cliente sceglie A o B
  - Lo sportello memorizza la scelta effettuata
  - Il display mostra:

```
Saldo = <saldo del conto selezionato>
Inserire import e operazione da effettuare
A = Prelevamento
B = Versamento
C = Annulla
```

Sportello Bancomat -- Requisiti

- Se il cliente seleziona A o B
  - L'importo inserito è prelevato o versato
  - Quindi, lo sportello torna nello stato precedente
- Se il cliente seleziona C
  - Lo sportello torna nello stato precedente

Sportello Bancomat -- Requisiti

- Possibili classi (nomi utilizzati nella descrizione)

```
ATM
User
Keypad
Display
Display message
Button
State
Bank account
Checking account
Savings account
Customer
Customer number
PIN
Bank
```

Sportello Bancomat -- Schede CRC

Customer
get accounts
match number and PIN

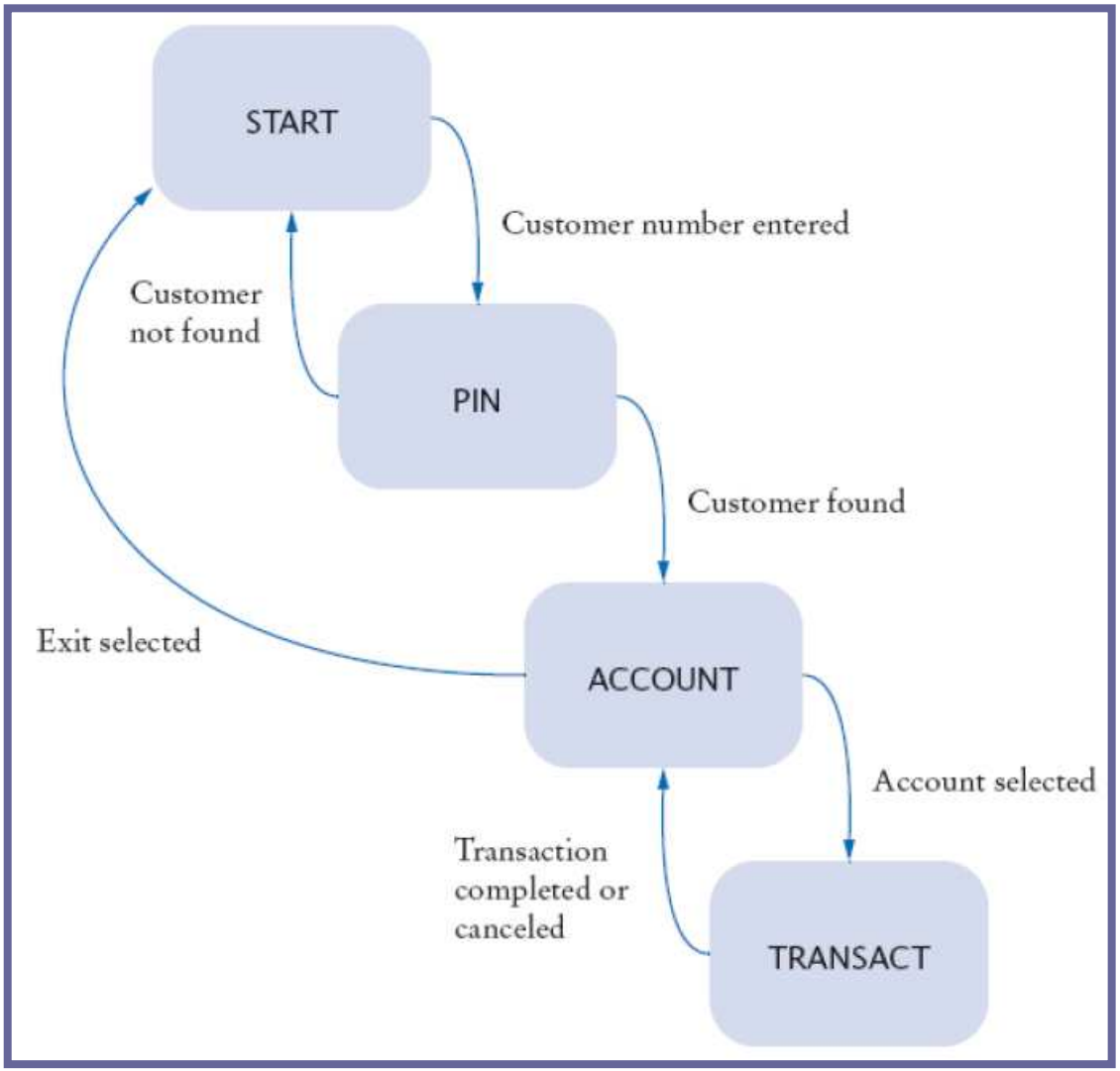
Bank	
find customer	Customer
read customers	

ATM	
manage state	Customer
select customer	Bank
select account	BankAccount
execute transaction	

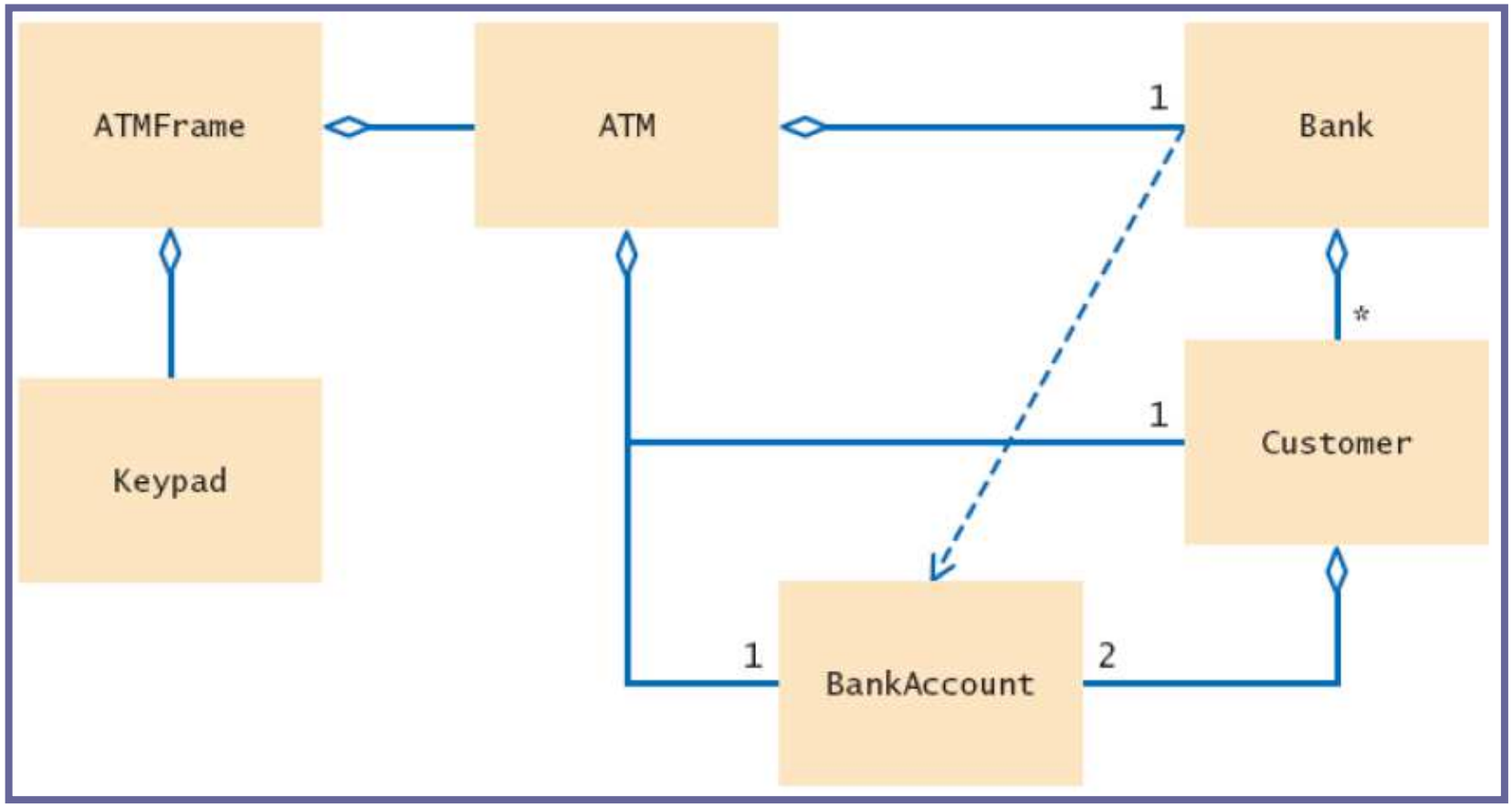
Stati dello sportello

- START: inserire il codice cliente
- PIN: Inserire il PIN
- ACCOUNT: Seleziona il conto
- TRANSACT: Seleziona la transazione

Diagramma di stato classe ATM



Sportello Bancomat -- Diagramma UML



Documentazione classe ATM

```
/**
   An ATM that accesses a bank.
*/
public class ATM
{
   /**
      Constructs an ATM for a given bank.
      @param aBank the bank to which this ATM connects
   */
   public ATM(Bank aBank) { }

   /**
      Sets the current customer number and sets state to PIN.
      (Precondition: state is START)
      @param number the customer number
   */
   public void setCustomerNumber(int number) { }
```

Documentazione classe ATM

```
/**
   Finds customer in bank. If found sets state to ACCOUNT,
      // else to START.
   (Precondition: state is PIN)
   @param pin the PIN of the current customer
*/
public void selectCustomer(int pin) { }

/**
   Sets current account to checking or savings. Sets state
      // to TRANSACT.
   (Precondition: state is ACCOUNT or TRANSACT)
   @param account one of CHECKING or SAVINGS
*/
public void selectAccount(int account) { }
```

Continued…

Documentazione classe ATM

```
   /**
      Withdraws amount from current account.
      (Precondition: state is TRANSACT)
      @param value the amount to withdraw
   */
   public void withdraw(double value) { }
   . . .
}
```

Sportello Bancomat – Implementazione

- Inizia l'implementazione con classi che non dipendono dalle altre
  - Keypad
  - BankAccount
- Quindi implementa `Customer` che dipende da `BankAccount`
- Questo approccio bottom-up consente di testare le classi individualmente

Sportello Bancomat – Implementazione

- Le classi aggregate nel diagramma UML danno le variabili di istanza

```
private Bank theBank;
```

- Dalla descrizione degli stati dell'ATM, si ha che abbiamo bisogno di un'altra variabile d'istanza:

```
private int state;
private Customer currentCustomer;
private BankAccount currentAccount;
```

Sportello Bancomat – Implementazione

- La maggior parte dei metodi sono facili da implementare
- Consideriamo `selectCustomer`:

```
/**
   Finds customer in bank. If found sets state to ACCOUNT,
      // else to START.
   (Precondition: state is PIN)
   @param pin the PIN of the current customer
*/
```

Sportello Bancomat – Implementazione

- La descrizione può essere tradotta facilmente in istruzioni Java:

```
public void selectCustomer(int pin)
{
   assert state == PIN;
   currentCustomer = theBank.findCustomer(customerNumber, pin);
   if (currentCustomer == null)
      state = START;
   else
      state = ACCOUNT;
}
```

File ATM.java

```
001: import java.io.IOException;
002:
003: /**
004:    An ATM that accesses a bank.
005: */
006: public class ATM
007: {
008:    /**
009:       Constructs an ATM for a given bank.
010:       @param aBank the bank to which this ATM connects
011:    */
012:    public ATM(Bank aBank)
013:    {
014:       theBank = aBank;
015:       reset();
016:    }
017:
```

Continued...

File ATM.java

```
018:    /**
019:       Resets the ATM to the initial state.
020:    */
021:    public void reset()
022:    {
023:       customerNumber = -1;
024:       currentAccount = null;
025:       state = START;
026:    }
027:
028:    /**
029:       Sets the current customer number
030:       and sets state to PIN.
031:       (Precondition: state is START)
032:       @param number the customer number.
033:    */
```

Continued...

File ATM.java

```
034:    public void setCustomerNumber(int number)
035:    {
036:       assert state == START;
037:       customerNumber = number;
038:       state = PIN;
039:    }
040:
041:    /**
042:       Finds customer in bank.
043:       If found sets state to ACCOUNT, else to START.
044:       (Precondition: state is PIN)
045:       @param pin the PIN of the current customer
046:    */
047:    public void selectCustomer(int pin)
048:    {
049:       assert state == PIN;
```

Continued...

File ATM.java

```
050:          currentCustomer
051:             = theBank.findCustomer(customerNumber, pin);
052:          if (currentCustomer == null)
053:             state = START;
054:          else
055:             state = ACCOUNT;
056:       }
057:
058:       /**
059:          Sets current account to checking or savings. Sets
060:          state to TRANSACT.
061:          (Precondition: state is ACCOUNT or TRANSACT)
062:          @param account one of CHECKING or SAVINGS
063:       */
064:       public void selectAccount(int account)
065:       {
```

Continued...

File ATM.java

```
066:        assert state == ACCOUNT || state == TRANSACT;
067:        if (account == CHECKING)
068:           currentAccount = currentCustomer.getCheckingAccount();
069:        else
070:           currentAccount = currentCustomer.getSavingsAccount();
071:        state = TRANSACT;
072:     }
073:
074:     /**
075:        Withdraws amount from current account.
076:        (Precondition: state is TRANSACT)
077:        @param value the amount to withdraw
078:     */
079:     public void withdraw(double value)
080:     {
081:        assert state == TRANSACT;
082:        currentAccount.withdraw(value);
083:     }
```

Continued...

File ATM.java

```
084:
085:    /**
086:       Deposits amount to current account.
087:       (Precondition: state is TRANSACT)
088:       @param value the amount to deposit
089:    */
090:    public void deposit(double value)
091:    {
092:       assert state == TRANSACT;
093:       currentAccount.deposit(value);
094:    }
095:
096:    /**
097:       Gets the balance of the current account.
098:       (Precondition: state is TRANSACT)
099:       @return the balance
100:    */
```

Continued...

File ATM.java

```
101:    public double getBalance()
102:    {
103:       assert state == TRANSACT;
104:       return currentAccount.getBalance();
105:    }
106:
107:    /**
108:       Moves back to the previous state.
109:    */
110:    public void back()
111:    {
112:       if (state == TRANSACT)
113:          state = ACCOUNT;
114:       else if (state == ACCOUNT)
115:          state = PIN;
116:       else if (state == PIN)
117:          state = START;
118:    }
```

Continued...

File ATM.java

```
119:
120:    /**
121:       Gets the current state of this ATM.
122:       @return the current state
123:    */
124:    public int getState()
125:    {
126:       return state;
127:    }
128:
129:    private int state;
130:    private int customerNumber;
131:    private Customer currentCustomer;
132:    private BankAccount currentAccount;
133:    private Bank theBank;
134:
```

Continued...

File ATM.java

```
135:    public static final int START = 1;
136:    public static final int PIN = 2;
137:    public static final int ACCOUNT = 3;
138:    public static final int TRANSACT = 4;
139:
140:    public static final int CHECKING = 1;
141:    public static final int SAVINGS = 2;
142: }
```

File Bank.java

```
01: import java.io.BufferedReader;
02: import java.io.FileReader;
03: import java.io.IOException;
04: import java.util.ArrayList;
05: import java.util.Scanner;
06:
07: /**
08:    A bank contains customers with bank accounts.
09: */
10: public class Bank
11: {
12:    /**
13:       Constructs a bank with no customers.
14:    */
15:    public Bank()
16:    {
17:       customers = new ArrayList<Customer>();
18:    }
```

Continued...

File Bank.java

```
19:
20:    /**
21:       Reads the customer numbers and pins
22:       and initializes the bank accounts.
23:       @param filename the name of the customer file
24:    */
25:    public void readCustomers(String filename)
26:          throws IOException
27:    {
28:       Scanner in = new Scanner(new FileReader(filename));
29:       boolean done = false;
30:       while (in.hasNext())
31:       {
32:          int number = in.nextInt();
33:          int pin = in.nextInt();
34:          Customer c = new Customer(number, pin);
35:          addCustomer(c);
36:       }
```

Continued...

File Bank.java

```
37:          in.close();
38:       }
39:
40:       /**
41:          Adds a customer to the bank.
42:          @param c the customer to add
43:       */
44:       public void addCustomer(Customer c)
45:       {
46:          customers.add(c);
47:       }
48:
49:       /**
50:          Finds a customer in the bank.
51:          @param aNumber a customer number
52:          @param aPin a personal identification number
```

Continued...

File Bank.java

```
53:          @return the matching customer, or null if no customer
54:          matches
55:       */
56:       public Customer findCustomer(int aNumber, int aPin)
57:       {
58:          for (Customer c : customers)
59:          {
60:             if (c.match(aNumber, aPin))
61:                return c;
62:          }
63:          return null;
64:       }
65:
66:       private ArrayList<Customer> customers;
67: }
68:
69:
```

Continued...

File Customer.java

```
01: /**
02:    A bank customer with a checking and a savings account.
03: */
04: public class Customer
05: {
06:    /**
07:       Constructs a customer with a given number and PIN.
08:       @param aNumber the customer number
09:       @param aPin the personal identification number
10:    */
11:    public Customer(int aNumber, int aPin)
12:    {
13:       customerNumber = aNumber;
14:       pin = aPin;
15:       checkingAccount = new BankAccount();
16:       savingsAccount = new BankAccount();
17:    }
18:
```

Continued...

File Customer.java

```
19:    /**
20:       Tests if this customer matches a customer number
21:       and PIN.
22:       @param aNumber a customer number
23:       @param aPin a personal identification number
24:       @return true if the customer number and PIN match
25:    */
26:    public boolean match(int aNumber, int aPin)
27:    {
28:       return customerNumber == aNumber && pin == aPin;
29:    }
30:
31:    /**
32:       Gets the checking account of this customer.
33:       @return the checking account
34:    */
35:    public BankAccount getCheckingAccount()
36:    {
```

Continued...

File Customer.java

```
37:          return checkingAccount;
38:       }
39:
40:       /**
41:          Gets the savings account of this customer.
42:          @return the checking account
43:       */
44:       public BankAccount getSavingsAccount()
45:       {
46:          return savingsAccount;
47:       }
48:
49:       private int customerNumber;
50:       private int pin;
51:       private BankAccount checkingAccount;
52:       private BankAccount savingsAccount;
53: }
```

Continued...

File ATMViewer.java

```
01: import java.io.IOException;
02: import javax.swing.JFrame;
03: import javax.swing.JOptionPane;
04:
05: /**
06:    A graphical simulation of an automatic teller machine.
07: */
08: public class ATMViewer
09: {
10:    public static void main(String[] args)
11:    {
12:       ATM theATM;
13:
14:       try
15:       {
16:          Bank theBank = new Bank();
17:          theBank.readCustomers("customers.txt");
```

Continued...

File ATMViewer.java

```
18:             theATM = new ATM(theBank);
19:          }
20:          catch(IOException e)
21:          {
22:             JOptionPane.showMessageDialog(null,
23:                   "Error opening accounts file.");
24:             return;
25:          }
26:
27:          JFrame frame = new ATMFrame(theATM);
28:          frame.setTitle("First National Bank of Java");
29:          frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
30:          frame.setVisible(true);
31:       }
32: }
33:
```

File ATMFrame.java

```
001: import java.awt.FlowLayout;
002: import java.awt.GridLayout;
003: import java.awt.event.ActionEvent;
004: import java.awt.event.ActionListener;
005: import javax.swing.JButton;
006: import javax.swing.JFrame;
007: import javax.swing.JPanel;
008: import javax.swing.JTextArea;
009:
010: /**
011:    A frame displaying the components of an ATM.
012: */
013: public class ATMFrame extends JFrame
014: {
015:    /**
016:       Constructs the user interface of the ATM frame.
017:    */
```

Continued…

File ATMFrame.java

```
018:    public ATMFrame(ATM anATM)
019:    {
020:       theATM = anATM;
021:
022:       // Construct components
023:       pad = new KeyPad();
024:
025:       display = new JTextArea(4, 20);
026:
027:       aButton = new JButton("  A  ");
028:       aButton.addActionListener(new AButtonListener());
029:
030:       bButton = new JButton("  B  ");
031:       bButton.addActionListener(new BButtonListener());
032:
033:       cButton = new JButton("  C  ");
034:       cButton.addActionListener(new CButtonListener());
```

Continued…

File ATMFrame.java

```
035:
036:       // Add components
037:
038:       JPanel buttonPanel = new JPanel();
039:       buttonPanel.setLayout(new GridLayout(3, 1));
040:       buttonPanel.add(aButton);
041:       buttonPanel.add(bButton);
042:       buttonPanel.add(cButton);
043:
044:       setLayout(new FlowLayout());
045:       add(pad);
046:       add(display);
047:       add(buttonPanel);
048:       showState();
049:
050:       setSize(FRAME_WIDTH, FRAME_HEIGHT);
051:    }
```

Continued...

File ATMFrame.java

```
052:
053:    /**
054:       Updates display message.
055:    */
056:    public void showState()
057:    {
058:       int state = theATM.getState();
059:       pad.clear();
060:       if (state == ATM.START)
061:          display.setText("Enter customer number\nA = OK");
062:       else if (state == ATM.PIN)
063:          display.setText("Enter PIN\nA = OK");
064:       else if (state == ATM.ACCOUNT)
065:          display.setText("Select Account\n"
066:                + "A = Checking\nB = Savings\nC = Exit");
067:       else if (state == ATM.TRANSACT)
068:          display.setText("Balance = "
```

Continued...

File ATMFrame.java

```
069:                   + theATM.getBalance()
070:                   + "\nEnter amount and select transaction\n"
071:                   + "A = Withdraw\nB = Deposit\nC = Cancel");
072:    }
073:
074:    private class AButtonListener implements ActionListener
075:    {
076:       public void actionPerformed(ActionEvent event)
077:       {
078:          int state = theATM.getState();
079:          if (state == ATM.START)
080:             theATM.setCustomerNumber((int) pad.getValue());
081:          else if (state == ATM.PIN)
082:             theATM.selectCustomer((int) pad.getValue());
083:          else if (state == ATM.ACCOUNT)
084:             theATM.selectAccount(ATM.CHECKING);
085:          else if (state == ATM.TRANSACT)
```

Continued...

File ATMFrame.java

```
086:            {
087:               theATM.withdraw(pad.getValue());
088:               theATM.back();
089:            }
090:            showState();
091:         }
092:      }
093:
094:      private class BButtonListener implements ActionListener
095:      {
096:         public void actionPerformed(ActionEvent event)
097:         {
098:            int state = theATM.getState();
099:            if (state == ATM.ACCOUNT)
100:               theATM.selectAccount(ATM.SAVINGS);
101:            else if (state == ATM.TRANSACT)
```

Continued…

File ATMFrame.java

```
102:            {
103:               theATM.deposit(pad.getValue());
104:               theATM.back();
105:            }
106:            showState();
107:         }
108:      }
109:
110:      private class CButtonListener implements ActionListener
111:      {
112:         public void actionPerformed(ActionEvent event)
113:         {
114:            int state = theATM.getState();
115:            if (state == ATM.ACCOUNT)
116:               theATM.reset();
```

Continued…

File ATMFrame.java

```
117:             else if (state == ATM.TRANSACT)
118:                theATM.back();
119:             showState();
120:          }
121:       }
122:
123:       private JButton aButton;
124:       private JButton bButton;
125:       private JButton cButton;
126:
127:       private KeyPad pad;
128:       private JTextArea display;
129:
130:       private ATM theATM;
131:
132:       private static final int FRAME_WIDTH = 300;
133:       private static final int FRAME_HEIGHT = 400;
134: }
```

File KeyPad.java

```
001: import java.awt.BorderLayout;
002: import java.awt.GridLayout;
003: import java.awt.event.ActionEvent;
004: import java.awt.event.ActionListener;
005: import javax.swing.JButton;
006: import javax.swing.JPanel;
007: import javax.swing.JTextField;
008:
009: /**
010:    A component that lets the user enter a number, using
011:    a button pad labeled with digits.
012: */
013: public class KeyPad extends JPanel
014: {
015:    /**
016:       Constructs the keypad panel.
017:    */
```

Continued...

File KeyPad.java

```
018:    public KeyPad()
019:    {
020:       setLayout(new BorderLayout());
021:
022:       // Add display field
023:
024:       display = new JTextField();
025:       add(display, "North");
026:
027:       // Make button panel
028:
029:       buttonPanel = new JPanel();
030:       buttonPanel.setLayout(new GridLayout(4, 3));
031:
032:       // Add digit buttons
033:
```

Continued...

File KeyPad.java

```
034:         addButton("7");
035:         addButton("8");
036:         addButton("9");
037:         addButton("4");
038:         addButton("5");
039:         addButton("6");
040:         addButton("1");
041:         addButton("2");
042:         addButton("3");
043:         addButton("0");
044:         addButton(".");
045:
046:         // Add clear entry button
047:
048:         clearButton = new JButton("CE");
049:         buttonPanel.add(clearButton);
050:
```

Continued...

File KeyPad.java

```
051:         class ClearButtonListener implements ActionListener
052:         {
053:            public void actionPerformed(ActionEvent event)
054:            {
055:               display.setText("");
056:            }
057:         }
058:         ActionListener listener = new ClearButtonListener();
059:
060:         clearButton.addActionListener(new
061:               ClearButtonListener());
062:
063:         add(buttonPanel, "Center");
064:      }
065:
```

Continued...

File KeyPad.java

```
066:    /**
067:       Adds a button to the button panel
068:       @param label the button label
069:    */
070:    private void addButton(final String label)
071:    {
072:       class DigitButtonListener implements ActionListener
073:       {
074:          public void actionPerformed(ActionEvent event)
075:          {
076:
077:             // Don't add two decimal points
078:             if (label.equals(".")
079:                   && display.getText().indexOf(".") != -1)
080:                return;
081:
082:             // Append label text to button
```

Continued...

File KeyPad.java

```
083:             display.setText(display.getText() + label);
084:          }
085:       }
086:
087:       JButton button = new JButton(label);
088:       buttonPanel.add(button);
089:       ActionListener listener = new DigitButtonListener();
090:       button.addActionListener(listener);
091:    }
092:
093:    /**
094:       Gets the value that the user entered.
095:       @return the value in the text field of the keypad
096:    */
097:    public double getValue()
098:    {
099:       return Double.parseDouble(display.getText());
100:    }
```

Continued…

File KeyPad.java

```
101:
102:    /**
103:       Clears the display.
104:    */
105:    public void clear()
106:    {
107:       display.setText("");
108:    }
109:
110:    private JPanel buttonPanel;
111:    private JButton clearButton;
112:    private JTextField display;
113: }
114:
```

File ATMTester.java

```
01: import java.io.IOException;
02: import java.util.Scanner;
03:
04: /**
05:    A text-based simulation of an automatic teller machine.
06: */
07: public class ATMTester
08: {
09:    public static void main(String[] args)
10:    {
11:       ATM theATM;
12:       try
13:       {
14:          Bank theBank = new Bank();
15:          theBank.readCustomers("customers.txt");
16:          theATM = new ATM(theBank);
17:       }
```

Continued...

File ATMTester.java

```
18:         catch(IOException e)
19:         {
20:            System.out.println("Error opening accounts file.");
21:            return;
22:         }
23:
24:         Scanner in = new Scanner(System.in);
25:
26:         while (true)
27:         {
28:            int state = theATM.getState();
29:            if (state == ATM.START)
30:            {
31:               System.out.print("Enter account number: ");
32:               int number = in.nextInt();
33:               theATM.setCustomerNumber(number);
34:            }
```

Continued...

File ATMTester.java

```
35:             else if (state == ATM.PIN)
36:             {
37:                System.out.print("Enter PIN: ");
38:                int pin = in.nextInt();
39:                theATM.selectCustomer(pin);
40:             }
41:             else if (state == ATM.ACCOUNT)
42:             {
43:                System.out.print("A=Checking, B=Savings, C=Quit:
44:                String command = in.next();
45:                if (command.equalsIgnoreCase("A"))
46:                   theATM.selectAccount(ATM.CHECKING);
47:                else if (command.equalsIgnoreCase("B"))
48:                   theATM.selectAccount(ATM.SAVINGS);
49:                else if (command.equalsIgnoreCase("C"))
50:                   theATM.reset();
```

Continued...

File ATMTester.java

```
51:                else
52:                   System.out.println("Illegal input!");
53:             }
54:             else if (state == ATM.TRANSACT)
55:             {
56:                System.out.println("Balance="
                      + theATM.getBalance());
57:                System.out.print("A=Deposit, B=Withdrawal,
                      C=Cancel: ");
58:                String command = in.next();
59:                if (command.equalsIgnoreCase("A"))
60:                {
61:                   System.out.print("Amount: ");
62:                   double amount = in.nextDouble();
63:                   theATM.deposit(amount);
64:                   theATM.back();
65:                }
66:                else if (command.equalsIgnoreCase("B"))
67:                {
```

Continued...

File ATMTester.java

```
68:                System.out.print("Amount: ");
69:                double amount = in.nextDouble();
70:                theATM.withdraw(amount);
71:                theATM.back();
72:             }
73:             else if (command.equalsIgnoreCase("C"))
74:                theATM.back();
75:             else
76:                System.out.println("Illegal input!");
77:          }
78:       }
79:    }
80: }
81:
```

Continued...

Osservazioni

- Perché `Bank` non mantiene una collezione di `BankAccount`?
- `Bank` mantiene la lista dei clienti così si può gestire anche l'accesso.
- Siccome un cliente può avere diversi conti, con `Customer` possiamo raggrupparli
- Quindi non necessita avere anche una lista dei conti direttamente in `Bank`

Osservazioni

- Cambio nei requisiti:
  - si richiede di salvare i saldi dei conti in un file dopo ogni transazione e recuperarli ogni volta che il programma riparte
- Cosa dobbiamo modificare nel progetto?
- `Bank` ha una responsabilità aggiuntiva: caricare e salvare i conti.
- `Bank` può assolevere a questa responsabilità in quanto ha accesso agli oggetti `Customer`