Conto bancario

Programmazione I B

2001-02

A. Martelli

Input/Output da finestre di dialogo

La maggior parte delle applicazioni Java interagiscono con l'utente attraverso *interfacce utente grafiche* (**GUI**).

Per un primo esempio si può utilizzare la classe **JOptionPane** contenuta nel package **javax.swing**

La classe realizza una finestra di dialogo e fornisce diversi metodi *statici* per leggere e scrivere stringhe nella finestra.

Metodi della classe **JOptionPane**:

static String showInputDialog(**String** messaggio)

Ad esempio **JOptionPane.showInputDialog("Inserire cifra")** crea una finestra come questa:

restituisce la stringa inserita in questo campo dall'utente

Metodi della classe **JOptionPane**:

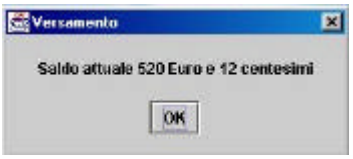
static void **showMessageDialog** con due parametri: il primo deve essere null, il secondo è una stringa che rappresenta il messaggio da visualizzare.

Es. **JOptionPane.showMessageDialog(null, "errore: prelievo troppo alto")**

showMessageDialog può avere anche quattro parametri:

il primo è null, il secondo è il messaggio, il terzo il titolo della finestra, il quarto indica il tipo di messaggio (visualizzato con icone diverse)

Es. **JOptionPane.showMessageDialog(null, "Saldo attuale" + ..., "Versamento", JOptionPane.PLAIN_MESSAGE)**



Esempio di programma che legge due numeri e ne stampa la somma.

Il file deve contenere l'istruzione

```
import javax.swing.JOptionPane;
```

Il main deve terminare con l'istruzione `System.exit(0);` che termina l'esecuzione del programma. Se non si mette questa istruzione, il programma non termina anche se l'esecuzione raggiunge la fine del main. Questo vale in generale per tutte le applicazioni grafiche.

```
import javax.swing.JOptionPane;
// legge due interi e ne stampa la somma
public class Somma
{  public static void main(String[] args)
   {  String primoNumero, secondoNumero;
      int num1, num2, somma;

      primoNumero =
         JOptionPane.showInputDialog("Primo numero");
      secondoNumero =
         JOptionPane.showInputDialog("Secondo numero");

      num1 = Integer.parseInt(primoNumero);
      num2 = Integer.parseInt(secondoNumero);
      somma = num1 + num2;
      JOptionPane.showMessageDialog(null,
                                    "La somma è "+somma);
      System.exit(0);
   }
}
```



```
String primoNumero, secondoNumero;
   int num1, num2, somma;

   primoNumero =
       JOptionPane.showInputDialog("Primo numero");
   .....
   num1 = Integer.parseInt(primoNumero);
```

Il programma legge i numeri come stringhe assegnandoli alle variabili **`primoNumero`** e **`secondoNumero`**. Le stringhe vengono convertite in interi con il metodo **`Integer.parseInt`**.

Se in input si fornisce una stringa che non rappresenta un intero, il metodo **parseInt** lancia un'eccezione.

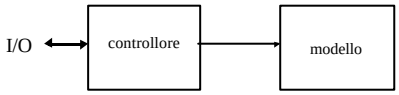
Un programma per gestire conti bancari

Il programma è realizzato ispirandosi alla cosiddetta architettura *model-view-controller*.

Noi usiamo una versione semplificata in cui la gestione di un conto è ottenuta con due componenti (oggetti):

il **modello** (**BankAccount**) implementa il conto bancario con le sue operazioni

il **controllore** (**AccountController**) gestisce il flusso dei dati dall'input al modello e dal modello all'output.



Il **modello**: *interfaccia* della classe **BankAccount**

boolean deposit(int amount)	Se amount non è negativo, aggiungi amount alla cifra sul conto e restituisci true. Altrimenti non modificare la cifra e restituisci false.
boolean withdraw(int amount)	Se amount non è negativo e non è maggiore della cifra sul conto, sottrai amount e restituisci true. Altrimenti non modificare la cifra e restituisci false.
int getBalance()	Restituisci il saldo del conto.

Il conto è in Euro. Per evitare le complicazioni dell'uso dei double le cifre sono rappresentate come interi in centesimi.

Ad es. 2.25 € è rappresentato con l'intero 225.

Il costruttore implicito di **BankAccount** inizializza gli oggetti a 0.

Se il valore del parametro di **deposit** o **withdraw** non è corretto, oltre a restituire false, viene anche dato un messaggio d'errore in una finestra di dialogo.


```
import javax.swing.*;

public class BankAccount
{  private int balance;

   public boolean deposit(int amount)
   {  if (amount >= 0)
      {  balance = balance + amount;
         return true;
      }
      else
      {  JOptionPane.showMessageDialog(null,
               "errore: versamento negativo");
         return false;
      }
   }

.........
```

```
    public boolean withdraw(int amount)
    {  if (amount < 0)
       {  JOptionPane.showMessageDialog(null,
             "errore: prelievo negativo");
          return false;
       }
       else if (amount > balance)
       {  JOptionPane.showMessageDialog(null,
             "errore: prelievo troppo alto");
          return false;
       }
       else
       {  balance = balance - amount;
          return true;
       }
    }

    public int getBalance()
    {  return balance;
    }
}
```


Il **controllore**: *Interfaccia* della classe **AccountController**

AccountController(BankAccount a)	Il costruttore ha come parametro il BankAccount che deve controllare
void processTransactions()	Esegue un ciclo che legge da una finestra di dialogo l'operazione da eseguire, la esegue sul conto e genera in output il risultato in una finestra di dialogo


```
import javax.swing.*;

public class AccountController
{  private BankAccount account;

   public AccountController(BankAccount a)
   {  account = a;
   }
.....
```

La variabile privata **account** contiene un riferimento al **BankAccount** da controllare.


```
public void processTransactions()
    {   boolean done = false;
        do
        {   String input = JOptionPane.showInputDialog(
                "Inserire: Versamento, Prelievo o Quit");
            char ch = input.toUpperCase().charAt(0);
            if (ch == 'Q')
                done = true;
            else if (ch == 'V')
            {   int amount = readAmount();
                boolean ok = account.deposit(amount);
                if (ok)
                    JOptionPane.showMessageDialog(null,
                        "Saldo attuale " + convert(account.getBalance()),
                        "Versamento", JOptionPane.PLAIN_MESSAGE);
            }
            else if (ch == 'P')
            {   int amount = readAmount();
                boolean ok = account.withdraw(amount);
                if (ok)
                    JOptionPane.showMessageDialog(null,
                        "Saldo attuale " + convert(account.getBalance()),
                        "Prelievo", JOptionPane.PLAIN_MESSAGE);
            }
            else
            {   JOptionPane.showMessageDialog(null,
                    "Comando errato");
            }
        } while (!done);
    }
```


La classe **AccountController** contiene anche due metodi *privati*:

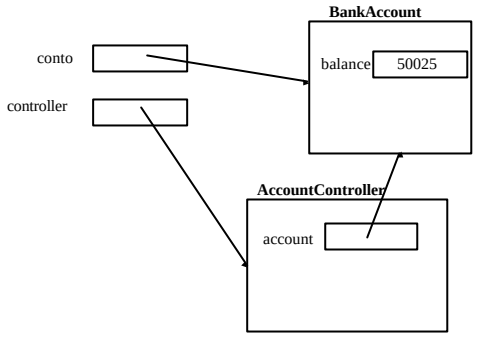
```
.......
private int readAmount()
  {  String input = JOptionPane.showInputDialog(
        "Inserire cifra");
     return (int)Math.round(Double.parseDouble(input)*100);
  }

private String convert(int amount)
  {return (amount/100 + " Euro e "+ amount%100+"centesimi");
  }
}
```

Il primo metodo converte la stringa letta in un intero, e il secondo fa il viceversa.

Il seguente main crea un conto, un controllore di questo conto ed esegue il metodo **processTransaction** del controllore.

```
public class Prova
{  public static void main (String[] args)
   {  BankAccount conto = new BankAccount( );
      AccountController controller = new
                          AccountController(conto);
      controller.processTransactions();
      System.exit(0);
   }
}
```



Per gestire due conti si potrebbero definire due controller, uno per ogni conto:

```
{  public static void main (String[] args)
   {  BankAccount conto1 = new BankAccount( );
      AccountController controller1 = new
                        AccountController(conto1);
      BankAccount conto2 = new BankAccount( );
      AccountController controller2 = new
                        AccountController(conto2);
      ..........
```

e poi chiedere all'utente su quale conto vuole operare.