Programmazione III

Prof.ssa Liliana Ardissono
Dipartimento di Informatica
Università di Torino

Pattern architetturale Observer Observable

Pattern Observer Observable

Il pattern Observer-Observable è un pattern utilizzato nelle applicazioni software per renderle modulari.

Observer Observable è molto utilizzato nelle librerie per lo sviluppo di interfacce grafiche e dobbiamo comprenderlo a fondo per capire come funzionano tali librerie.

Observer Observable è però indipendente dalle GUI e viene utilizzato anche in altri contesti di sviluppo di applicazioni. Per esempio, ogni volta che un componente di una applicazione deve reagire autonomamente ai cambiamenti di stato di un altro componente.

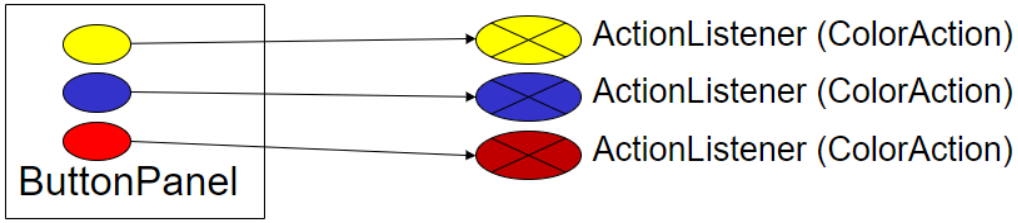
Noi vediamo Observer Observable in relazione alle GUI.

Il pattern Observer-Observable viene utilizzato nelle GUI per permettere di disaccoppiare la gestione delle azioni utente dalla gestione delle componenti delle interfacce grafiche. In pratica, si assume che:

- L'applicazione rappresenti il proprio stato in modo esplicito, attraverso opportune variabili;
- Le azioni dell'utente sull'interfaccia grafica, catturate dai Listener, cambino lo stato (cioè, il valore delle variabili) dell'applicazione;
- L'interfaccia utente osservi lo stato dell'applicazione e, se ci sono cambiamenti, reagisca modificando i dati visualizzati secondo le proprie regole interne.

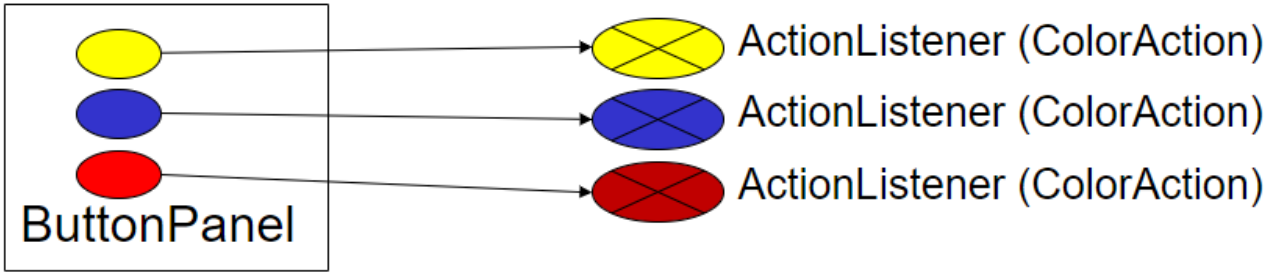
Es. nel ButtonPanel *(l'esempio non è preciso – serve solo per dare un'idea intuitiva)*

- rendiamo esplicito il colore del background, dichiarando una variabile «colore» dell'applicazione;
- quando l'utente schiaccia un bottone, il Listener del bottone modifica il valore della variabile «colore» con il valore opportuno (giallo/blu/rosso)
- Il ButtonPanel osserva i cambiamenti di valore della variabile «colore» e, a seconda del valore che essa assume, colora il proprio background opportunamente



Questo permette di:

- Semplificare i listener dei componenti grafici: i listener hanno solo il compito di interpretare le azioni utente e modificare lo stato dell'applicazione di conseguenza.
- Definire le regole di visualizzazione dell'interfaccia grafica nell'interfaccia stessa. Le regole potrebbero essere complesse, e devono accedere ai componenti dell'interfaccia utente → è bene che siano interne ad essa.



Observer Observable in Java

Vediamo il pattern Observer Observable direttamente con esempi Java.

In Java è possibile creare degli oggetti **Observer** o **Observable (deprecate da JDK 10)**.

Un oggetto **Observer** osserva uno o più oggetti **Observable**, registrandosi presso questi oggetti.

Quando un oggetto **Observable** modifica il proprio stato, notifica il cambiamento a tutti gli **Observer** registrati presso di lui.

La notifica consiste nella esecuzione del metodo **update** degli oggetti **Observer**.

Più precisamente:

Observer è un'*interfaccia* che contiene il metodo

update(Observable ob, Object extra_arg)

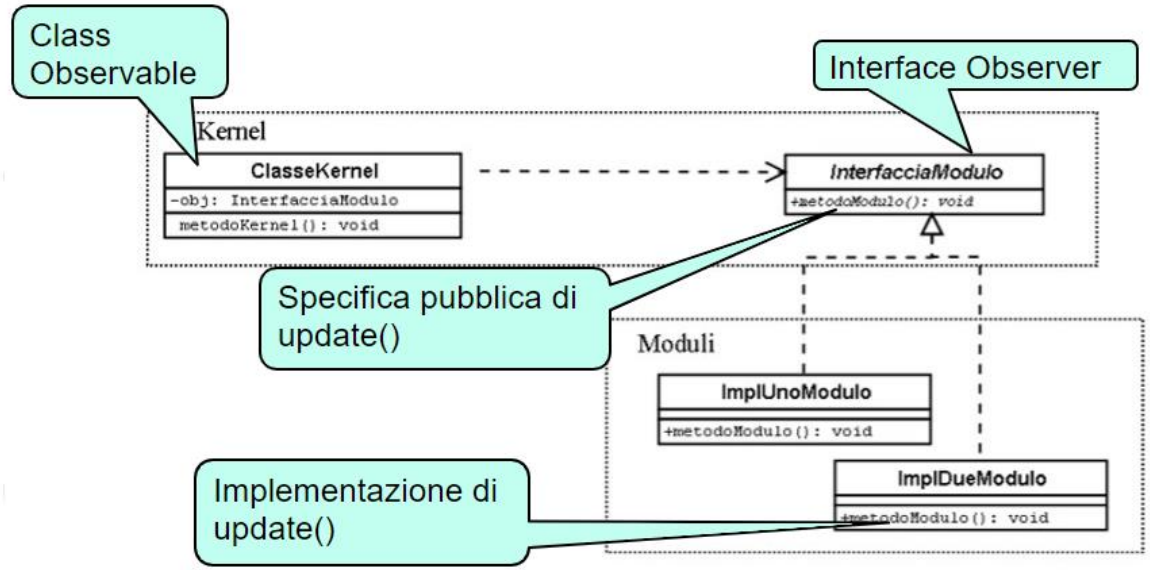
che viene chiamato ogni volta che l'oggetto osservato (**Observable**) è modificato.

Il parametro **ob** è l'oggetto **Observable** che notifica il cambiamento

Il parametro **extra_arg** può essere usato per passare informazione addizionale.

Verso i pattern architetturali

Situazione simile all'esempio di Arrays e soprattutto di Jbutton!



Observable è una *classe (deprecata da Java10)* che fornisce i metodi

addObserver(Observer o)

aggiunge l'**Observer o** all'insieme degli osservatori di questo oggetto

setChanged()

marca questo oggetto, indicando che il suo stato è cambiato

notifyObservers(Object arg)

se questo oggetto ha cambiato stato, notifica tutti i suoi osservatori chiamando il loro metodo **update**. I due argomenti di **update** sono questo oggetto e l'oggetto **arg**.

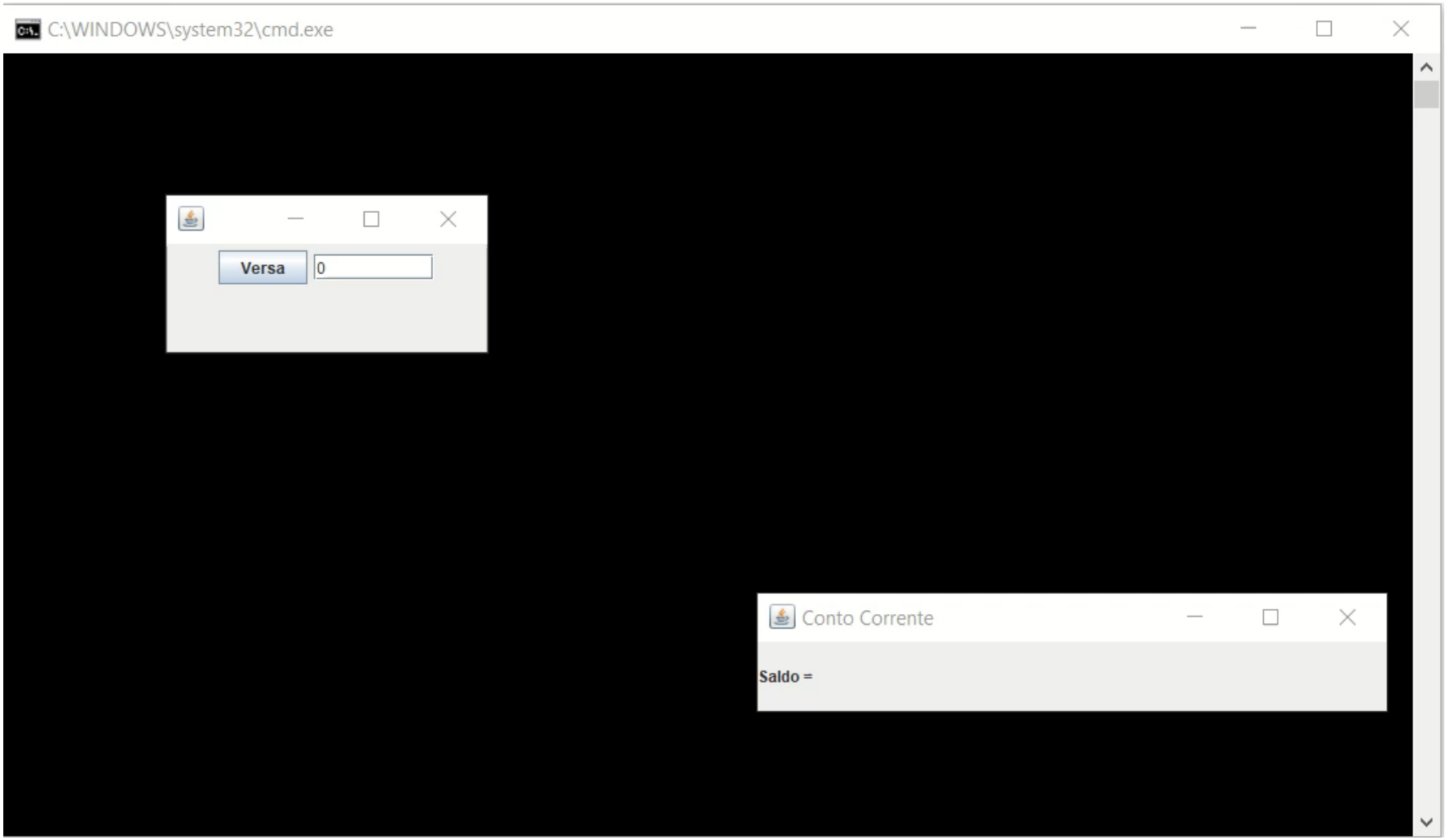
notifyObservers()

come sopra, ma il secondo argomento di **update** è **null**.

Si supponga ad esempio di avere una classe **ContoBancario** e di voler realizzare una finestra associata ad un oggetto **ContoBancario** che mostri sempre il valore aggiornato del saldo del conto.

Questo può essere realizzato definendo il conto come **Observable** e registrando la finestra come **Observer** del conto.

Ogni volta che il conto viene modificato con un versamento o un prelievo, il conto notificherà il cambiamento alla finestra eseguendone il metodo **update()**.


C:\WINDOWS\system32\cmd.exe
Versa
0
Conto Corrente
Saldo =

```
class ContoBancario extends Observable {
private int saldo = 0;
   public void prelievo(int val)
   {  saldo -= val;
      setChanged();      // marca il cambiamento di stato
      notifyObservers(); // notifica il cambiamento agli osservatori
   }
   public void versamento(int val)
   {  saldo += val;
      setChanged();      // marca il cambiamento di stato
      notifyObservers(); // notifica il cambiamento agli osservatori
   }
   public int getSaldo()
   {  return saldo;
   }
}
```

la finestra non conosce l'oggetto che osserva: essa chiede solo che il parametro ob di update sia un ContoBancario

```
public class Finestra extends JFrame implements Observer
{  private JLabel display;

   public Finestra() {
      display = new JLabel();
      add(display);
      display.setText("Saldo = " + 0);
      setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
      pack();
      setVisible(true);
   }
   public void update(Observable ob, Object extra_arg) {
            if (ob!=null && ob instanceof ContoBancario) {
            display.setText("Saldo = " +
                  ((ContoBancario)ob).getSaldo());
            }
   }
}
```

```
class GestisciOperazioni extends JFrame implements ActionListener
{
private JButton button;
    private JTextField inputVal;
    private JPanel panel;
    private ContoBancario cb;

    public GestisciOperazioni(ContoBancario conto) {
        super ("GestisciOperazioni");
        cb = conto;
        panel = new JPanel();
        add(panel);
        button = new JButton("Versa");
        panel.add(button);
        button.addActionListener(new ActionListener() {
                public void actionPerformed (ActionEvent e){
                    int val = Integer.parseInt(inputVal.getText()}
                    cb.versamento(val);
                }
                });
```

```
inputVal = new JTextField("0", 8);
      panel.add(inputVal);

      setLocation(100,100);
      setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
      setSize(400,100);
      setVisible(true);
   }
}
```

Il file **ObserverConto** contiene un metodo *main()* che crea un **ContoBancario**, una **Finestra** collegata a questo conto e registra la Finestra presso il conto.

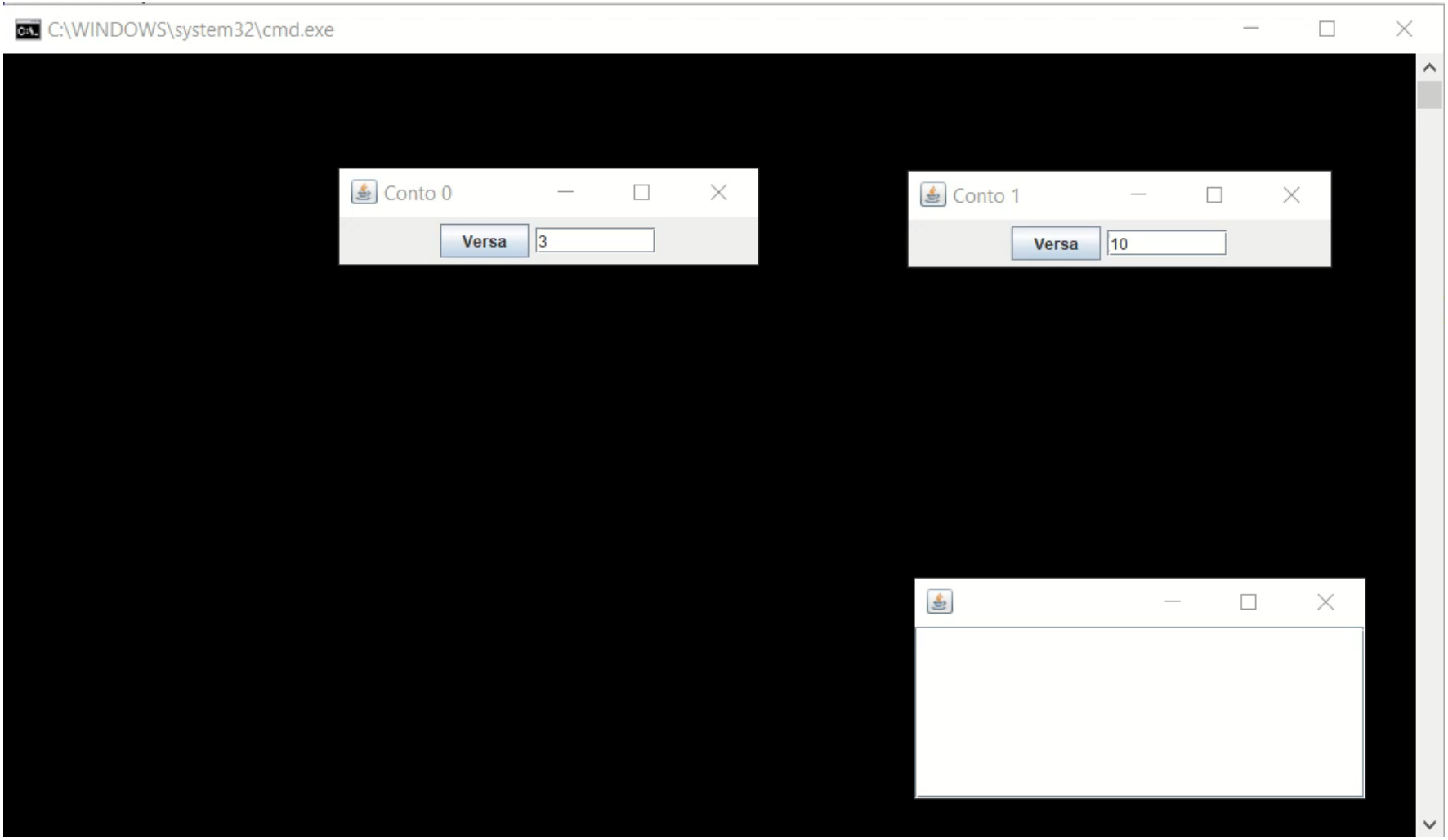
Viene fornita anche una interfaccia grafica **GestisciOperazioni** attraverso cui è possibile eseguire dei versamenti sul conto.

```
class ObserverConto {
   public static void main(String[] args) {
            ContoBancario cb = new ContoBancario();
            Finestra f = new Finestra();
            cb.addObserver(f); // aggancia l'osservatore all'osservato
            GestisciOperazioni v = new GestisciOperazioni(cb);
   }
}
```

La **Finestra** è completamente disaccoppiata dal conto osservato: la finestra lo conosce solo attraverso il parametro del metodo **update**.

In questo modo, per esempio, è possibile registrare la finestra presso conti diversi: la finestra può determinare l'identità del conto dinamicamente analizzando il primo parametro di **update**.

Ad esempio il file **ObserverContoApp1** realizza una finestra che osserva più conti. La finestra contiene una **JTextArea** inserita in un **JSCrollPane** in cui vengono scritti numero e saldo di un conto, ogni volta che questo viene modificato.


C:\WINDOWS\system32\cmd.exe
Conto 0
Versa
3
Conto 1
Versa
10

NB: le classi/interface Observer e Observable sono state deprecate a partire da Java10 in quanto non si usano più in modo esplicito nello sviluppo delle interfacce utente.

Le moderne librerie per lo sviluppo di interfacce grafiche (per esempio, JavaFX) hanno classi che implementano il meccanismo internamente → il programmatore può utilizzare queste classi per

- Definire oggetti osservabili (per esempio, liste osservabili - ObsevableList)
- Definire le componenti grafiche che osservano tali oggetti (per esempio, ListView)

Tuttavia, il programmatore deve conoscere il meccanismo sottostante perché deve comunque:

- Definire i metodi di gestione delle componenti grafiche
- «Agganciare» gli osservatori agli osservati esplicitamente (solo il programmatore sa chi deve osservare chi)

Ringraziamenti

Grazie al Prof. Emerito Alberto Martelli del Dipartimento di Informatica dell’Università di Torino per aver redatto la prima versione di queste slides.