Programmazione III

Prof.ssa Liliana Ardissono
Dipartimento di Informatica
Università di Torino

Pattern architetturale Model View Controller (MVC)

Pattern MVC per le GUI

Il pattern MVC (Model View Controller) organizza l'architettura delle applicazioni che hanno un'interfaccia grafica (GUI) in componenti per aumentare la loro modularità e per separare le attività da svolgere nella gestione dell'interazione con l'utente.

MVC divide l'applicazione in 3 componenti principali:

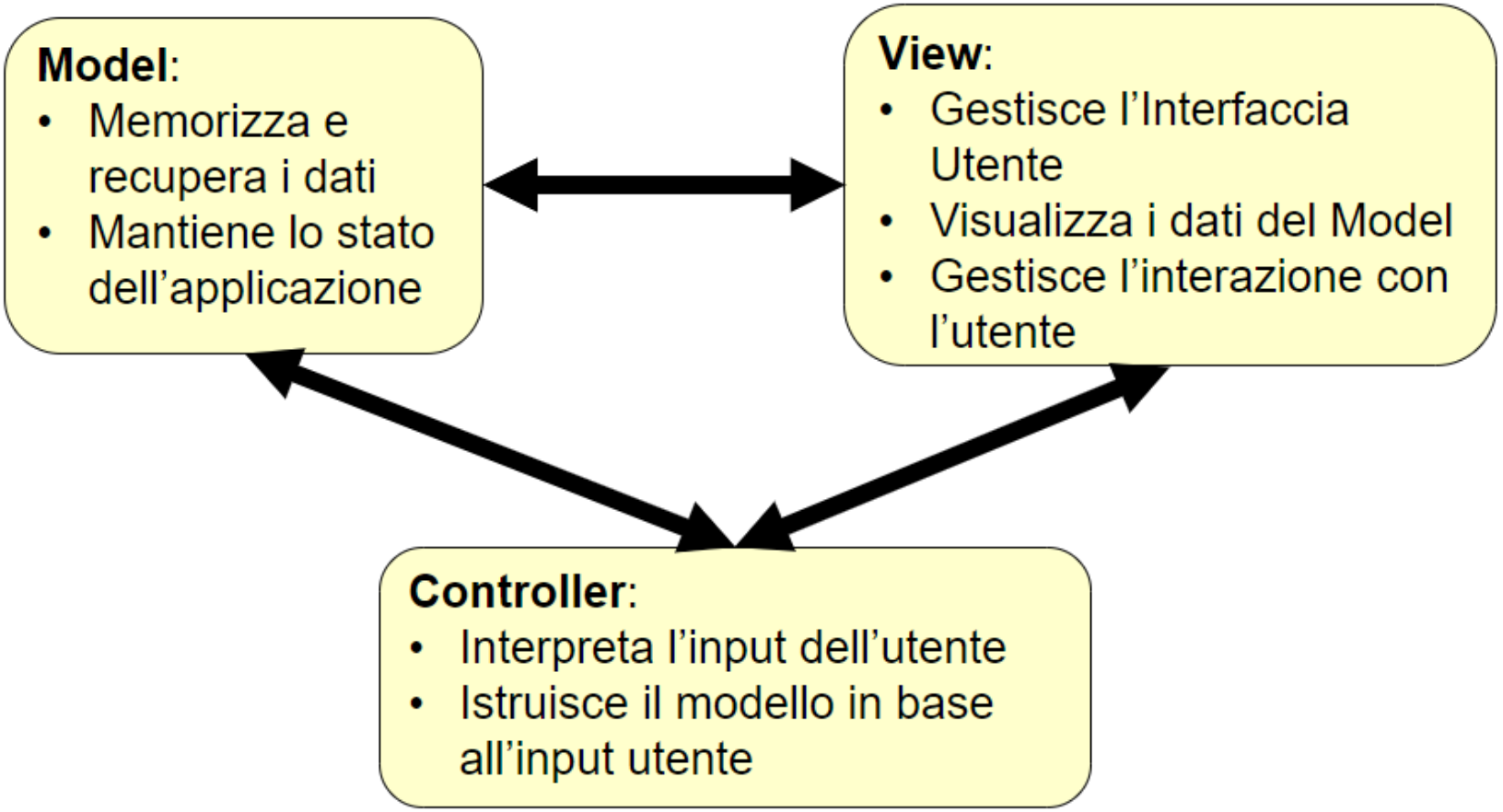
- Modello (Model)
- Vista (View)
- Controllore (Controller)

MVC

Un programma si compone di

- **Modello** (**Model**): modella e calcola il problema che desideriamo risolvere.
- **Vista** (**View**): rappresenta una “fotografia” dello stato interno del modello spesso per facilitarne la sua lettura/interpretazione all’utente umano.
- **Controllore** (**Controller**): controlla il flusso di dati nel programma, dalla vista al modello e quindi nuovamente alla vista.

Pattern MVC


Model:
• Memorizza e recupera i dati
• Mantiene lo stato dell'applicazione
View:
• Gestisce l'Interfaccia Utente
• Visualizza i dati del Model
• Gestisce l'interazione con l'utente
Controller:
• Interpreta l'input dell'utente
• Istruisce il modello in base all'input utente

MVC – flusso dei dati

- L’utente agisce sulla vista di un programma agendo su una delle sue componenti di controllo (per esempio un bottone).
- Il controllore è avvertito di tale evento ed esamina la vista per rilevarne le informazioni aggiuntive.
- Il controllore invia tali informazioni al modello che effettua la computazione richiesta e aggiorna il proprio stato interno.
- Il controllore (o il modello) richiede alla vista di visualizzare il risultato della computazione.
- La vista interroga il modello sul suo nuovo stato interno e visualizza l’informazione all’utente.

MVC

- La computazione viene guidata dalla serie di eventi generati dall'utente tramite la GUI.
- Il programma processa gli eventi come input, aggiorna il proprio modello interno e lo visualizza tramite la vista.
- Il controllore ha il compito di gestire il flusso di eventi e dati dalla vista al modello e quindi nuovamente verso la vista.
- Più controllori, viste e modelli possono coesistere a formare un programma, se necessario.

MVC – applicazione di gestione di un conto corrente bancario – modello dei dati

```
class ContoBancario {       // Model dell'applicazione
        private int saldo;
        public ContoBancario() {
                saldo = 0;
        }
        public void versamento(int val) {
                saldo += val;
        }
        public int getSaldo() {
                return saldo;
        }
}
```

MVC – esempio – senza Observer Observable - I

1. Utente inserisce il numero di soldi e preme “Versa”
2. l’evento è ascoltato dal controller
3. il controller invia il messaggio di versa() al modello
4. il controller invia il messaggio di updateView() alla vista
5. la vista richiede i dati al modello per aggiornarsi (getVal())

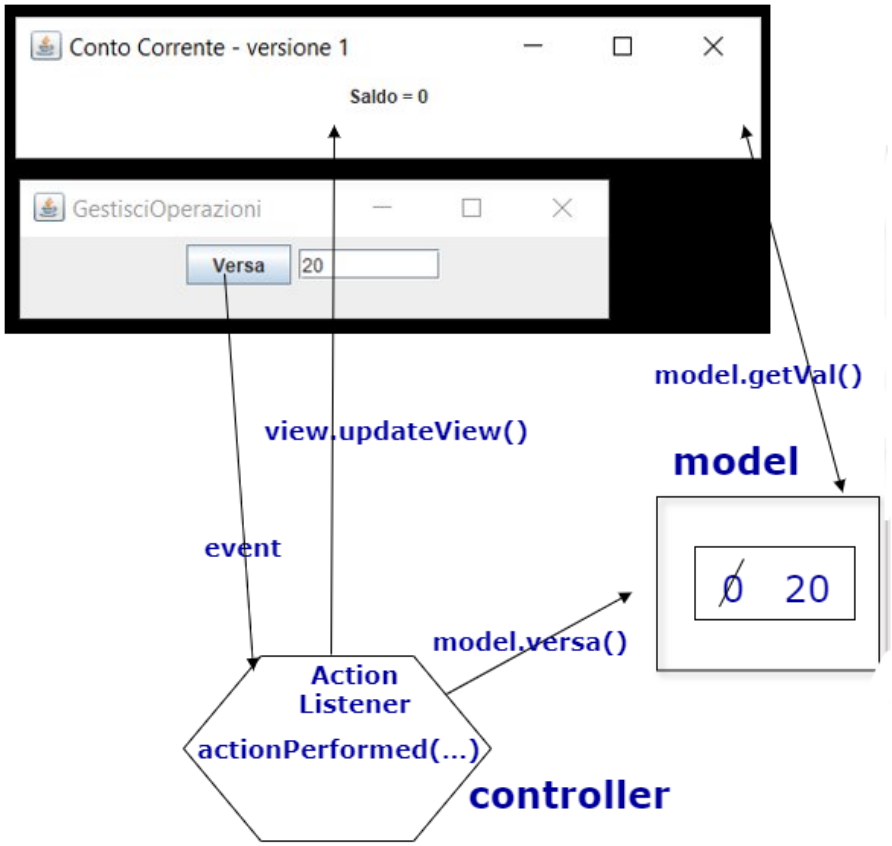
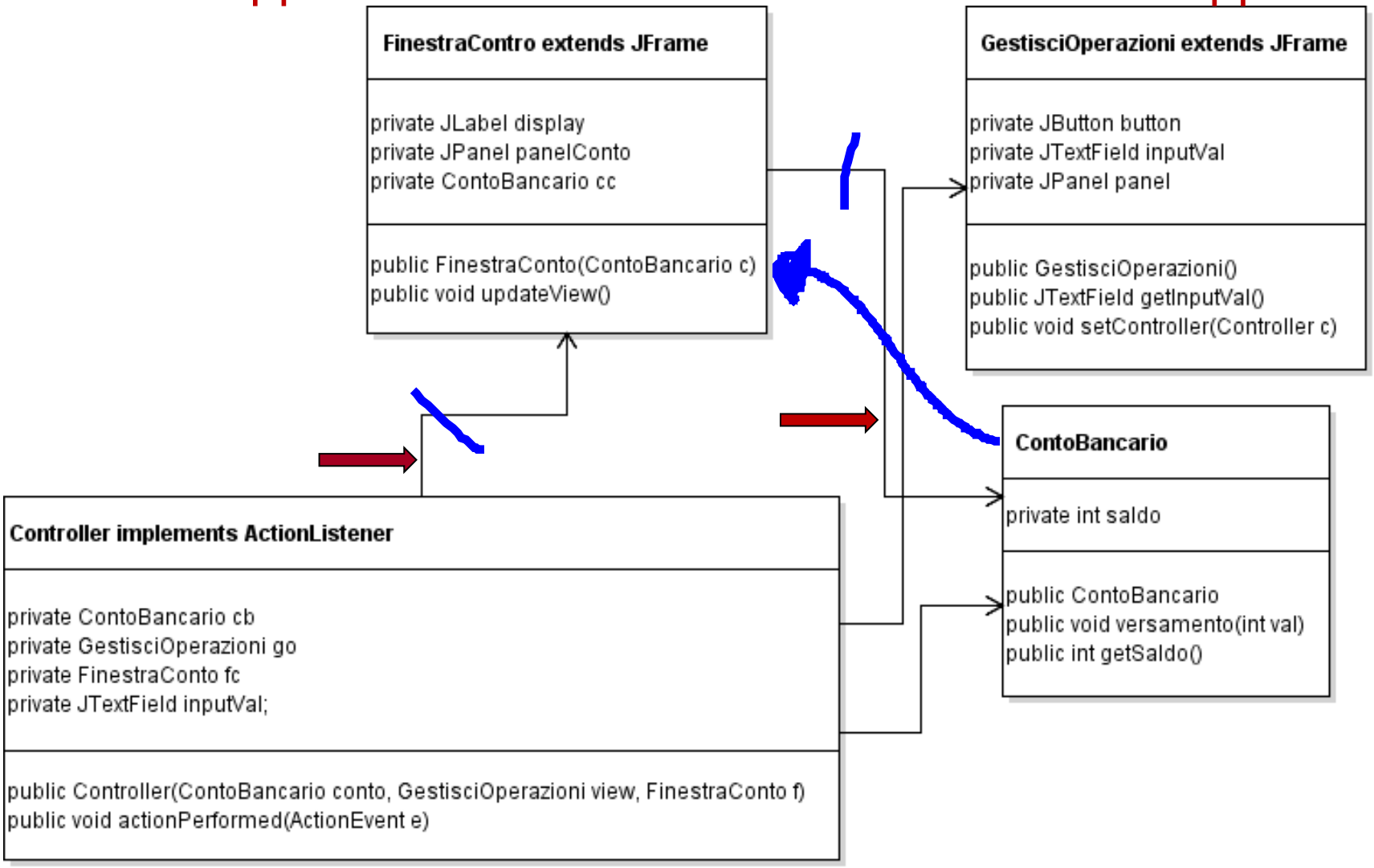


Diagramma delle classi

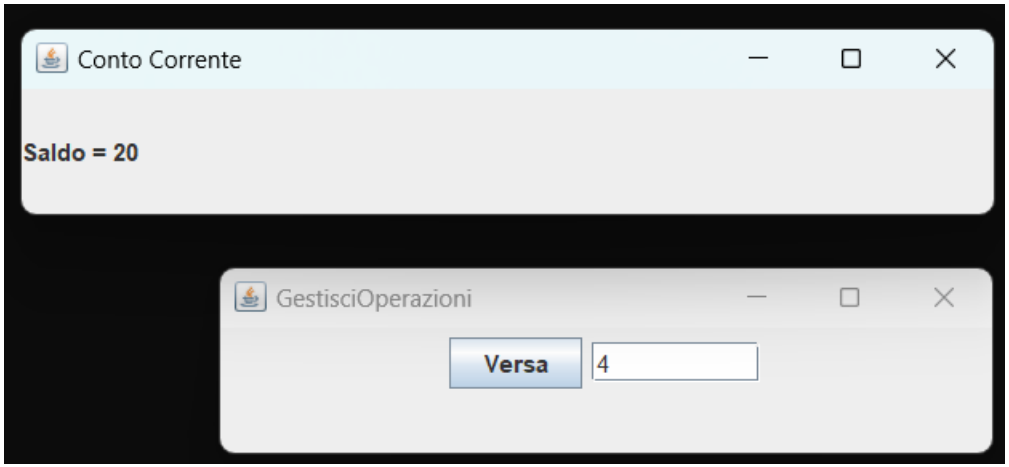
Ci sono troppe relazioni tra le classi → non è disaccoppiato!!



MVC – esempio con Observer Obs. - I

Combinando MVC con il pattern Observer Observable si disaccoppiano parzialmente le componenti:

- la comunicazione tra il modello (osservato) e la vista del conto bancario (osservatore) viene gestita attraverso la registrazione di observers e la notifica dei cambiamenti di stato agli observers



MA: ricordate che Observer.java e Observable.java sono deprecate → voi dovrete usare le librerie grafiche nuove (e le property), descritte con Java FXML!

MVC – Model osservabile

```
class ContoBancario extends Observable {
        private int saldo;
        public ContoBancario() { saldo = 0; }
        public void settaSaldoIniziale(int val) {
                saldo = val; setChanged();  notifyObservers(); }
        public void prelievo(int val) {
                saldo -= val; setChanged(); notifyObservers(); }
        public void versamento(int val) {
                saldo += val; setChanged(); notifyObservers(); }
        public int getSaldo() { // serve per interrogare il model
                return saldo; }
}
```

MVC – Vista 1: Osservatore

```
class FinestraConto extends JFrame implements Observer {
  private JLabel display;
  public FinestraConto() {
        super("Conto Corrente");
        display = new JLabel(); display.setText("Saldo = ");
        add(display);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setSize(500,100); setVisible(true); }
  public void update(Observable ob, Object extra_arg) {
        if (ob!=null && ob instanceof ContoBancario) {
              display.setText("Saldo = " +
                          ((ContoBancario)ob).getSaldo()); }
}}
```

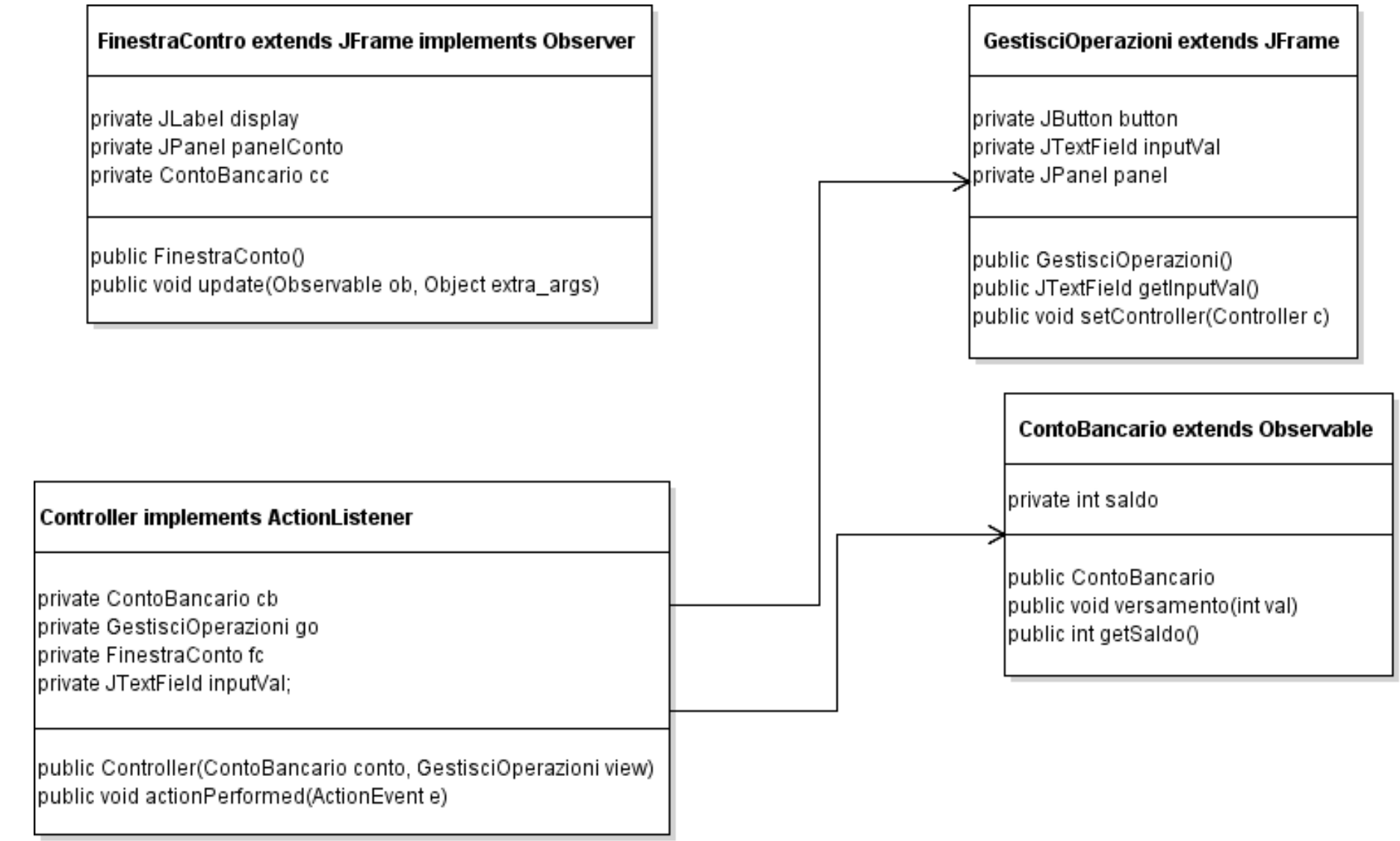
MVC – Vista 2 con Controller

```
class GestisciOperazioni extends JFrame {
   private JButton button; private JTextField inputVal;
   private JPanel panel; private ContoBancario cb;
   public GestisciOperazioni(ContoBancario conto) {
        super("GestisciOperazioni"); cb = conto;
        panel = new JPanel(); button = new JButton("Versa"); …
        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); …
  }}
```

MVC – Main

```
public class ObserverContoApp2 {
    public static void main(String[] args) {
                                // crea il Model (osservabile)
        ContoBancario cb = new ContoBancario();
                                // crea la prima view
        FinestraConto f = new FinestraConto();
                                // aggiunge l'osservatore del Model
        cb.addObserver(f);
                                // crea la seconda view che conosce
                                // il Model ma contiene il Controller
        GestisciOperazioni v = new GestisciOperazioni(cb);
        }
}
```

Diagramma delle classi



MVC – ulteriore miglioramento - I

Si può migliorare il codice introducendo una interface per rappresentare le GUI (pattern Façade) → risulta facile cambiare la GUI senza modificare sensibilmente il programma.

→ INDIPENDENZA DEL CONTROLLER DALLA IMPLEMENTAZIONE DELLA VISTA

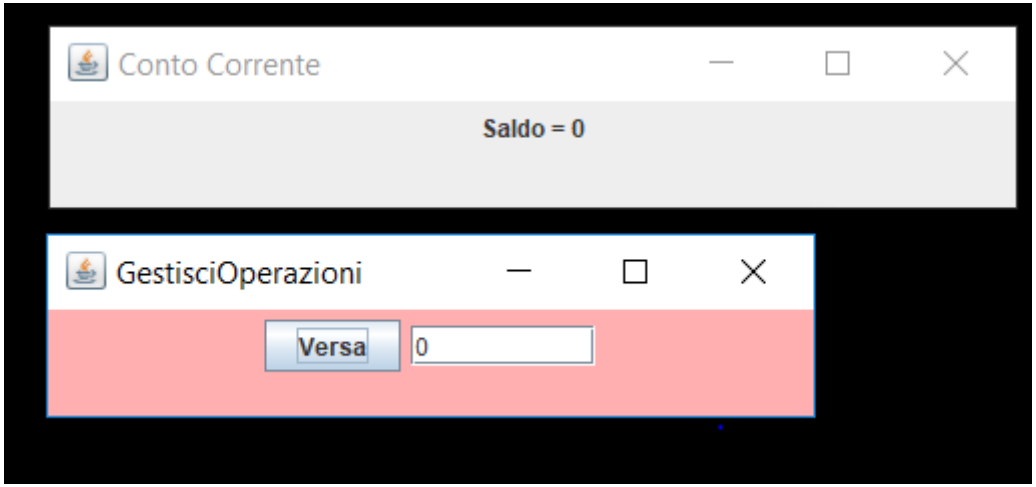
```
interface IGestisci {
        public JTextField getInputVal();
        public void setController(Controller c);
}
class GestisciOperazioni1 extends JFrame implements IGestisci { …}
class GestisciOperazioni2 extends JFrame implements IGestisci { …}
```

MVC – ulteriore miglioramento - II

```
public class ObserverContoApp2MVC {
   public static void main(String[] args) {
        ContoBancario cb = new ContoBancario(); // modello
        FinestraConto f = new FinestraConto();  // prima vista
        cb.addObserver(f); // aggiungo la prima vista come
osservatrice del model conto corrente bancario
         cb.settaSaldoIniziale(0);

        //IGestisci v = new GestisciOperazioni1();  // prima vista
        IGestisci v = new GestisciOperazioni2();  // seconda vista
        Controller c = new Controller(cb, v);   // controller
        v.setController(c);       // aggancio il controller alla vista
        }
}
```

MVC – ulteriore miglioramento - III



Siccome il Controller richiede un **IGestisci**, si può passare come parametro la GUI desiderata cambiando solo una riga di codice nel main() dell'applicazione. Il Controller non deve cambiare codice e controlla GUI diverse basandosi sui loro API.

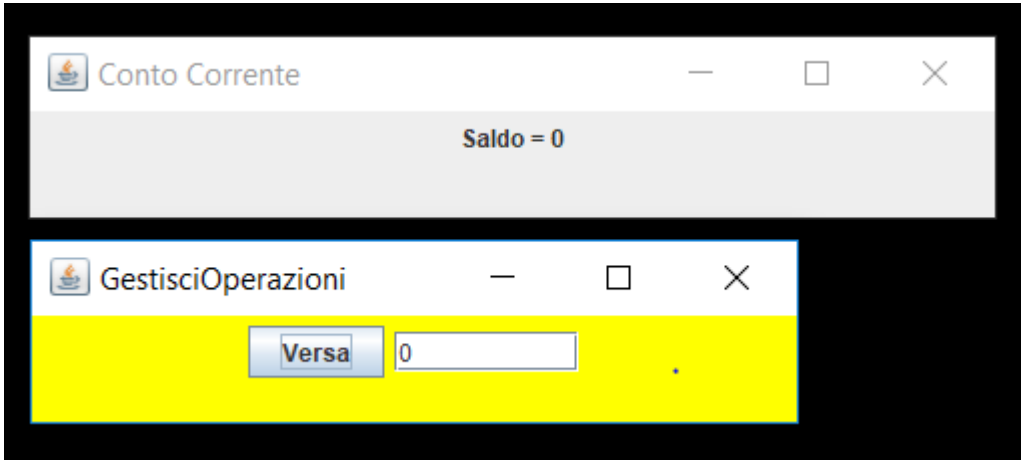
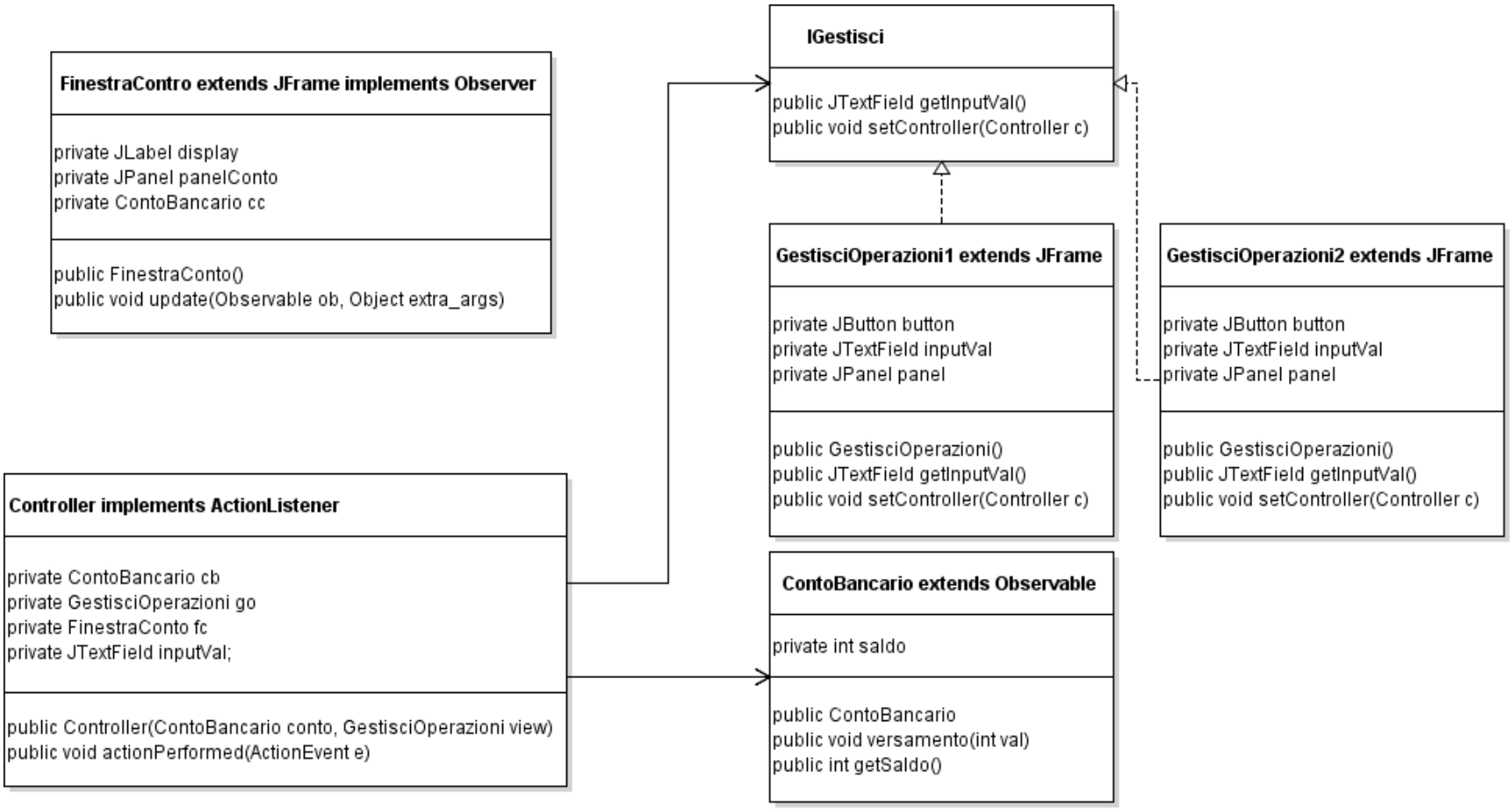


Diagramma delle classi dell'esempio



Vedere l'applicazione ObserverContoApp2MVC

MVC - vantaggi

- Le classi che formano l'applicativo possono essere più facilmente riutilizzate
- L'applicativo è organizzato in parti semplici e comprensibili (ogni parte ha le sue specifiche finalità)
- La modifica di una parte non coinvolge e non interferisce con le altre parti (maggiore flessibilità nella manutenzione del software)

Ringraziamenti

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