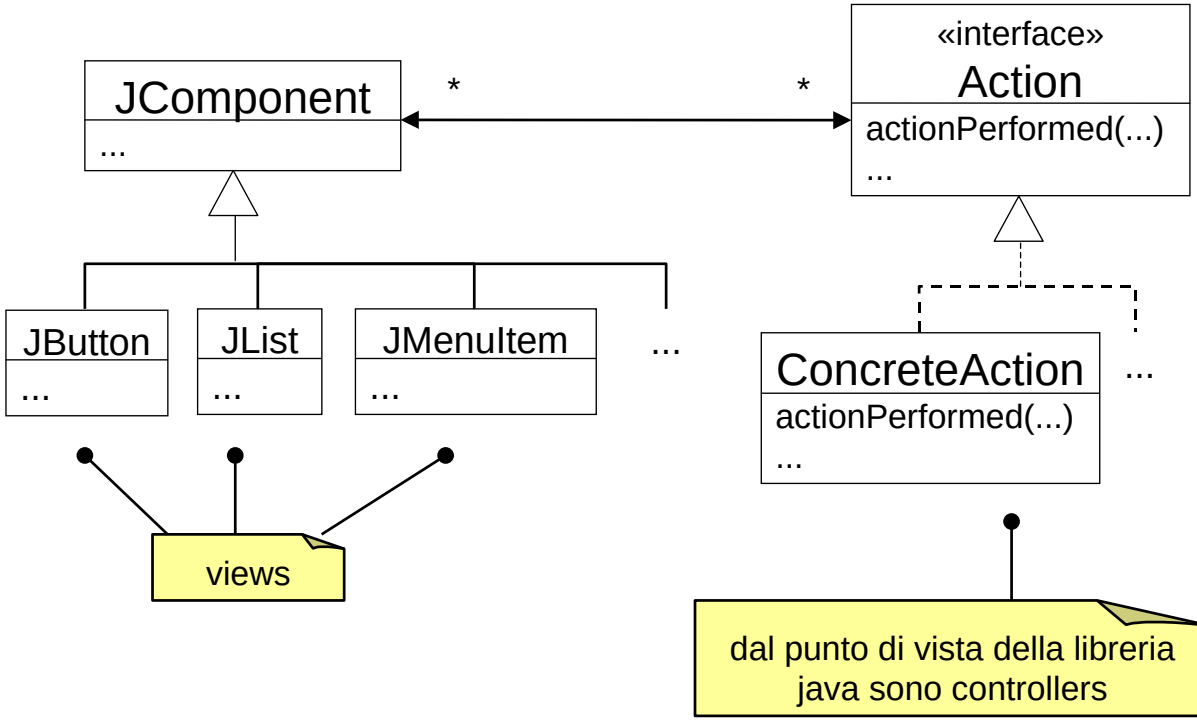
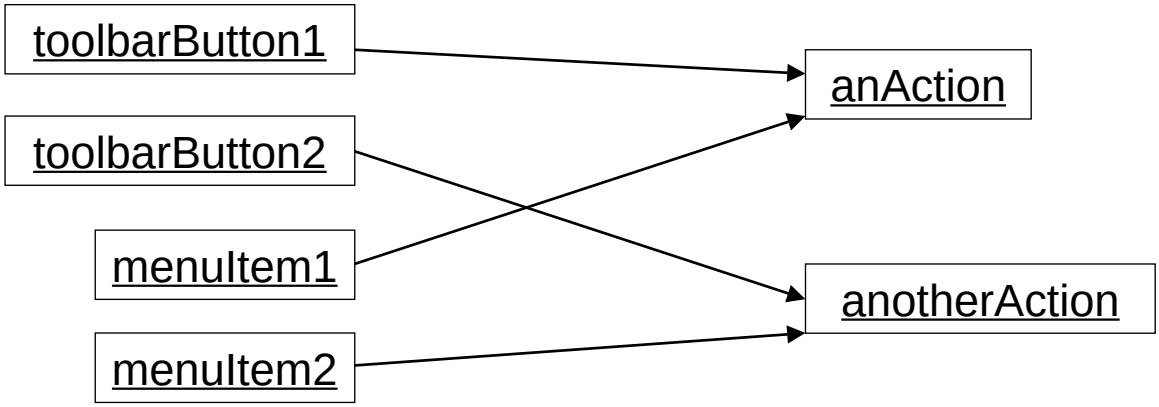
MVC: esempio in java

views e controllers in javax.swing

supporto per disaccoppiamento di viste e controller nel linguaggio java

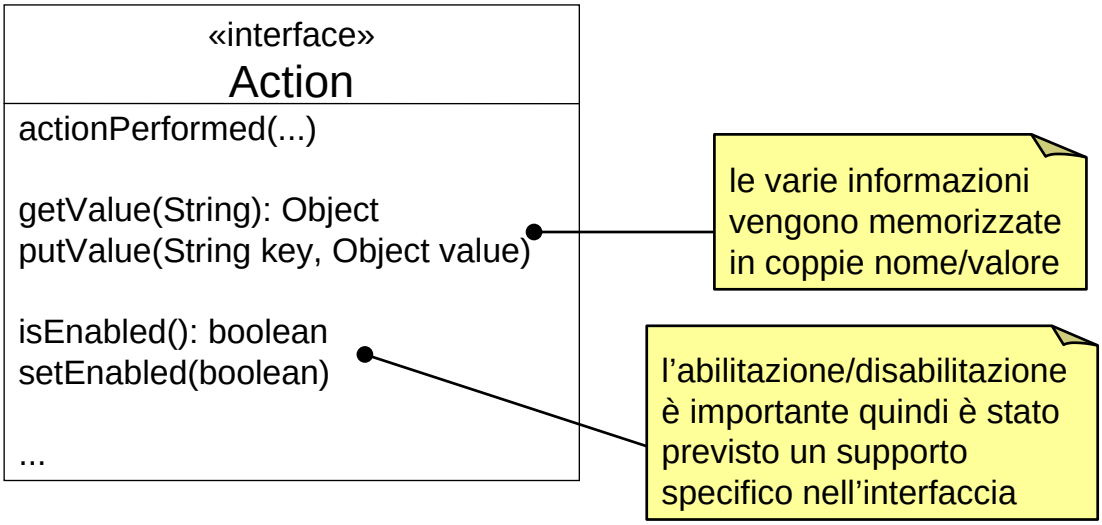


scenario tipico



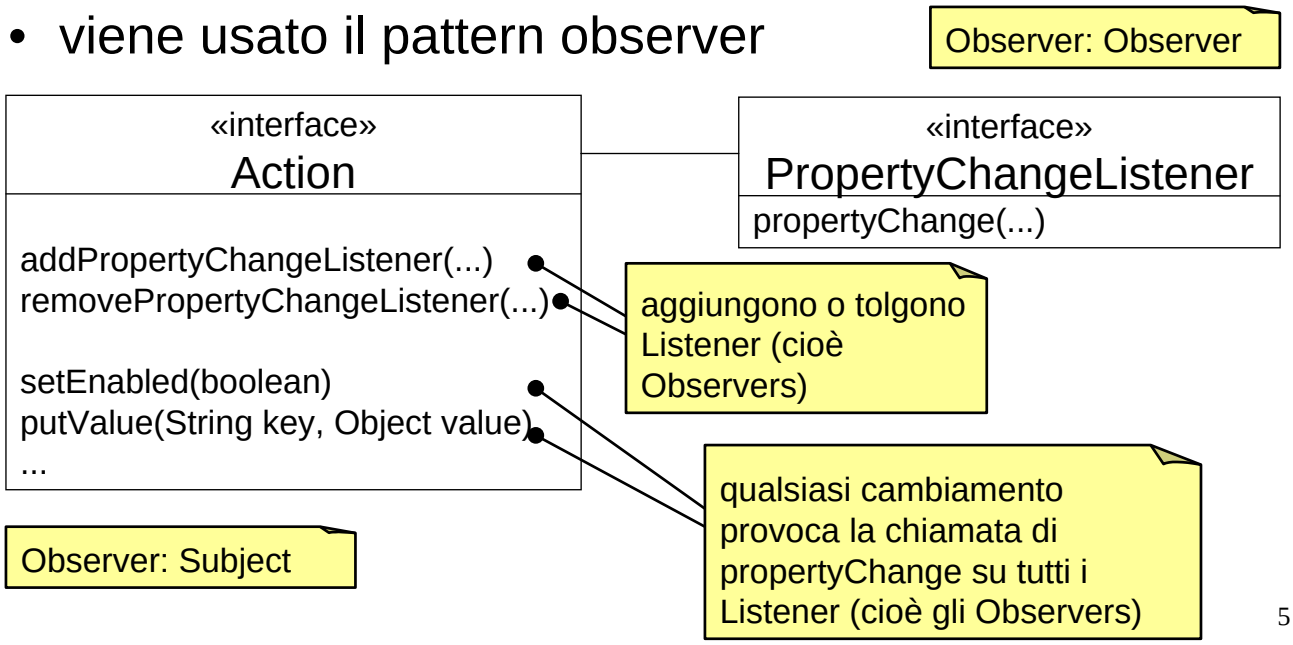
javax.swing.Action

- Action modella anche lo stato dell'azione (abilitata/disabilitata), l'icona associata, il messaggio di tooltip ecc.



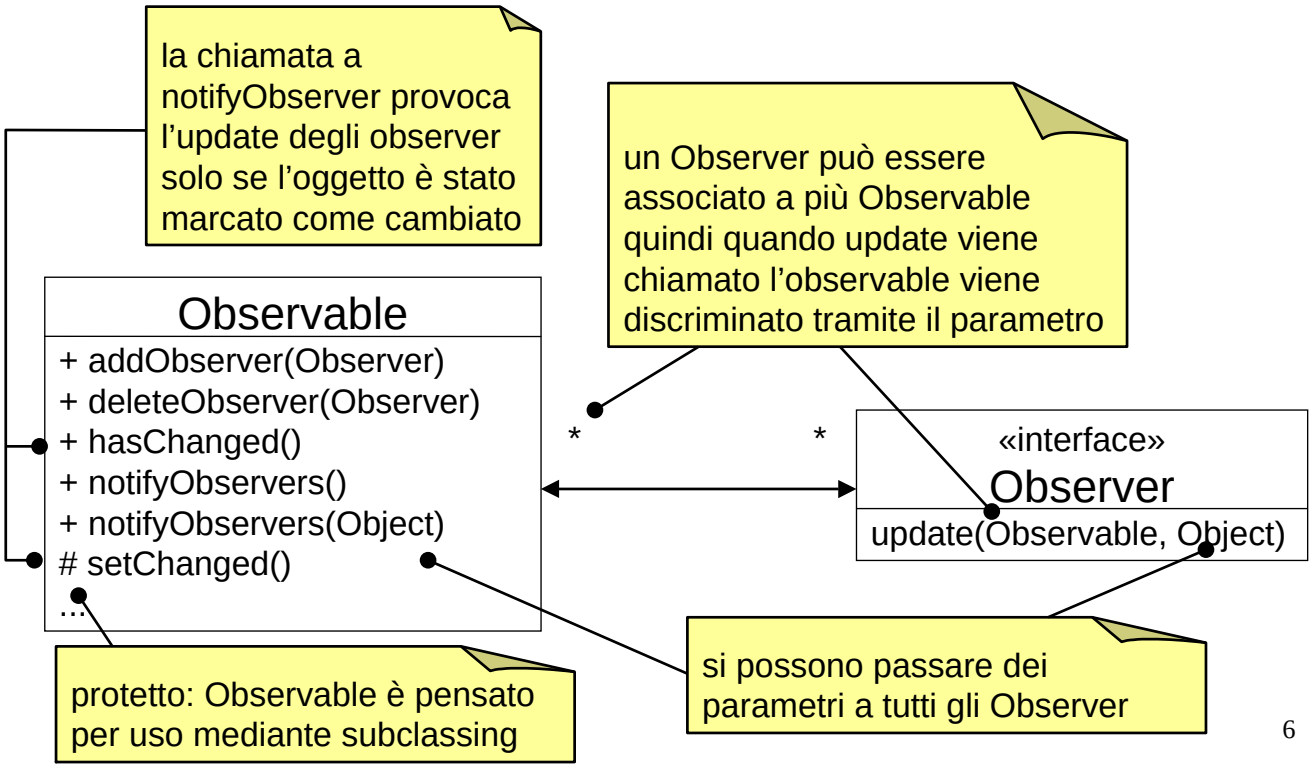
javax.swing.Action

- un cambiamento in un Action si deve riflettere in un cambiamento nei Componenti grafici ad essa associati
- viene usato il pattern observer



Observer in java

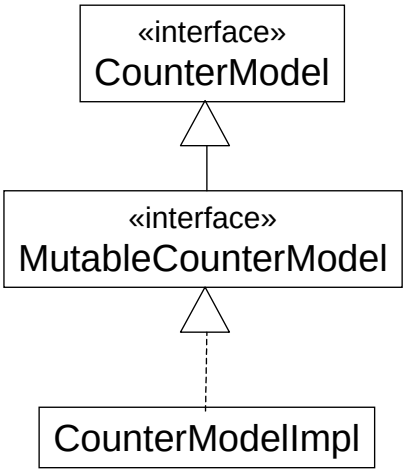
supporto per il pattern Observer nel linguaggio java



un piccolo esempio completo

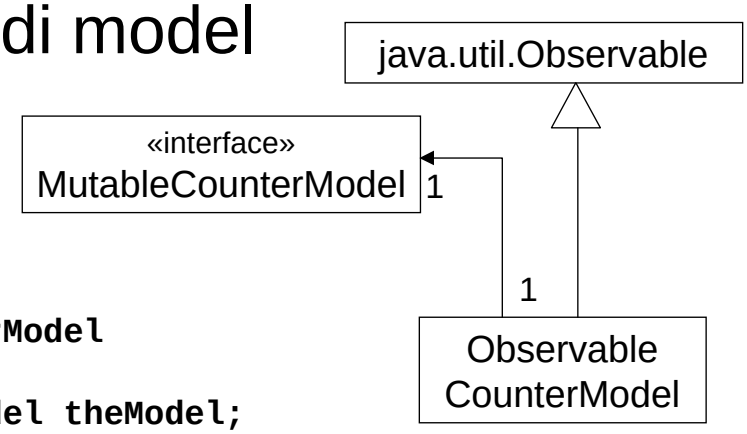
- il modello è un contatore
- i casi d'uso da realizzare nel controller sono
  - incremento
  - doppio incremento
  - rendere dispari il valore mediante un incremento se tale valore non è già dispari
- la view è una interfaccia grafica dotata di una azione per ciascun caso d'uso

model




```
interface CounterModel
    {
    public int getValue();
    }

interface MutableCounterModel
    extends CounterModel
    {
    public void inc();
    }

class CounterModelImpl
    implements MutableCounterModel
    {
    private int v;
    public CounterModelImpl()
        {v=0;}
    public void inc()
        {v++;}
    public int getValue()
        {return v;}
    }
```

osservabilità di model

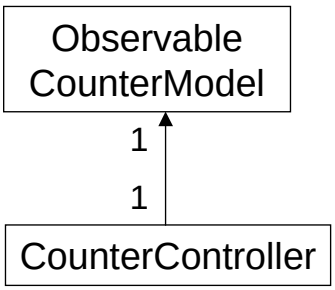



```
class ObservableCounterModel
    extends Observable
    implements MutableCounterModel
    {
    private MutableCounterModel theModel;

    public ObservableCounterModel(MutableCounterModel m)
        {theModel= m;}

    public int getValue()
        {return theModel.getValue();}

    public void inc()
        {
        theModel.inc();
        setChanged();
        }
    }
```

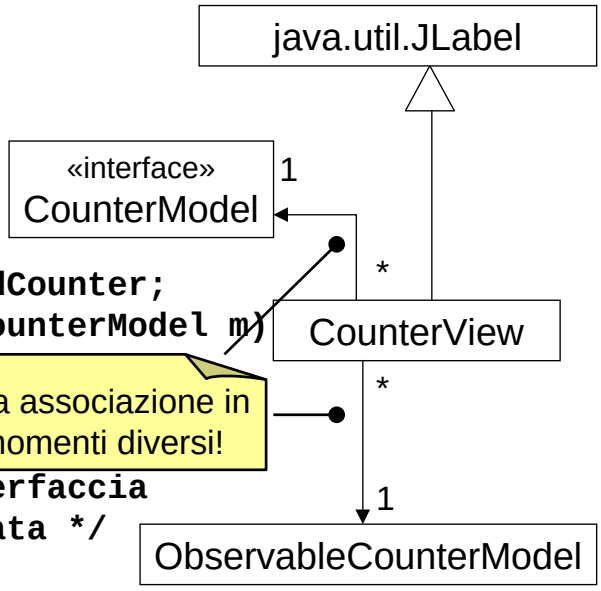


```
class CounterController
    {
    private ObservableCounterModel counter;
    public CounterController(ObservableCounterModel c)
        { counter=c; }
    public void increment()
        {
        counter.inc();
        counter.notifyObservers();
        }
    public void doubleIncrement()
        {
        counter.inc();
        counter.inc();
        counter.notifyObservers();
        }
    public void makeOdd()
        {
        if ( counter.getValue()%2 == 0 )
            {
            counter.inc();
            counter.notifyObservers();
            }
        }
    }
```

controller

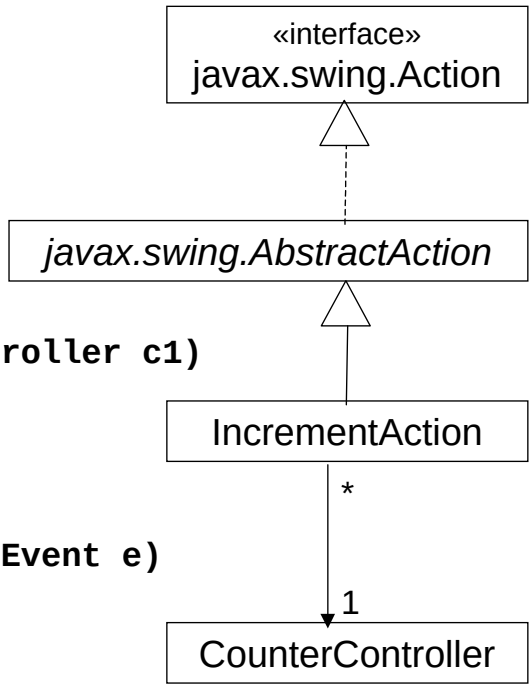


view




```
class CounterView
    extends JLabel
    implements Observer
    {
    private CounterModel theObservedCounter;
    public CounterView( ObservableCounterModel m)
        {
        m.addObserver(this);
        theObservedCounter=m;
        /* da ora in poi solo l'interfaccia
        immutable del modello e' usata */
        update(null,null);
        }
    public void update(Observable dummy1, Object dummy2)
        {
        String s=
        new Integer(theObservedCounter.getValue()).toString();
        setText(s);
        repaint();
        }
    }
```


view: una azione




```
class IncrementAction
   extends AbstractAction
   {
   private CounterController c;
   public IncrementAction(CounterController c1)
        {
        super("Incremento");
        c=c1;
        }
   public void actionPerformed(ActionEvent e)
        {c.increment();}
   }
```

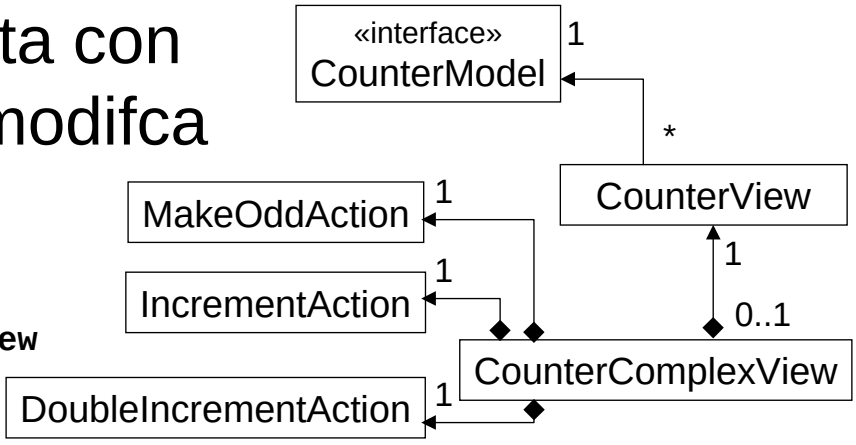
le azioni si distinguono per il nome e per il contenuto di actionPerformed

view: le altre azioni

- la struttura delle altre Action è praticamente la stessa

```
class DoubleIncrementAction extends AbstractAction
    {
    private CounterController c;

    public DoubleIncrementAction(CounterController c1)
        {
        super("Doppio incremento");
        c=c1;
        }

    public void actionPerformed(ActionEvent e)
        {c.doubleIncrement();}
    }


class MakeOddAction extends AbstractAction
    {
    private CounterController c;

    public MakeOddAction(CounterController c1)
        {
        super("Rendi dispari");
        c=c1;
        }

    public void actionPerformed(ActionEvent e)
        {c.makeOdd();}
    }
```


view: una vista con possibilità di modifca




```
class CounterComplexView
   extends JPanel
   {
   private CounterView cv;
   public CounterComplexView(ObservableCounterModel m,
                             CounterController c)
       {
       cv= new CounterView(m);
       setLayout(new GridLayout(4,1));
       add(cv);
       Action a=new IncrementAction(c);
       add(new JButton(a));
       add(new JButton(new DoubleIncrementAction(c)));
       add(new JButton(new MakeOddAction(c)));
       }
   }
```

main

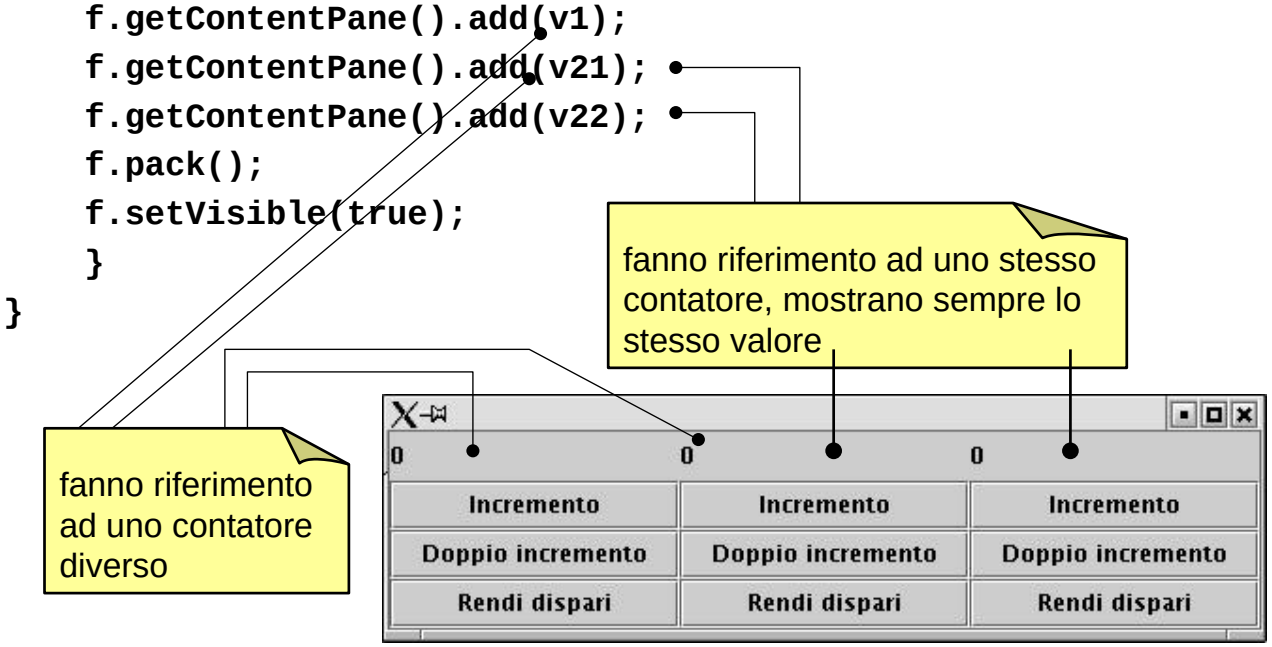

```
public static void main(String args[])
    {
    // Model
    ObservableCounterModel m1=
        new ObservableCounterModel(
            new CounterModelImpl());
    // Controller
    CounterController c1=new CounterController(m1);
    // View
    CounterComplexView v1=new CounterComplexView(m1,c1);
    // GUI setup
    JFrame f= new JFrame();
    f.getContentPane().add(v1);
    f.pack();
    f.setVisible(true);
    }
```


main

```
public static void main(String args[])
    {
    // Un model con un controller e una vista
    ObservableCounterModel m1=
        new ObservableCounterModel(
            new CounterModelImpl());
    CounterController c1=new CounterController(m1);
    CounterComplexView v1=new CounterComplexView(m1,c1);

    // un modello con un controlle e due viste
    ObservableCounterModel m2=
        new ObservableCounterModel(
            new CounterModelImpl());
    CounterController c2=new CounterController(m2);
    CounterComplexView v21=new CounterComplexView(m2,c2);
    CounterComplexView v22=new CounterComplexView(m2,c2);

    ...
```

main

```
    ...
    // GUI setup
    JFrame f= new JFrame();
    f.getContentPane().setLayout(new GridLayout(1,3));
    f.getContentPane().add(v1);
    f.getContentPane().add(v21);
    f.getContentPane().add(v22);
    f.pack();
    f.setVisible(true);
    }
}
```



variante: ComplexCounterView con Lock

- introduciamo un nuovo bottone che inibisce la possibilità di incremento singolo
- facciamo questo sfruttando il fatto che i JComponenti sono observer di Action
- ci aspettiamo che disabilitando l'azione il bottone corrispondente cambi stato

una nuova azione: LockUnlockAction

- LockUnlockAction agisce su una azione *c* abilitandola o disabilitandola
- varia il proprio nome coerentemente
  - quando *c* è abilitata il nome è “Lock”
  - quando *c* è disabilitata il nome è “Unlock”

```
class LockUnlockAction extends AbstractAction
    {
    private Action c;
    public LockUnlockAction(Action c1)
        {
        c=c1;
        updateName(); // alinea il proprio nome con lo stato di c
  }
   ....
```


una nuova azione: LockUnlockAction

- quando LockUnlockAction è invocata verifica lo stato dell’azione a cui è associata e
  - ne inverte lo stato
  - allinea il proprio nome con la situazione attuale

```
   ....
   public void actionPerformed(ActionEvent e)
      {
      c.setEnabled(!c.isEnabled());
      updateName();
      }
   private void updateName()
      {
      if (  c.isEnabled() )
          putValue(Action.NAME, "Lock");
      else
          putValue(Action.NAME, "Unlock");
      }
  }
```

i due possibili stati