# Esercitazioni Ing.Sw

Alessandro Margara / Gian Enrico Conti

II sem 2023

Webex:

https://politecnicomilano.webex.com/meet/gianenrico.conti

gianenrico.conti@polimi.it

Esercitazione 2

# Requirements

ing. Gian Enrico Conti

Prova Finale - Ingegneria del Software

# Requirements

- PDF già su WeBeep (https://webeep.polimi.it/pluginfile.php/946303/mod_folder/content/0/requirements.pdf?forcedownload=1)

- Timeline:
  - Data di consegna: venerdì 30 giugno 2023 13:00:00 CEST
  - Data di valutazione: da fssare (a partire da lunedì 3 luglio 2023)
  - Modalità di valutazione, aule e orari, verranno comunicati successivamente.

# Shop List

- diagramma UML iniziale dell'applicazione (ad alto livello);
- diagrammi UML finali
- implementazione **funzionante** del gioco conforme alle regole del gioco e alle specifiche presenti nel PDF
- codice sorgente dell'implementazione;
- codice sorgente dei test di unità.

DOVE?

# Requisiti  Game-specific

- Regole Semplifcate: NON comprende le carte obiettivo comune e i relativi punteggi (le carte obiettivo personale sono invece incluse)

- .Regole Complete: tutte le regole per lo svolgimento di *normali partite (*come indicato nel manuale del gioco)

- L'univocità del nickname deve essere garantita dal server in fase di accettazione del giocatore.

POLITECNICO MILANO 1863

# Requisiti  Game-agnostic

- un sistema  distribuito
- 1 singolo  server in grado di gestire una partita alla volta
- multipli client (uno per giocatore) (una sola partita alla volta.
- pattern MVC (Model-View-Controller) per progettare l’intero sistema.

POLITECNICO MILANO 1863

# Server

- JavaSE
- TCP  (i.e. Socket) e / o RMI
- Se entrambi, le partite devono permettere che i diversi giocatori utilizzano tecnologie diverse
- le specifiche **non** richiedono cambiamento “on the fly” del protocollo di comunicazione per un client
- Deve essere instanziato una sola volta al fine di gestire una singola partita

  (tranne nel caso in cui venga implementata la funzionalità avanzata “partite multiple”).

- UDP.. HTTP.. Rest ... ??

POLITECNICO MILANO 1863

# Client

- JavaSE
- instanziabile più volte (una per giocatore), **anche sulla**
- **stessa macchina.**
- L’interfaccia grafica Swing o JavaFX.
- SE TUI + GUI all’avvio, deve permettere al giocatore di selezionare il tipo di interfaccia
- Idem x scelta protocollo

POLITECNICO MILANO 1863

# Client: Connessione

- Se non ci sono partite in fase di avvio, viene creata una nuova partita, altrimenti l'utente entra automaticamente a far parte della partita in fase di avvio.
- Il giocatore che crea la partita sceglie il numero di giocatori che ne fanno parte.
- Se c'è una partita in fase di avvio, il giocatore viene automaticamente aggiunto alla partita.
- La partita inizia non appena si raggiunge il numero di giocatori atteso (in base alla scelta effettuata dal primo giocatore in fase di creazione della partita).

POLITECNICO MILANO 1863

# Client: Connessione/disconessione/fault

- Il server consente ai vari giocatori di svolgere i propri turni secondo le regole del gioco.
- È necessario gestire sia il caso in cui i giocatori escano dalla partita, sia il caso in cui **cada** la connessione di rete.

*In entrambi i casi la partita dovrà terminare e tutti i giocatori verranno notificati.*

POLITECNICO MILANO 1863

# F. Avanzate

- Vedi PDF...
- Le funzionalità avanzate facoltative
- Se mancano F. "base" NON verranno valutate

# Valutazione

| Requisiti Soddisfatti | Voto Massimo |
|---|---|
| Regole Semplificate + TUI + RMI o Socket | 18 |
| Regole Complete + TUI + RMI o Socket | 20 |
| Regole Complete + TUI + RMI o Socket + 1 FA | 22 |
| Regole Complete + TUI + GUI + RMI o Socket + 1 FA | 24 |
| Regole Complete + TUI + GUI + RMI + Socket + 1 FA | 27 |
| Regole Complete + TUI + GUI + RMI + Socket + 2 FA | 30 |
| Regole Complete + TUI + GUI + RMI + Socket + 3 FA | 30L |

Tabella 1: Tabella di valutazione (FA=Funzionalità avanzata)

POLITECNICO MILANO 1863

# Valutazione (PDF)

- La qualità della progettazione, con particolare riferimento ad un uso appropriato di interfacce, ereditarietà, composizione tra classi, uso dei design pattern (statici, di comunicazione e architetturali) e divisione delle responsabilità.

- La stabilità dell'implementazione e la conformità alle specifiche.

- La leggibilità del codice scritto, con particolare riferimento a nomi di variabili/metodi/classi/package, all'inserimento di commenti in inglese e documentazione JavaDoc in inglese, la mancanza di codice ripetuto e metodi di eccessiva lunghezza.

- L'efficacia e la copertura dei casi di test, il nome e i commenti di ogni test dovranno chiaramente specificare le funzionalità testate e i componenti coinvolti.

- L'utilizzo degli strumenti (IntelliJ IDEA, Git, Maven, ...).

- L'autonomia, l'impegno e la comunicazione (con i responsabili e all'interno del gruppo) durante tutte le fasi del progetto.

POLITECNICO MILANO 1863

Esercitazione 2

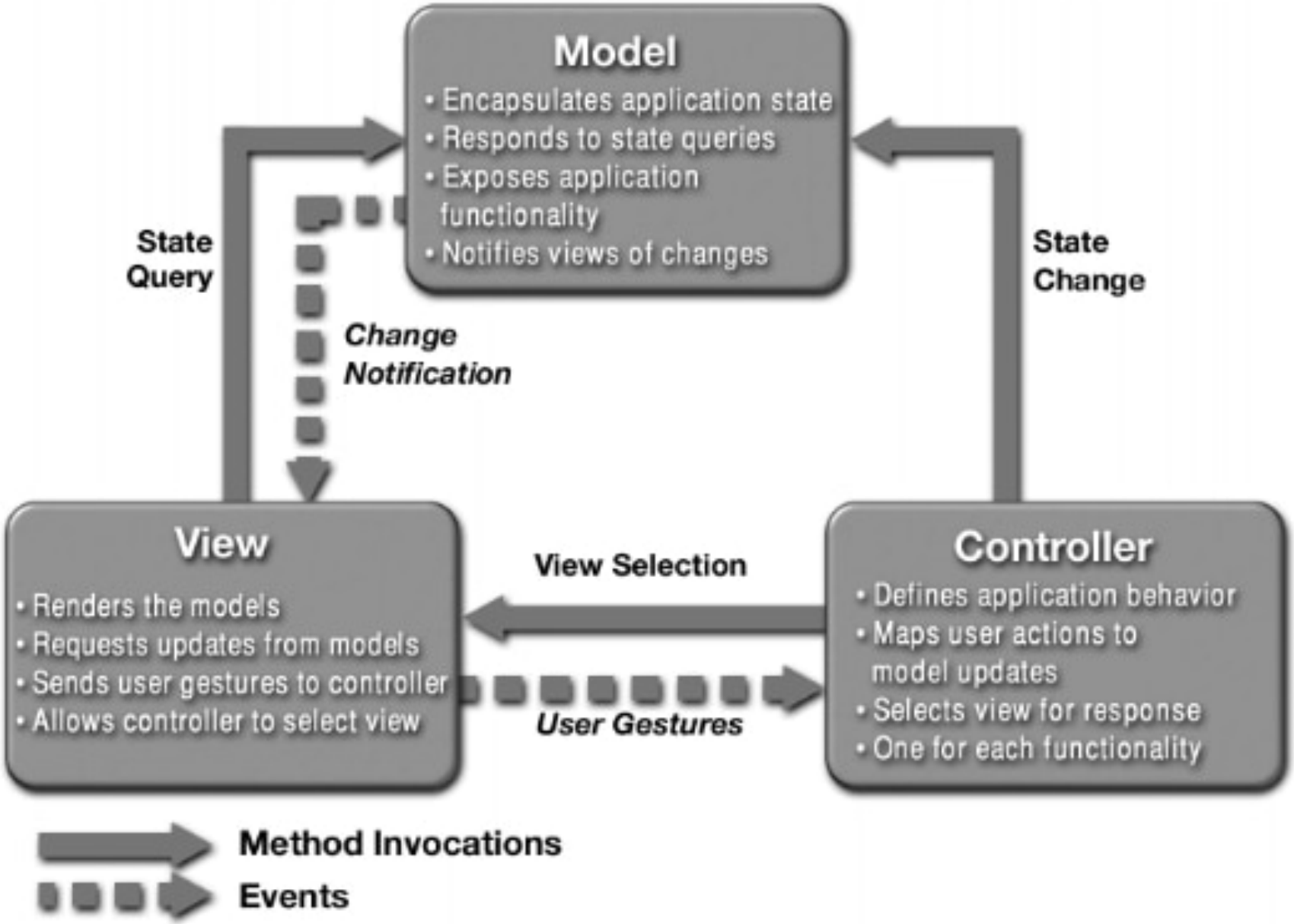
# MVC

ing. Gian Enrico Conti

# Model-View-Controller

- ***Pattern architetturale*** che separa il modello dei dati di una applicazione dalla rappresentazione grafica (view) e dalla logica di controllo (controller)

# MVC Oracle/JEEE



# MVC History:

Copyright © 1988 ParcPlace Systems. All Rights Reserved.

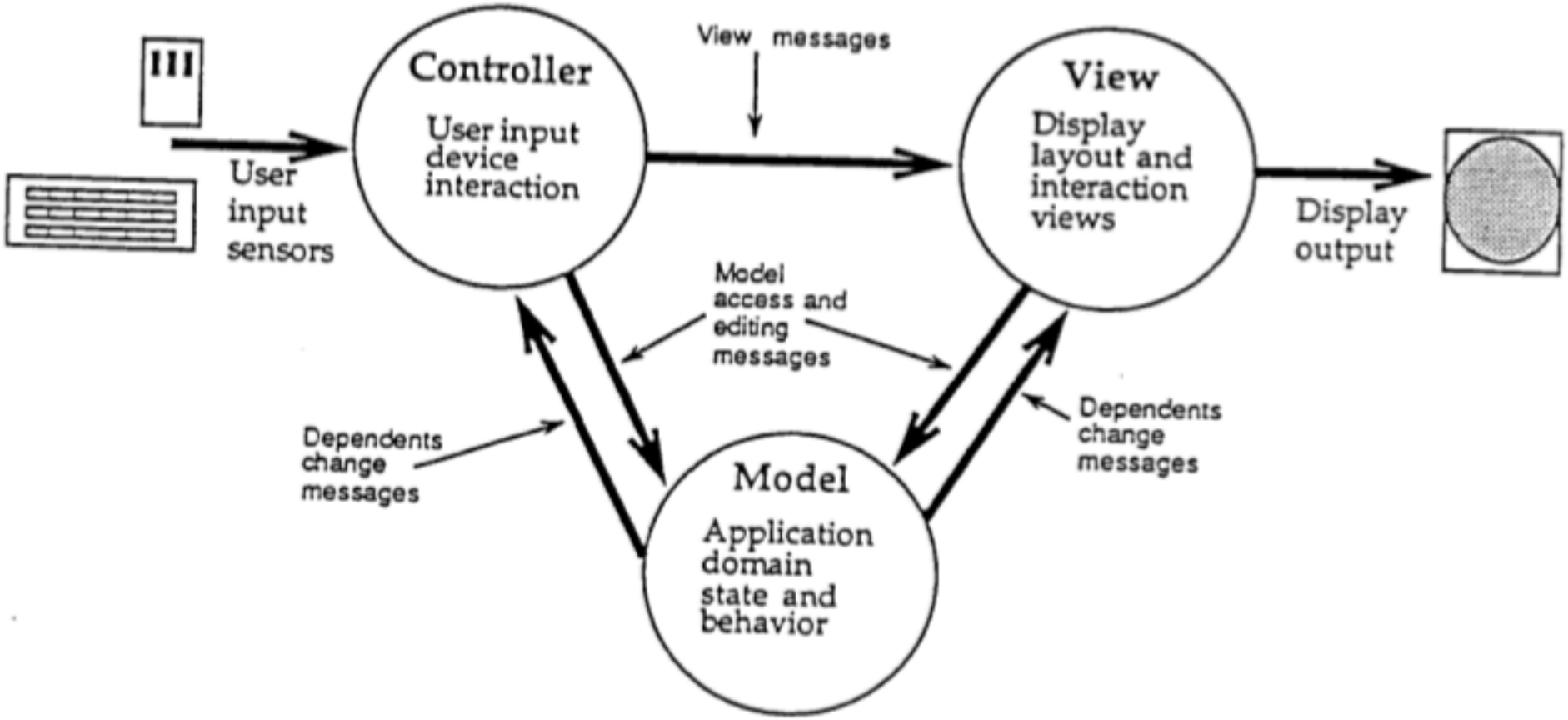
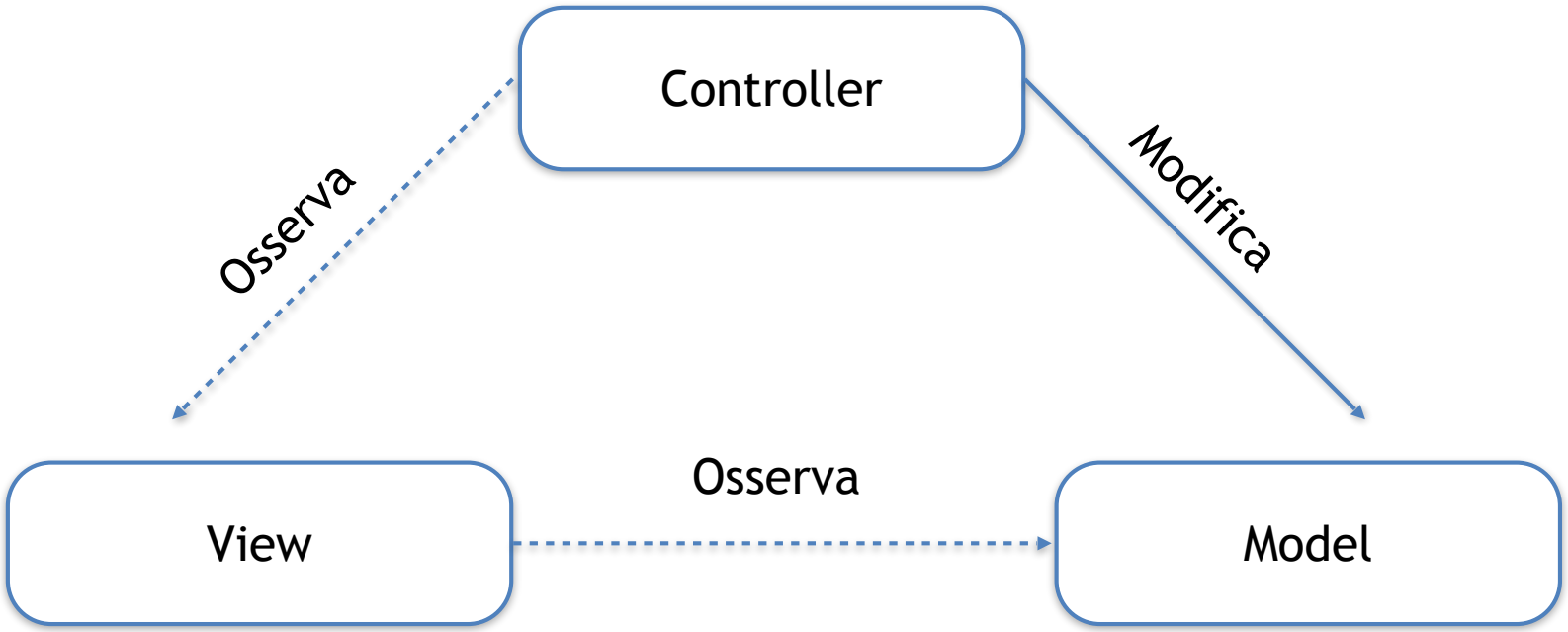



**Figure 1: Model-View-Controller State and Message Sending**

# Model-View-Controller: Now

- ***Pattern architetturale*** che separa il modello dei dati di una applicazione dalla rappresentazione grafica (view) e dalla logica di controllo (controller)



# MVC Oracle/JEEE Model

- **Model** - The model represents enterprise data and the business rules that govern access to and updates of this data. Often the model serves as a software approximation to a real-world process, so simple real-world modeling techniques apply when defining the model.

POLITECNICO MILANO 1863

# MVC Oracle/JEEE definitions

- **View** -The view renders the contents of a model. It accesses enterprise data through the model and specifies how that data should be presented. It is the view's responsibility to maintain consistency in its presentation when the model changes. This can be achieved by using a push model, where the view registers itself with the model for change notifications, or a pull model, where the view is responsible for calling the model when it needs to retrieve the most current data.

POLITECNICO MILANO 1863

# MVC Oracle/JEEE definitions

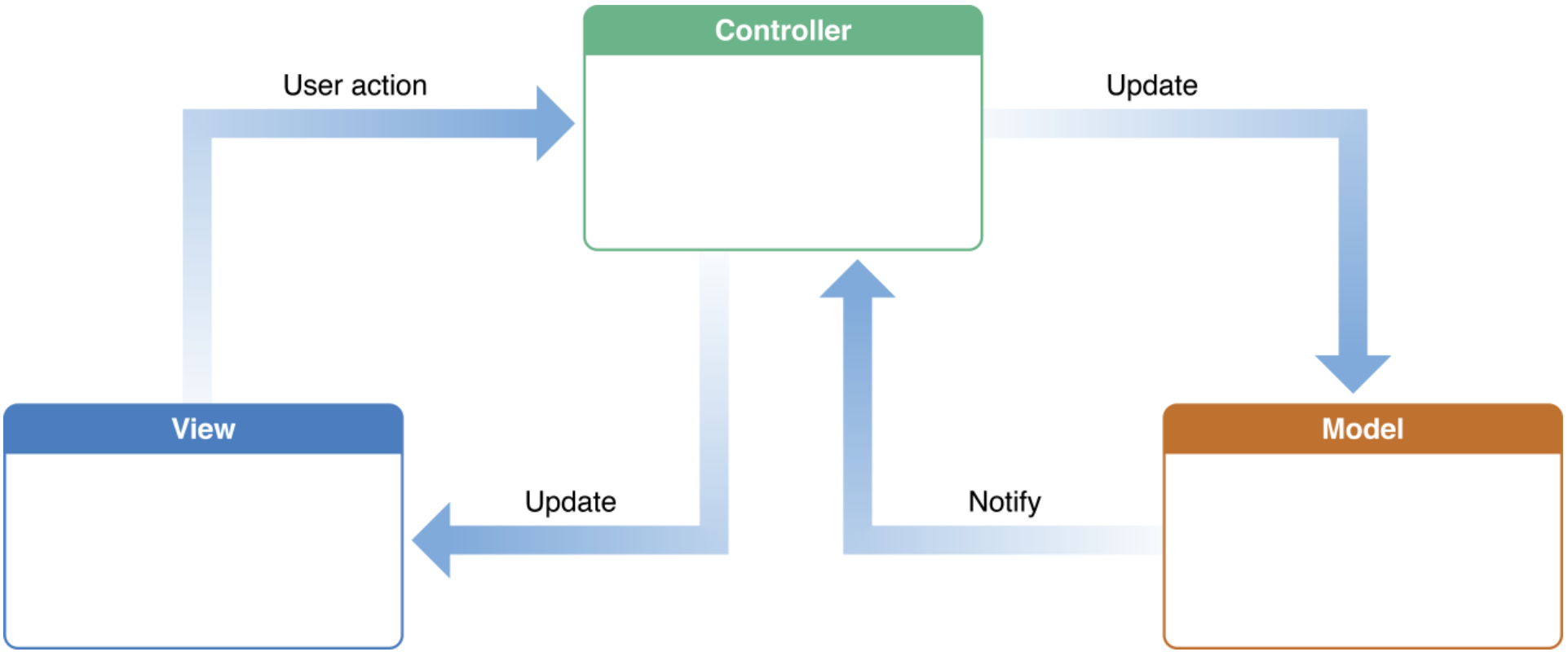
- **Controller** - The controller translates interactions with the view into actions to be performed by the model.

- In a stand-alone GUI client, *user interactions could be button clicks or menu selections*
- in a Web application, they appear as GET and POST HTTP requests. The actions performed by the model include activating business processes or changing the state of the model.

- Based on the user interactions and the outcome of the model actions, the controller *responds by selecting an appropriate view.*

POLITECNICO MILANO 1863

# Model-View-Controller: Varianti

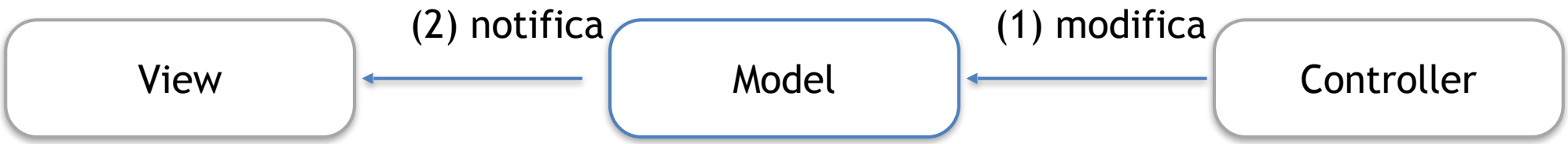



POLITECNICO MILANO 1863

# Model

- ***Incapsula lo <u>stato</u>*** *di una applicazione*
  - *contiene le classi che descrivono i dati dell'applicazione e le operazioni per manipolarli*

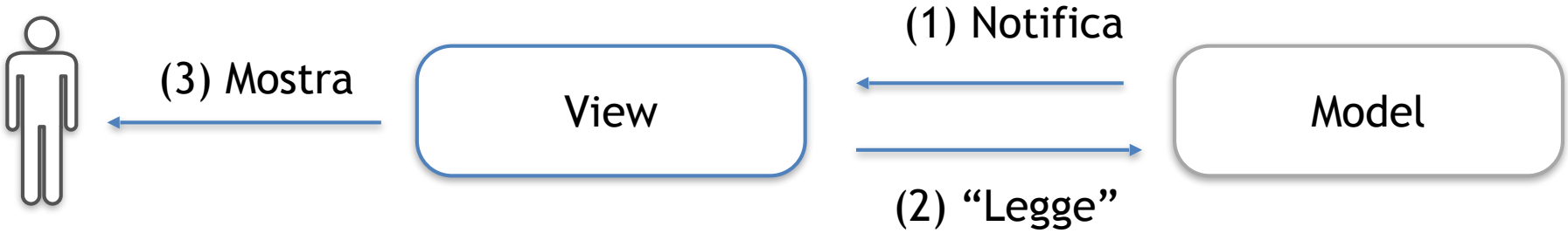
- *Fornisce i metodi per* ***accedere*** ai dati
- ***<u>Notifica</u>*** i cambiamenti di stato (alla view)



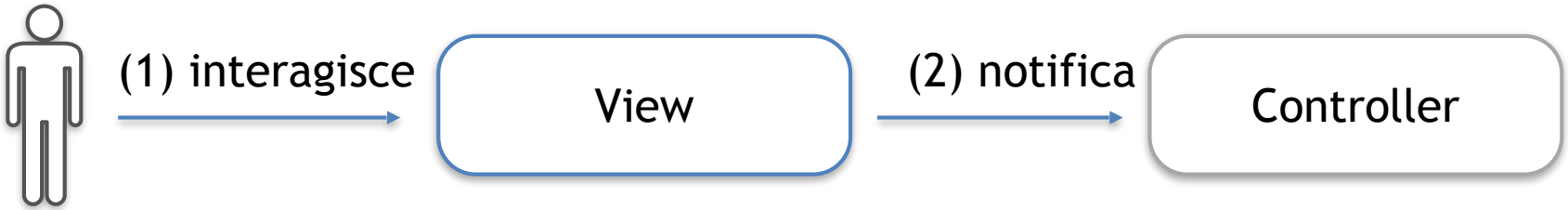

POLITECNICO MILANO 1863

# View

- ***Mostra*** *lo stato dell'applicazione (modello)*




- *Gestisce* ***l'interazione*** *con l'utente*

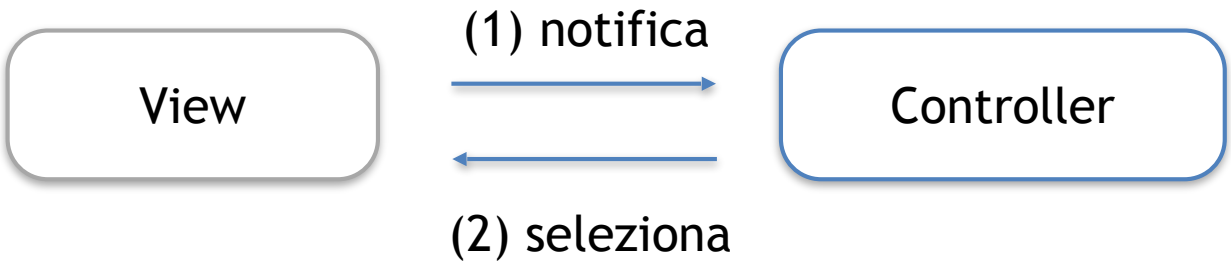



POLITECNICO MILANO 1863

# Controller

- *Rappresenta la **logica applicativa***
- *Collega le **azioni** dell'utente con **modifiche allo stato***


- *Sceglie cosa deve essere mostrato*



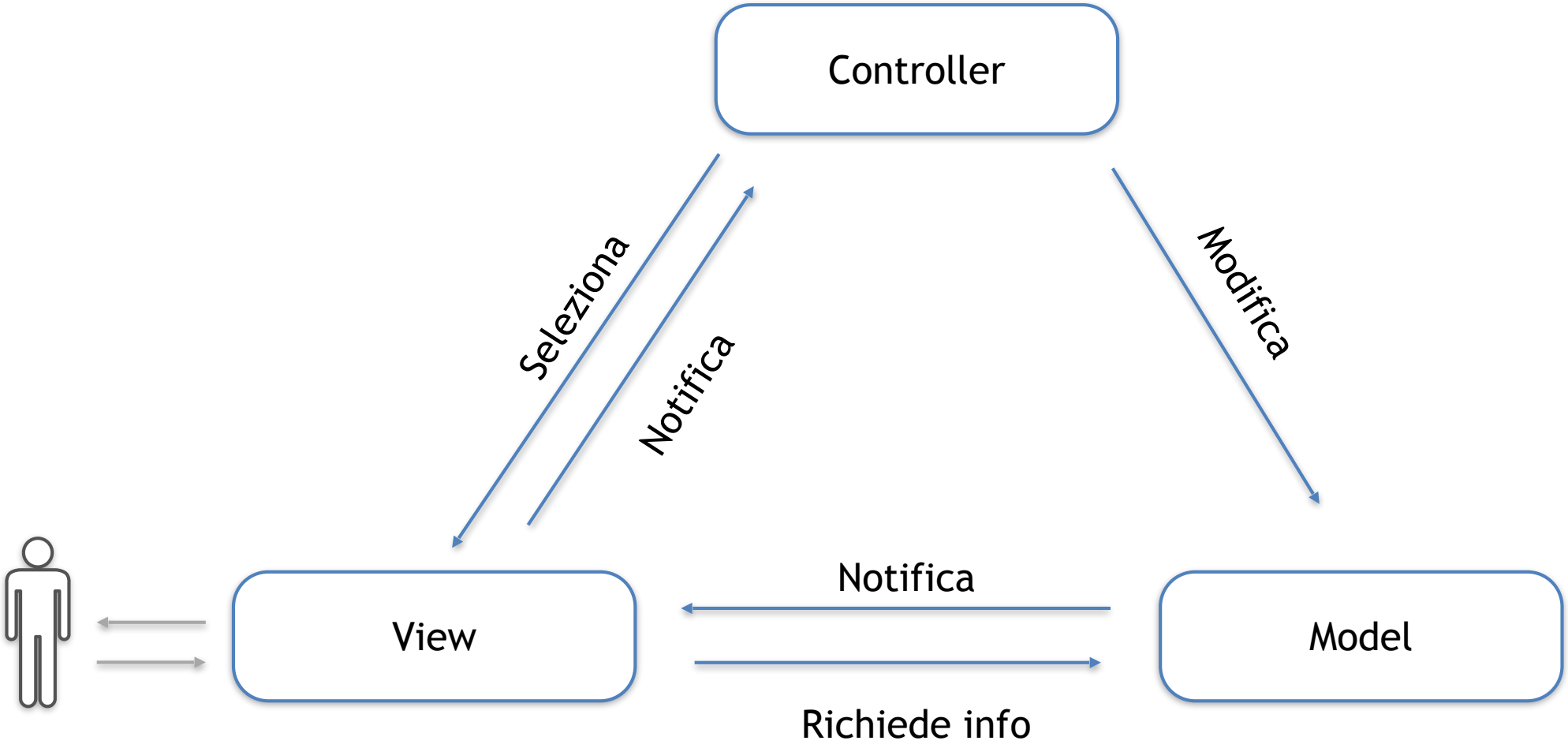

POLITECNICO MILANO 1863

# View - Controller

- *La view deve rendere accessibile al controller metodi generici / comportamentali (es. showMessage, resetBoard, showFinalScores … ). Il controller non deve mai intervenire direttamente sugli elementi grafici.*

POLITECNICO MILANO 1863

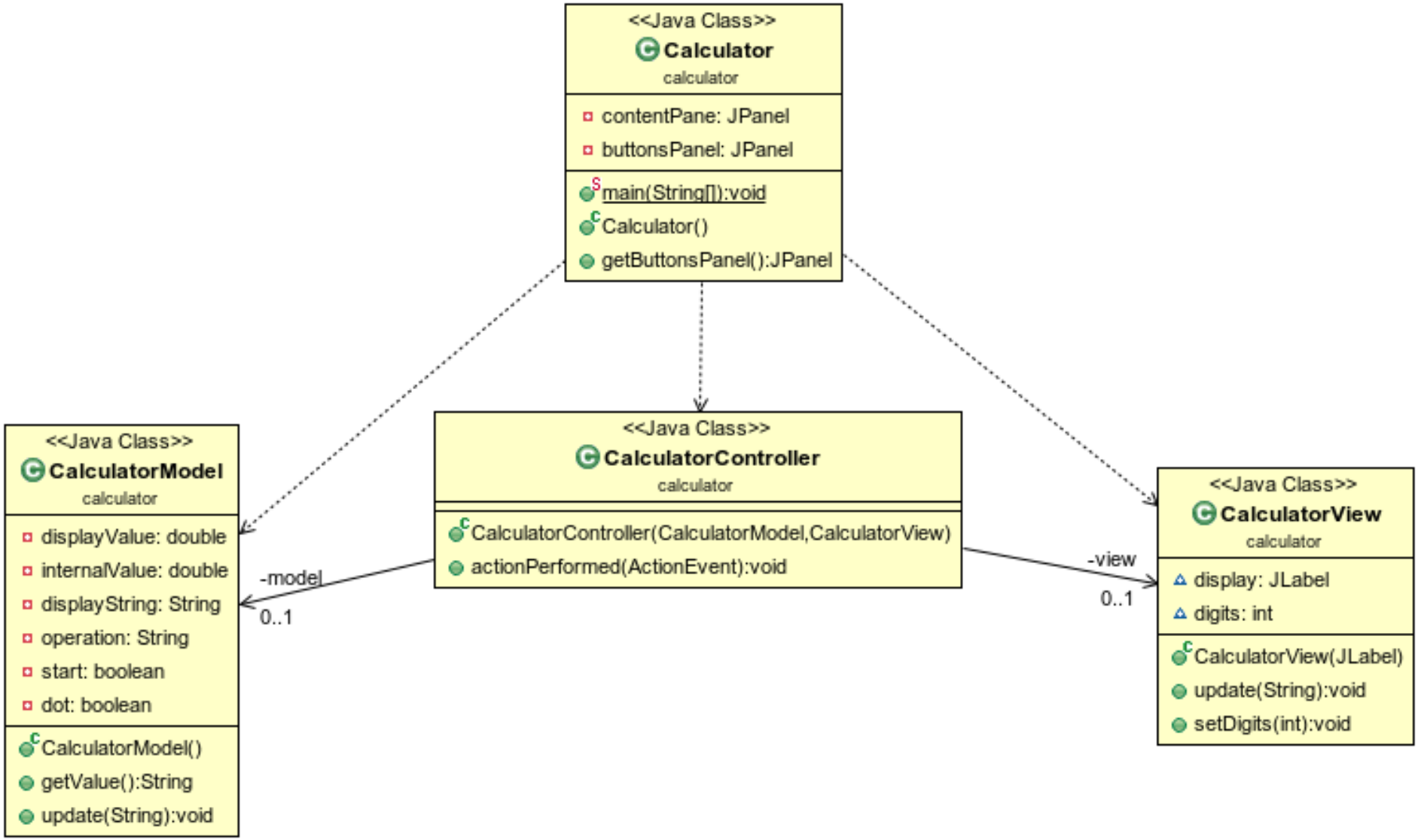
# Model-View-Controller




**Si.. ma in pratica??**

POLITECNICO MILANO 1863

# Esempio


<<Java Class>>
Calculator
calculator
contentPane: JPanel
buttonsPanel: JPanel
main(String[]):void
Calculator()
getButtonsPanel():JPanel
<<Java Class>>
CalculatorModel
calculator
displayValue: double
internalValue: double
displayString: String
operation: String
start: boolean
dot: boolean
CalculatorModel()
getValue():String
update(String):void
<<Java Class>>
CalculatorController
calculator
CalculatorController(CalculatorModel,CalculatorView)
actionPerformed(ActionEvent):void
<<Java Class>>
CalculatorView
calculator
display: JLabel
digits: int
CalculatorView(JLabel)
update(String):void
setDigits(int):void
-model
0..1
-view
0..1

# Un pezzettino di codice

```
public class CalculatorController implements ActionListener {
  private CalculatorModel model;
  private CalculatorView view;

  public CalculatorController(CalculatorModel model, CalculatorView view) {
    this.model = model;
    this.view = view;
  }
  …
```

La classe controller instanzia model e view.
(La App instanzia il controller.)

# Ns caso:

```
public class GameController implements ActionListener {
   private GameModel model;
   private GameView view;

 public GameController(GameModel model, GameView view) {
      this.model = model;
      this.view = view;
   }
//or:
 public GameController) {
      this.model = new GameModel();
      this.view = new GameView();
   }
```

La classe controller instanzia model e view.

(La App instanzia il controller, oppure in casi semplici e' la App stessa...)

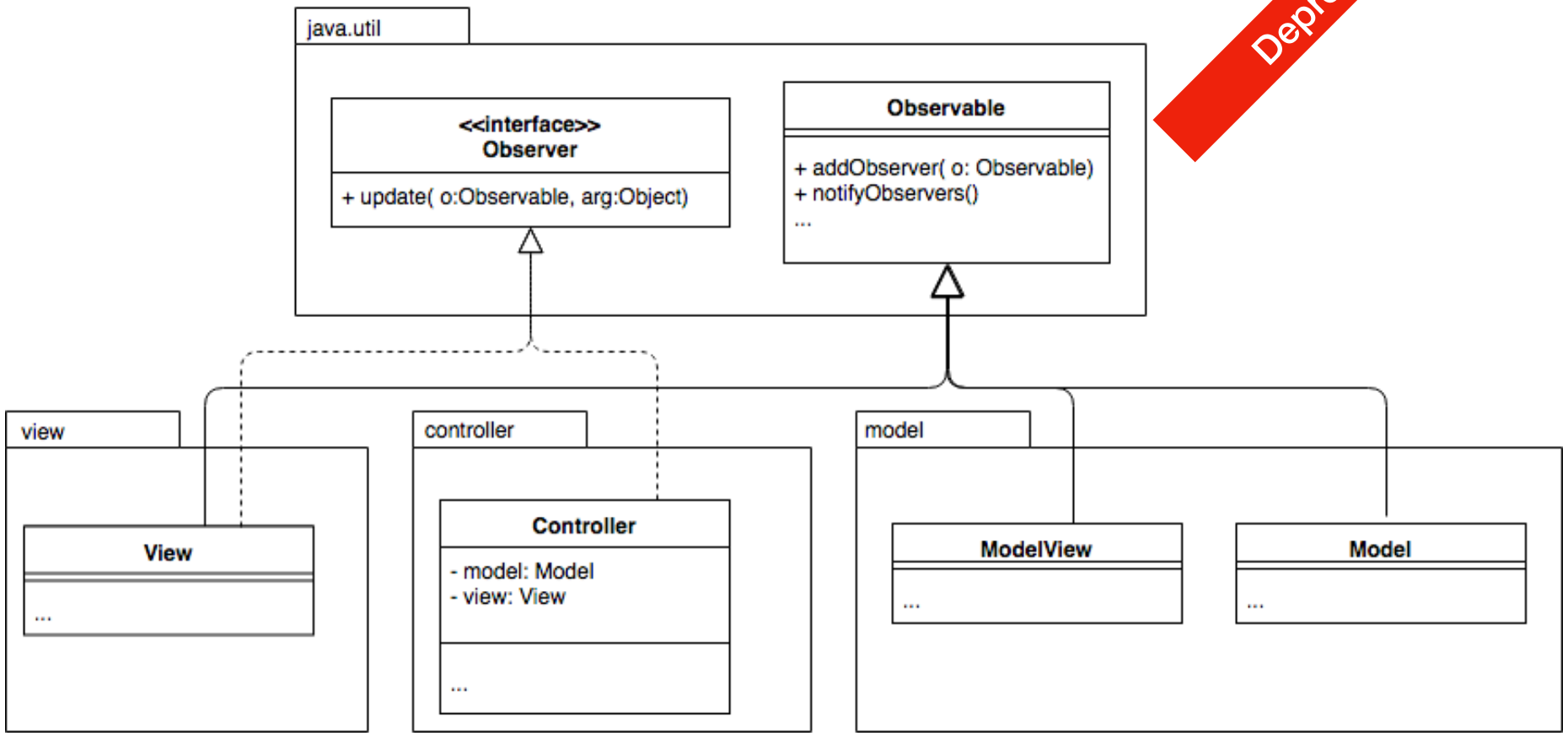
# Come si realizza?

- *Utilizzo della **programmazione ad eventi***
- *Un oggetto sorgente genera eventi*
- *Un ascoltatore registrato viene notificato degli eventi*

# Come si realizza?

- ....

# Observer e Observable




POLITECNICO MILANO 1863

# Programmazione ad eventi

**EVENTI**

- *Azioni dell'utente o notifiche dal model al controller*
- *Sono classi che contengono informazioni dettagliate sull'evento*

**ASCOLTATORI ( Listeners )**

- *Si mettono in ascolto di un evento*
- *Devono avere dei metodi per poter reagire agli eventi*
- *Possono esserci più ascoltatori per un evento*

**SORGENTI DI EVENTI**

- *Notificano gli eventi agli interessati*
- *La notifica avviene invocando i metodi sugli ascoltatori*
- *Devono avere un metodo per permettere la registrazione degli ascoltatori*

POLITECNICO MILANO 1863

# Passaggio Oggetti

- ***Sempre*** *oggetti che rappresentano eventi o creati appositamente*
- *Oggetti modificabili del modello* ***non devono*** *arrivare all'interfaccia*

**Soluzioni:**

- ***Interfacce limitate*** *(solo metodi per la visualizzazione delle informazioni selezionate)*
- ***Oggetti immutabili***
- *Oggetti creati appositamente per la visualizzazione*

# MVC + Rete

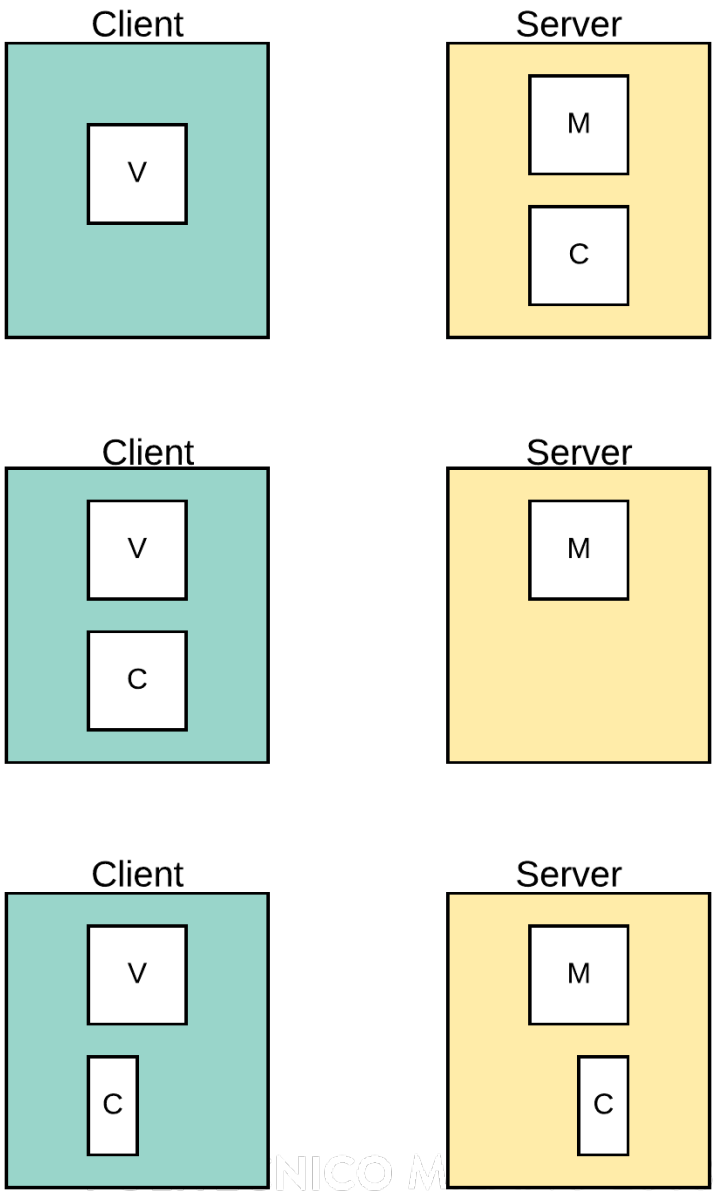
- *Non far confusione tra MVC e network (separazione dei task)*
- *Non hardcodare le tecnologie di rete (estensibilità del codice)*
- *Nascondere la gestione della rete alla classi MVC (encapsulation)*

# “Distributed MVC”

*Come distribuisco M/V/C in un’ applicazione client-server?*

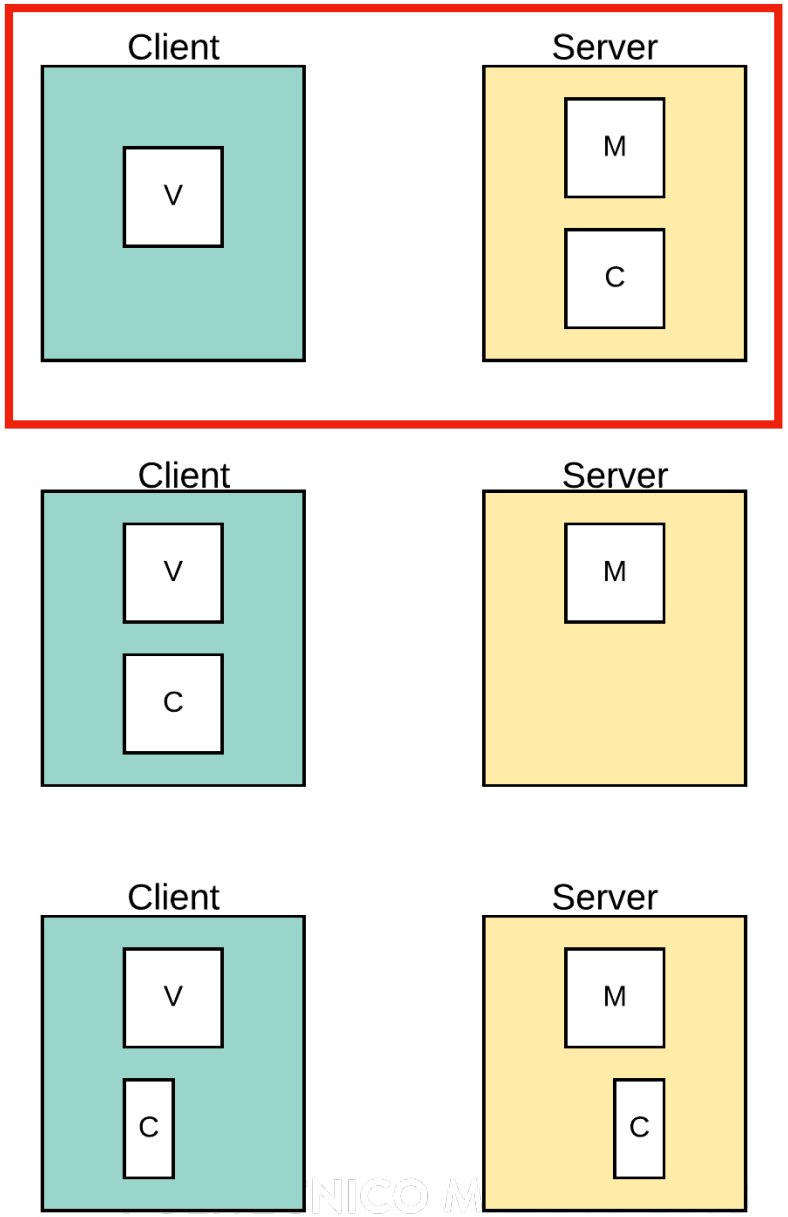
# “Distributed MVC”

**Differenti approcci..**

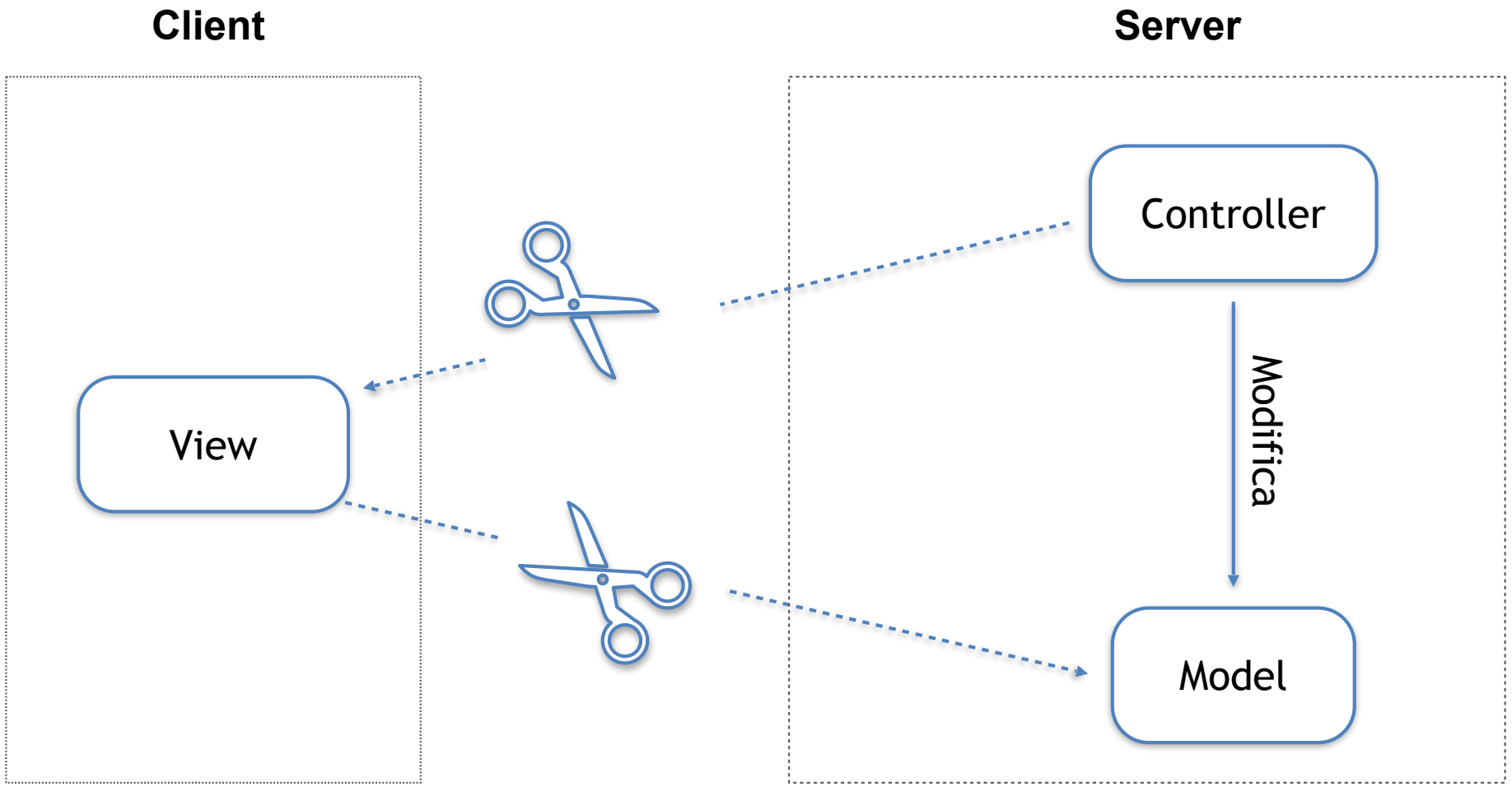


# “Distributed MVC”

**L’ anno scorso 1..**

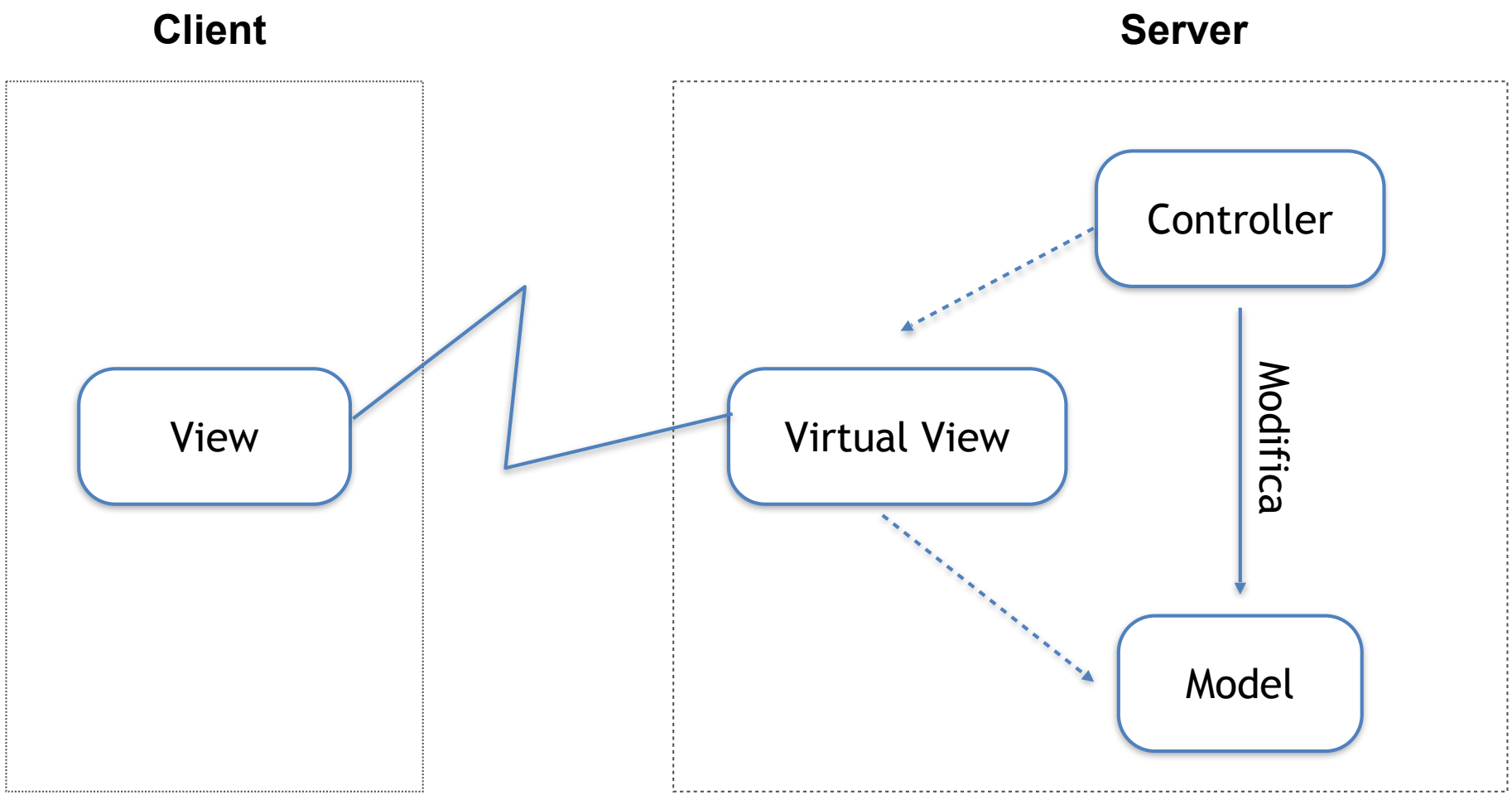


# MVC + Rete




POLITECNICO MILANO 1863

# MVC + Rete




POLITECNICO MILANO 1863

# MVC + Rete

Virtual View

- *Come una normale View è **Observer** (**LISTENER..**) del model*
- *Come una normale View è **Observable (manda eventi..)** del (AL) controller*
- ***Agisce sulla rete** “all’insaputa” di model e controller*
  - *Inoltra gli eventi ricevuti dal modello attraverso la rete (alla view del client)*
  - *Riceve eventi dalla view del client attraverso la rete e li inoltra al controller*

POLITECNICO MILANO 1863

# “Distributed MVC: considerations”

**network**: quanti dati/richieste devono essere spediti tra client e server?

**Reattivita**’(responsiveness): una reattivita’ piu’ alta con thick client

**Sicurezza**: non ho controllo client, quando dati escono da un perimetro sicuro (server) aumentano i rischi
performance: per performance elevate, sposto nel server

**Load balancing**: posso spostare computazione (parte del control) nel client per scaricare il server.

# Ok, da dove parto?

Per quanto detto:

- Separazione dei task
- Astrazione
- Modello indipendente da GUI/CLI/network..
- Network "agnostico" su contenuto
- Modello e test...

Quindi Parallelizzare non solo si può**... si deve..**

# Per non farci mancare nulla...

## JAVADOC

# Inserire un commento Javadoc (0)

Un commento Javadoc è nella forma

```
/**
 * Brief my comments
 */

public static void main( String[] args )
{
...
```

# Scrivere Javadoc (1)

PLS commenti "sensati":

```
/**
 * "domain" test. we are testing if exiting from domain is safe
 */
@Test
public void negTest()
{
    Calculator c = new Calculator();
    assertTrue( c.Factorial(-1) >  0 );
}
```

```
/**
 * provo se va tutto */
@Test
public void negTest()
...
```

# Inserire un commento Javadoc (2)

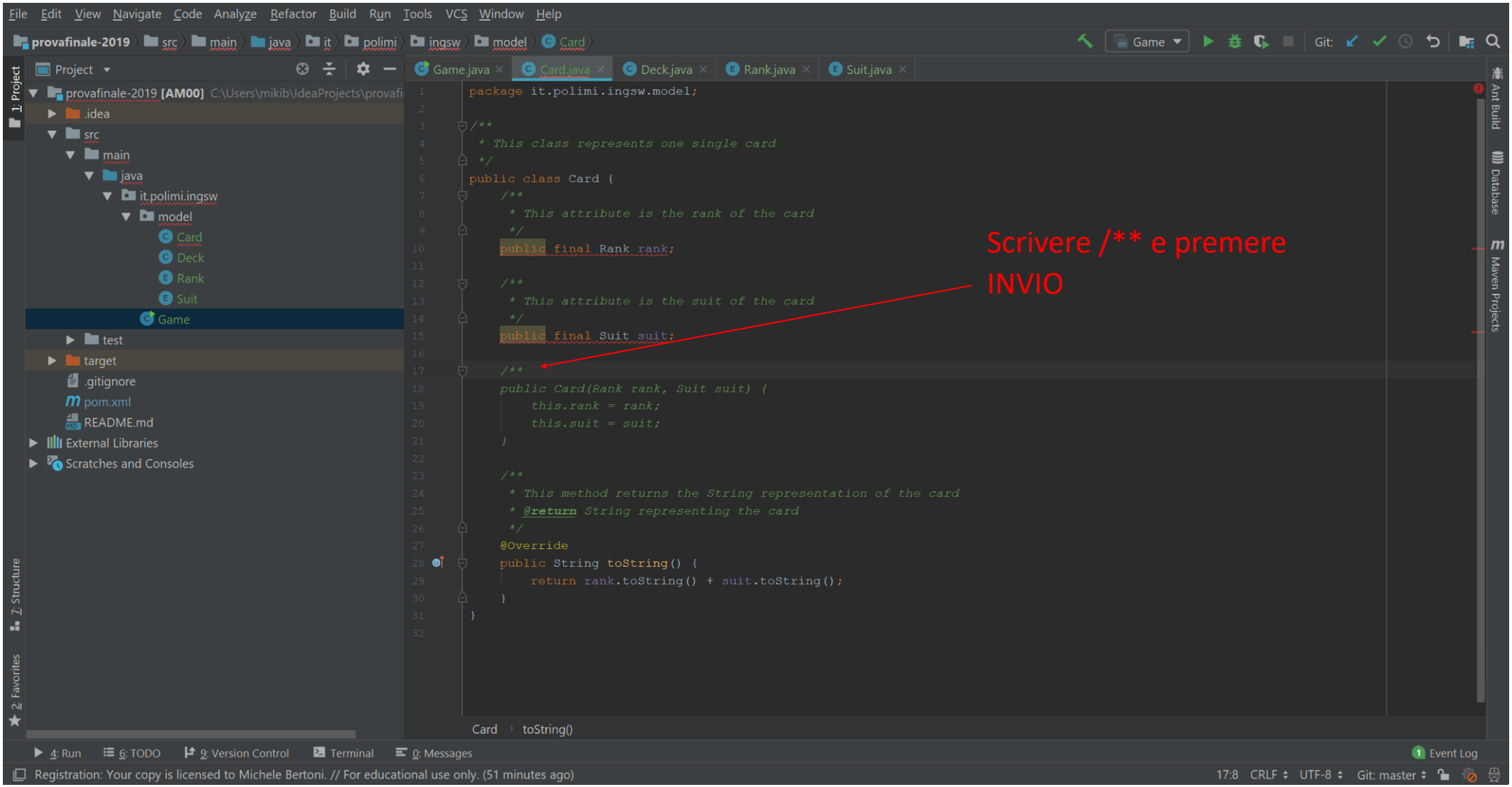
Occorre commentare:

- Classi
- Costruttori (e relativi parametri)
- Attributi
- Metodi (e relativi parametri e/o valori ritornati)

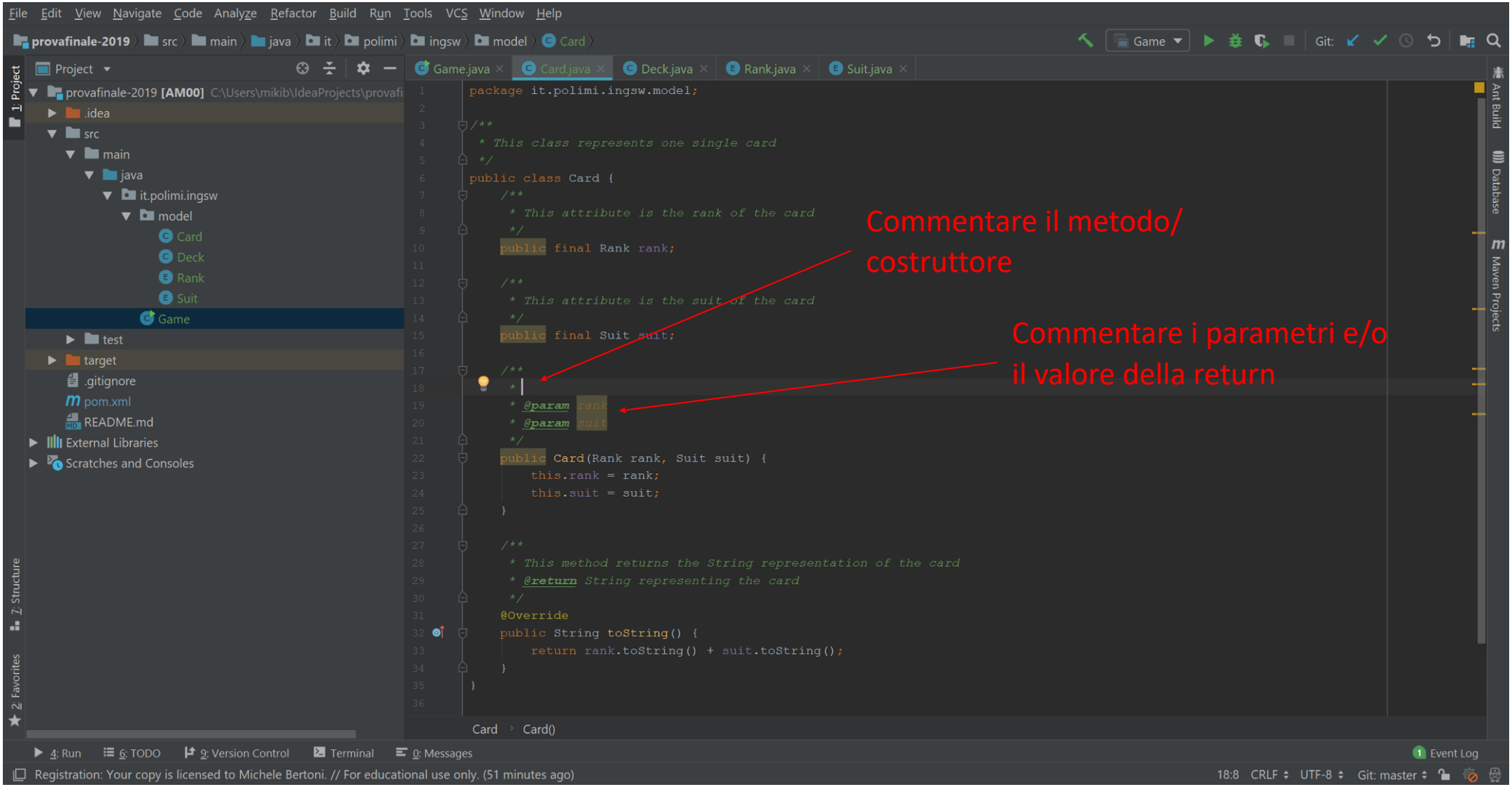
con visibilità public, protected, package-private e private

*I commenti, come il codice,* ***in inglese....***

# Inserire un commento Javadoc



# Inserire un commento Javadoc



# Generare Javadoc (1)

IntelliJ IDEA permette di generare automaticamente Javadoc

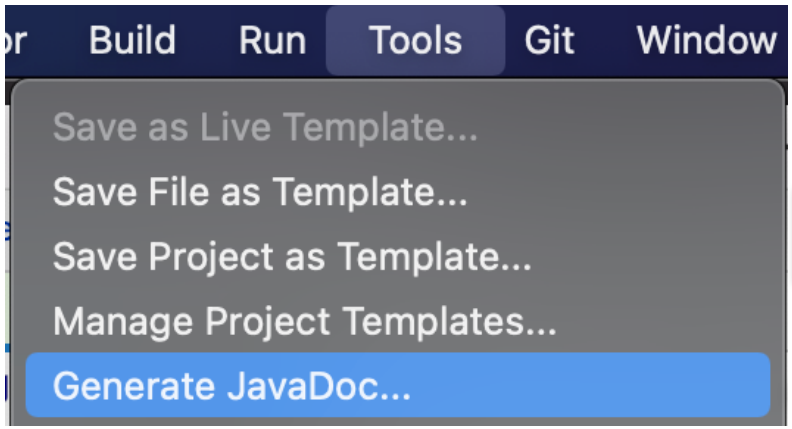
La Javadoc deve essere aggiornata ad ogni nuova versione del software (release), in modo da comprendere le nuove funzionalità e le funzionalità eventualmente modificate

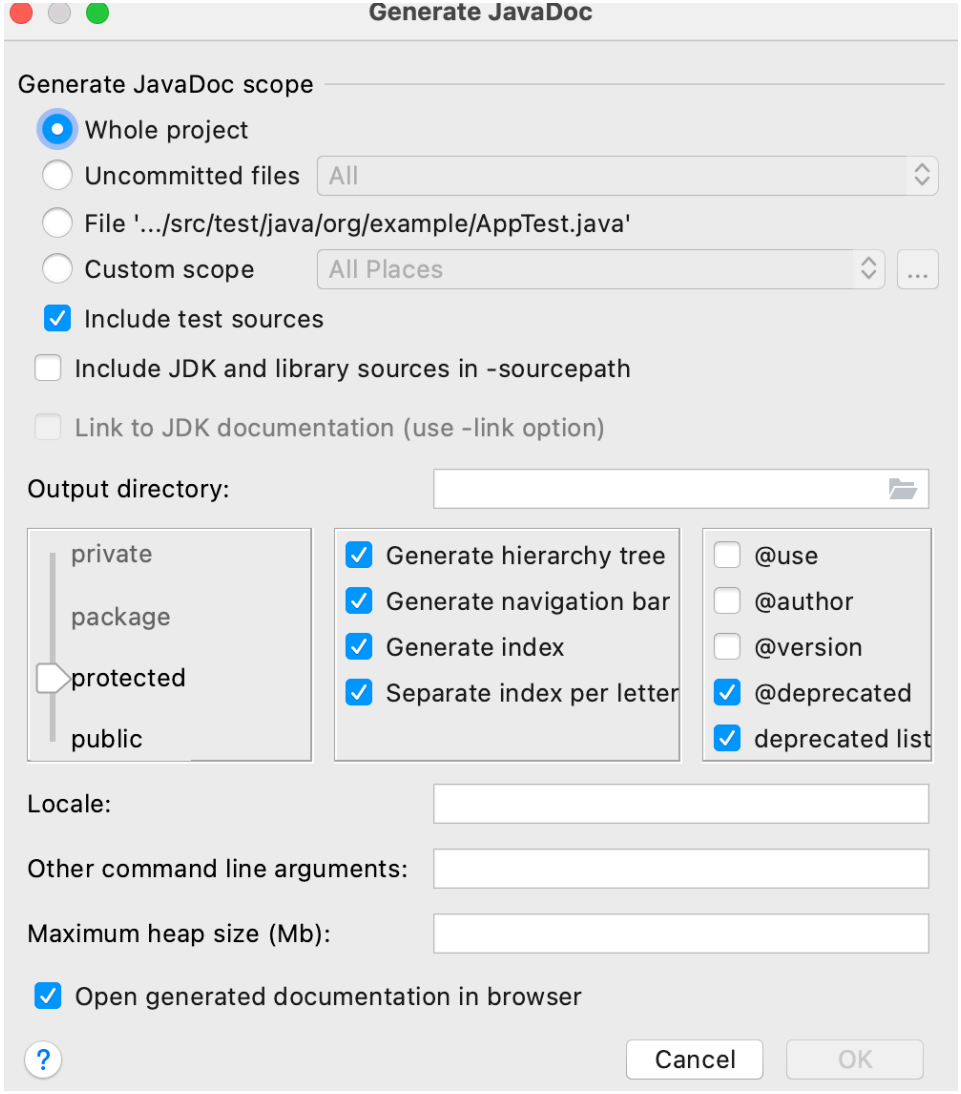
# Generare Javadoc (2)

- Deve essere caricata su GitHub insieme al codice
- In una posizione facilmente accessibile, per esempio
  in una cartella javadoc all'interno della root del progetto (javadoc/)

*Per questo progetto è importante scrivere la Javadoc poco per volta (subito dopo aver finito di scrivere la classe, il metodo, ecc.)*

# Generare Javadoc



# Leggere la Javadoc

La Javadoc generata è composta da un grande numero di file

Per poter leggere la Javadoc basta aprire il file index.html

È possibile trovare questo file alla posizione javadoc/index.html

Aprendolo col browser è possibile leggere la Javadoc come una pagina web:

- Menù laterale che ricalca la suddivisione in package del progetto
- Struttura ad albero per analizzare l'ereditarietà
- Indice analitico

È possibile saltare facilmente da una voce all'altra utilizzando i numerosi collegamenti ipertestuali

# Leggere la Javadoc