Esercitazioni Ing.Sw

BARESI LUCIANO / Gian Enrico Conti

Feb/Mar 2021

JavaDoc

Breve introduzione

Javadoc

Scrivere e generare la Javadoc con IntelliJ IDEA

Slides Credits: Michele Bertoni

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)

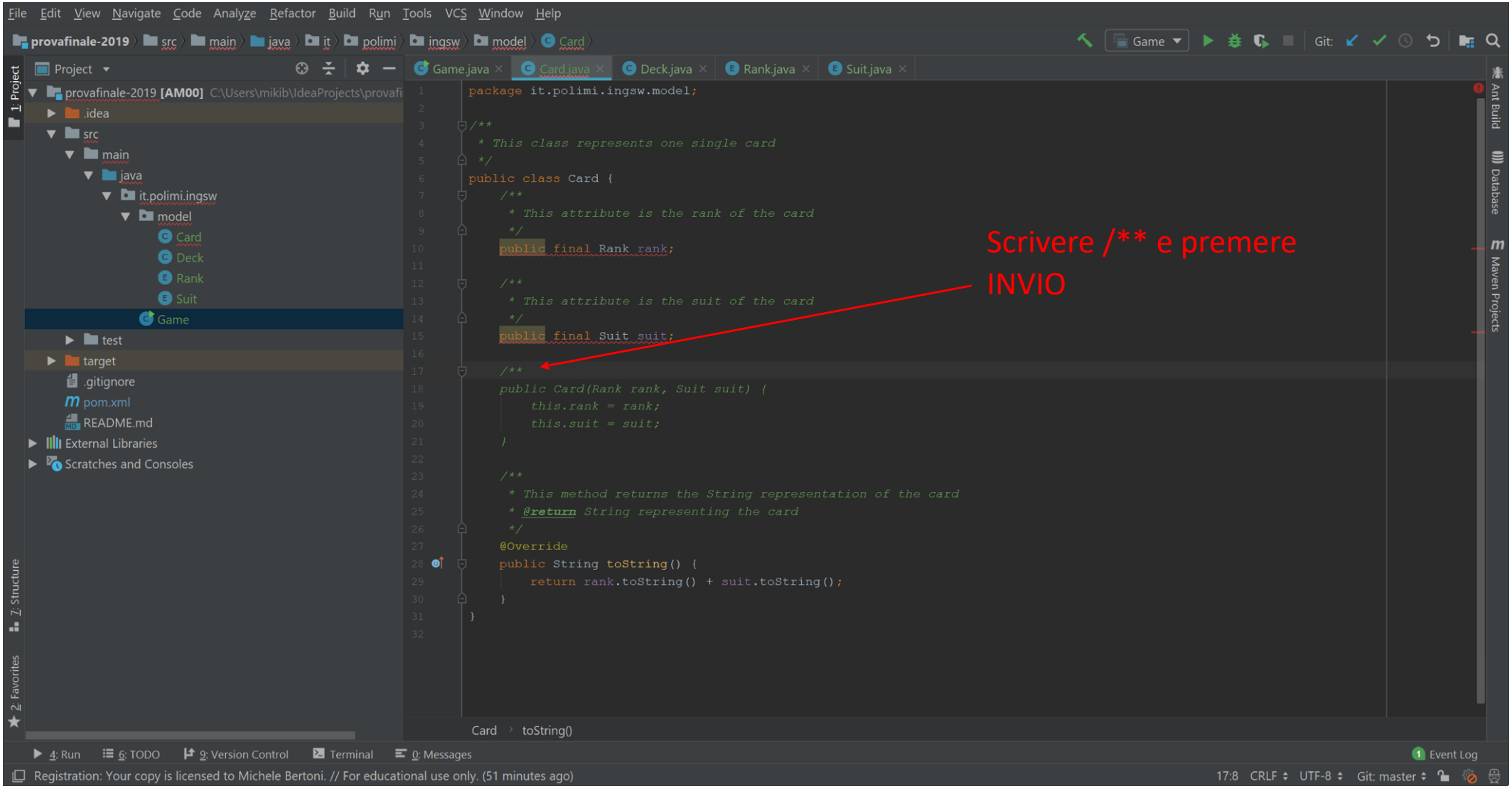
Occorre commentare:

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

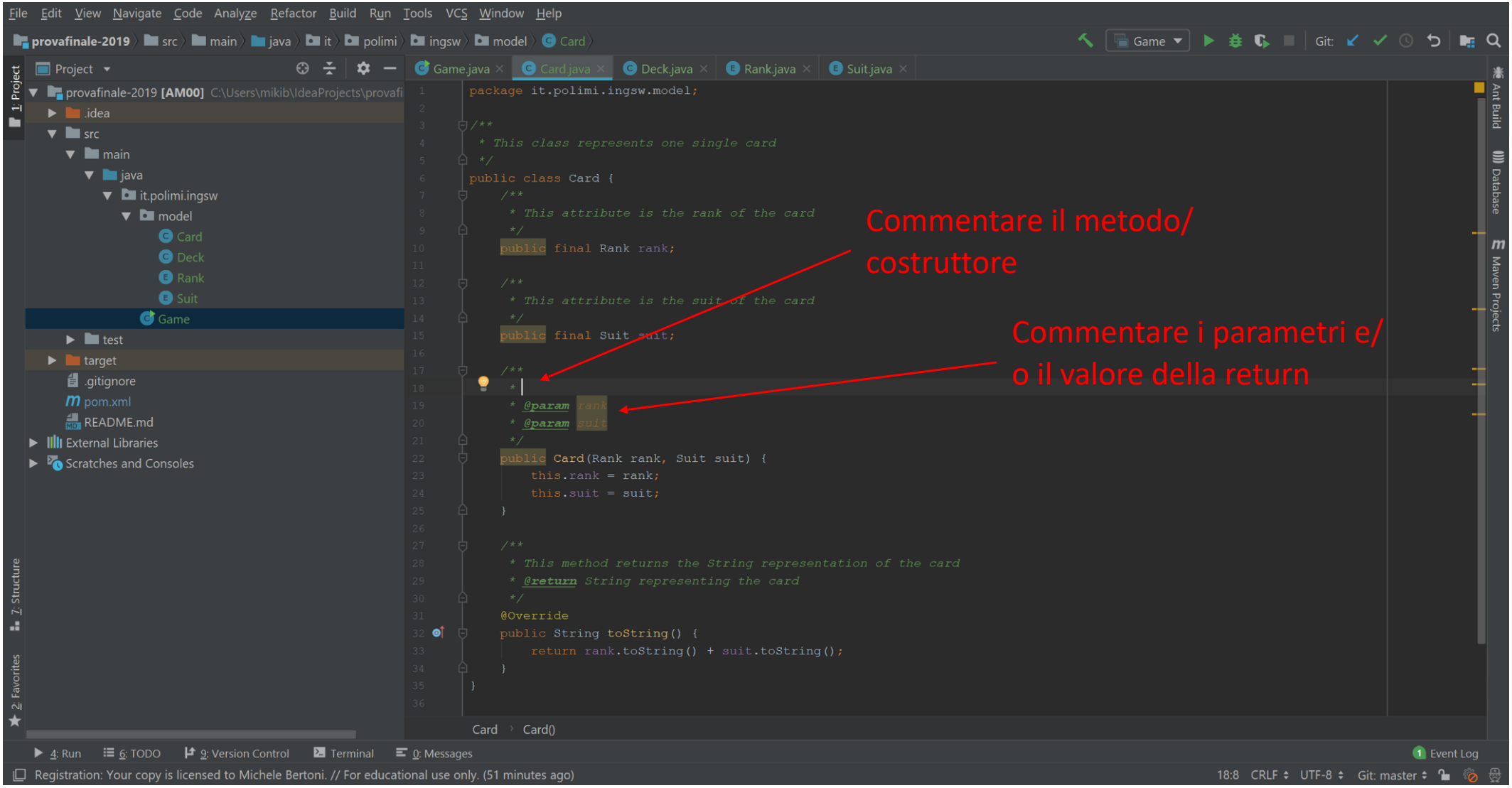
con visibilità public, protected, package-private e private

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

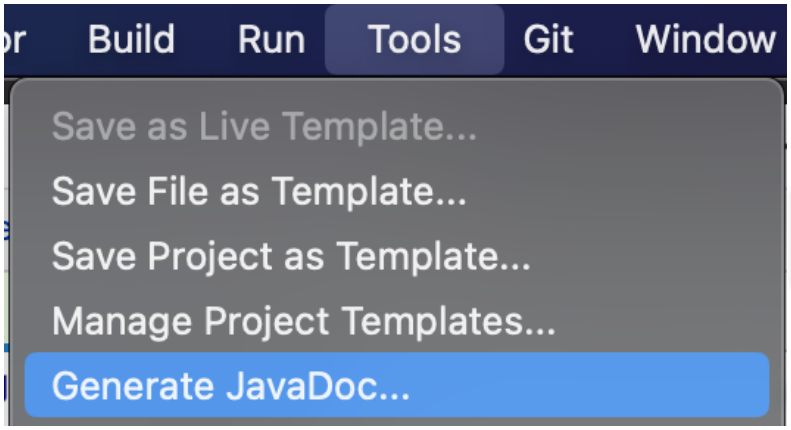
La Javadoc deve essere aggiornata ad ogni nuova versione del software (release), in modo da comprendere le nuove funzionalità e le funzionalità eventualmente modificate

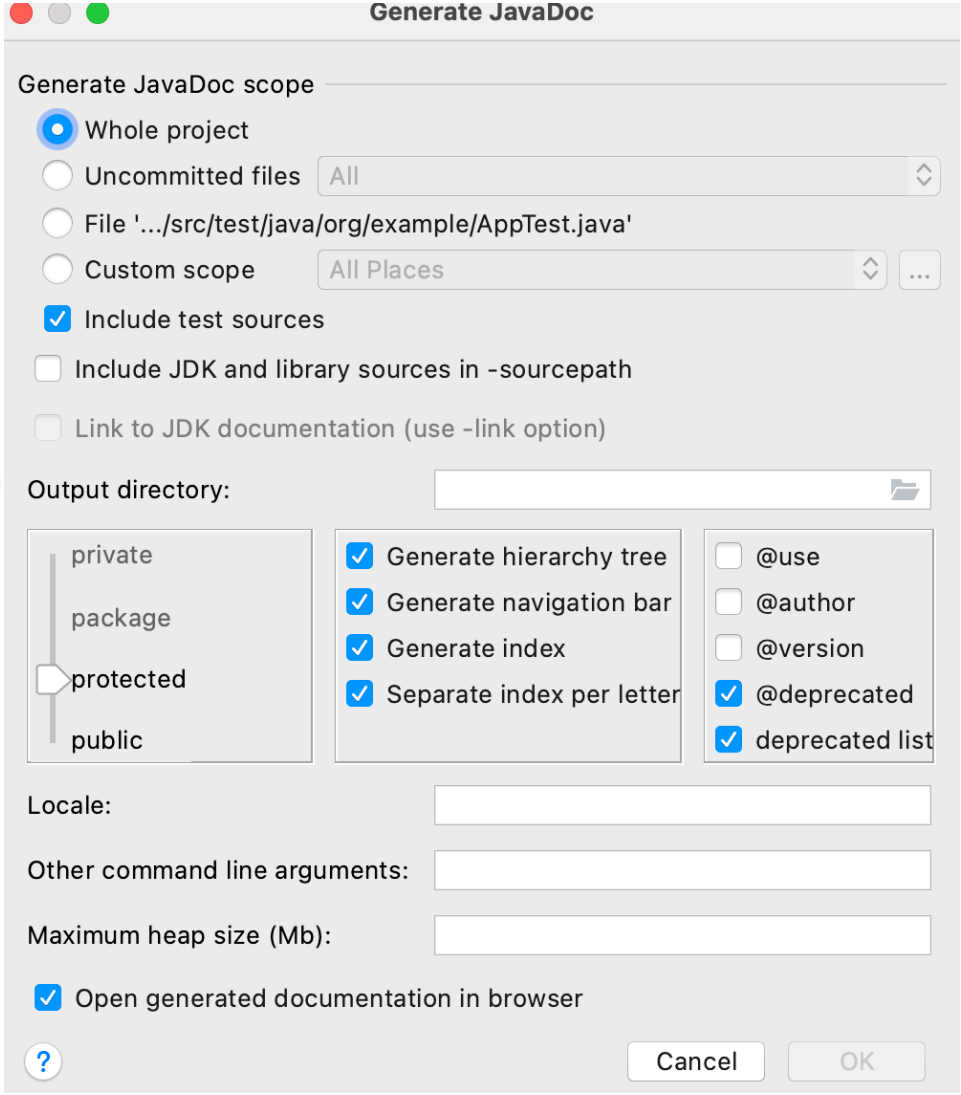
Generare Javadoc (2)

- 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

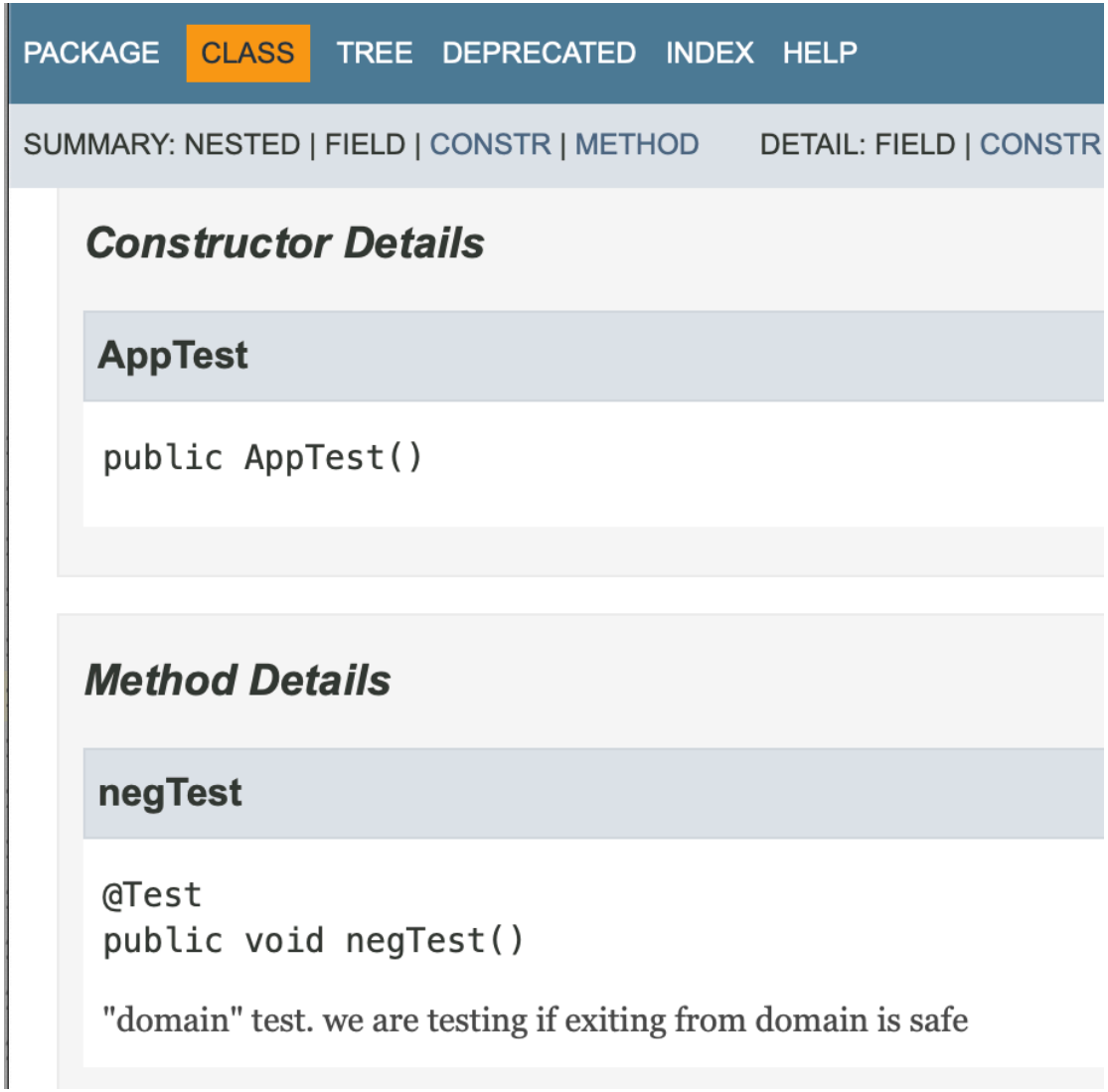
È possibile trovare questo file alla posizione javadoc/index.html

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



Model View Controller

Breve introduzione

MVC

- Design pattern decisamente famoso
- Separa il modello dell'informazione dalla rappresentazione e dalla gestione
  - Stesso modello per gestioni e/o rappresentazioni diverse
- Proposto nel 1979 da Trygve Reenskaug
  - Usato per le interfacce Apple (Lisa e Mac)
  - Molti framework suggeriscono/impongono il pattern
- Antipattern (da evitare)
  - Tutto mischiato in una sola classe

Tre componenti

- Model
  - Responsabile dei dati e delle regole d'accesso ad essi
  - Rappresenta il core dell'applicazione
  - Anche chiamato modello di dominio
- View
  - Visualizza i dati
  - Spesso usa tecnologie specifiche (html, xml)
- Controller
  - Gestisce gli input e regola l'accesso al Model
  - Implementa la logic di business

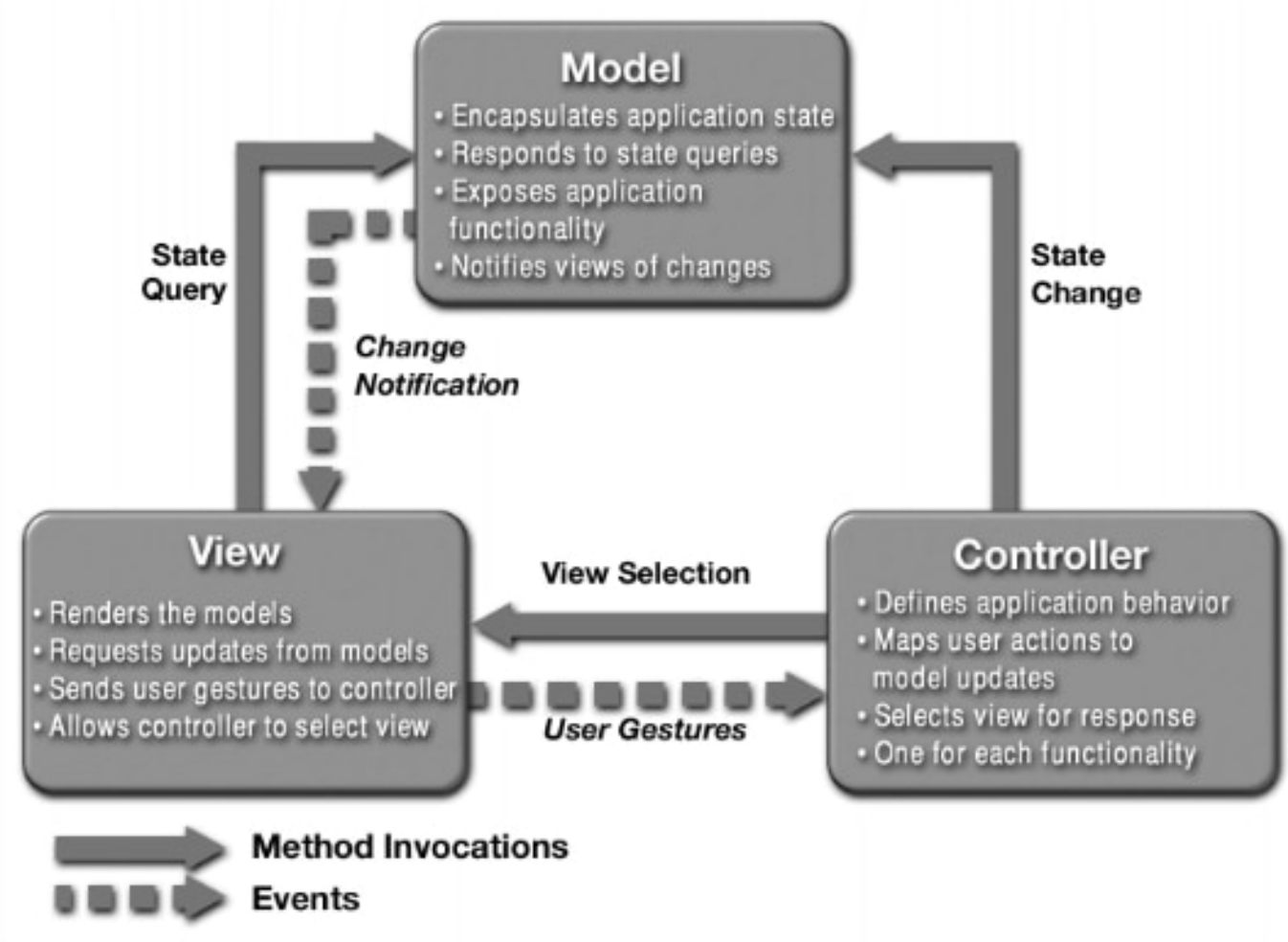
Vantaggi

- Minimizzazione delle dipendenze
- Separazione chiara tra logica di presentazione e logica di business
- Identificazione precisa delle responsabilità di ogni oggetto
- Sviluppo in parallelo facilitato
- Semplificazione della manutenzione
  - Indipendenza tra classi

Svantaggi

- Incremento della complessità
- Inefficienza dell'accesso ai dati attraverso l'interfaccia (??)
- Può diventare fondamentale l'uso e la conoscenza di diverse tecnologie

Versione base



https://www.oracle.com/technical-resources/articles/javase/mvc.html

MVC History:

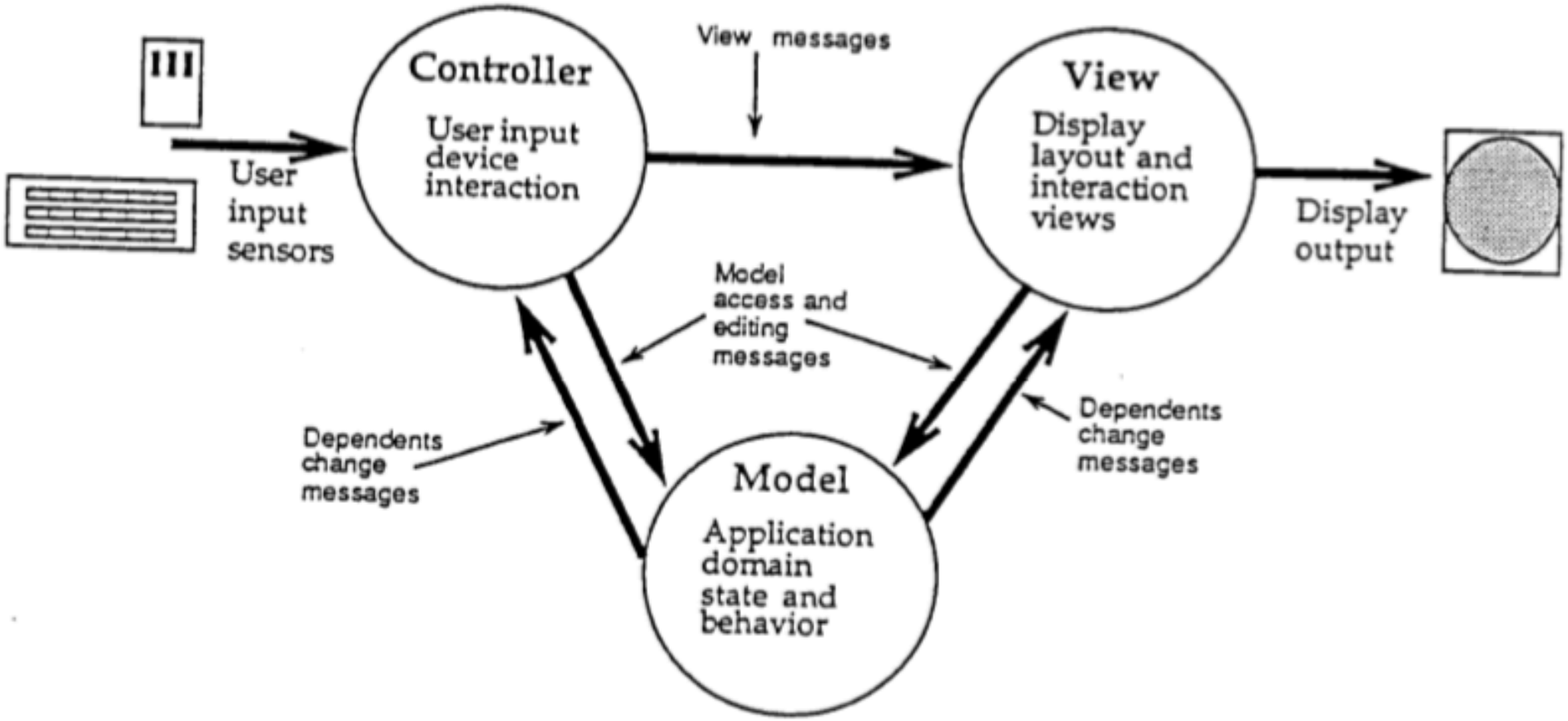
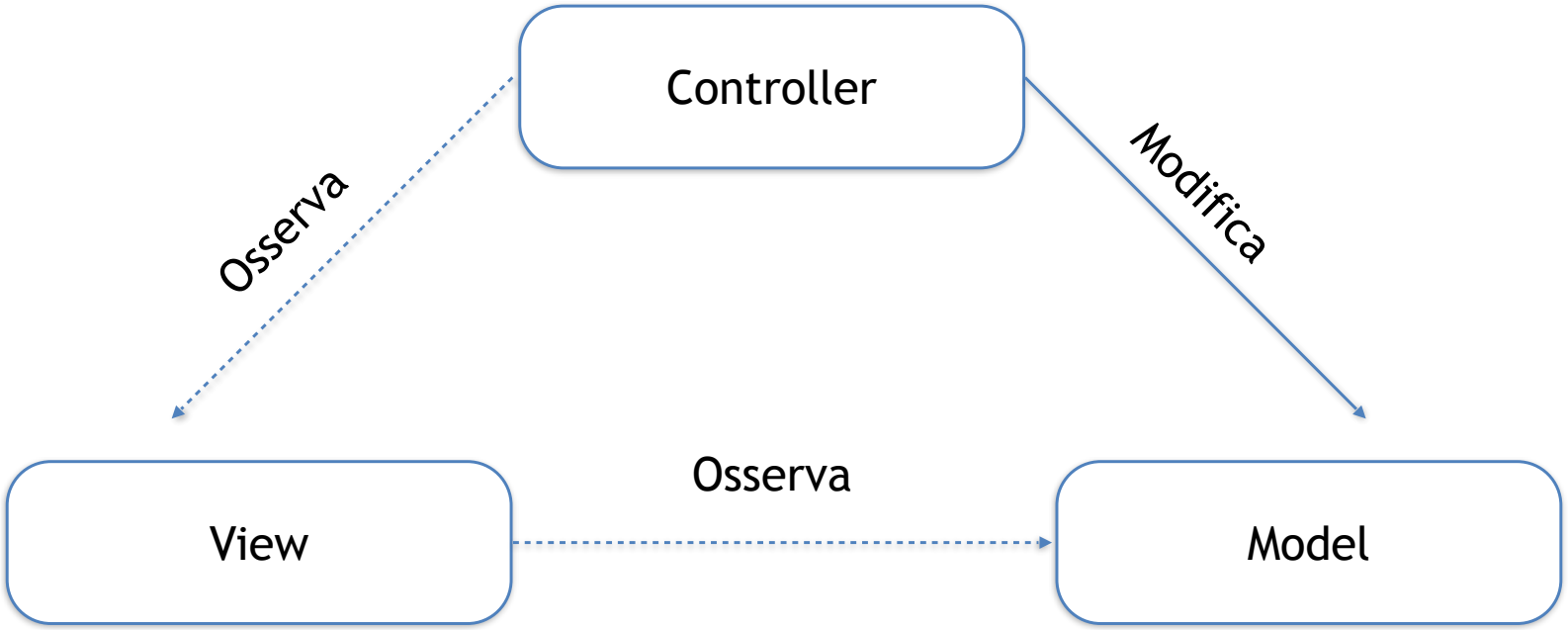


Figure 1: Model-View-Controller State and Message Sending

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.

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.

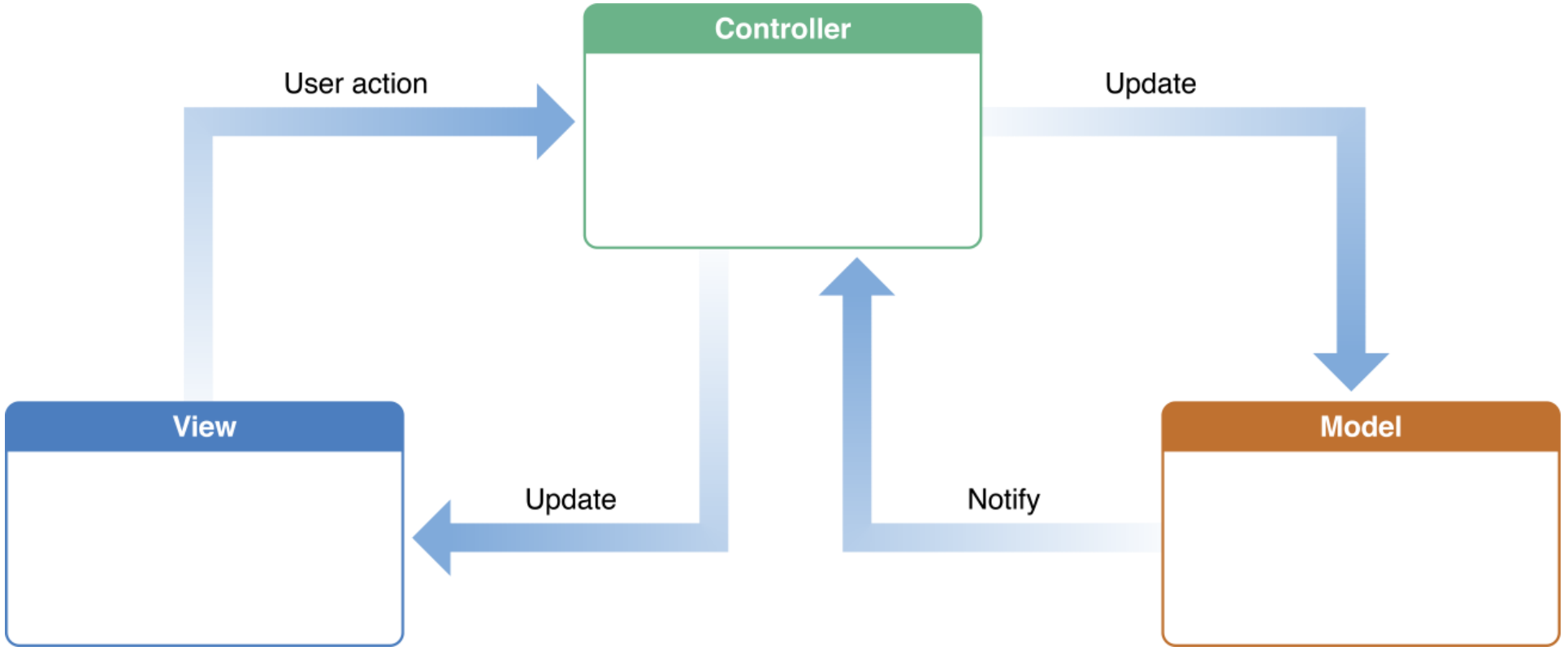
MVC Oracle/JEEE definitions

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

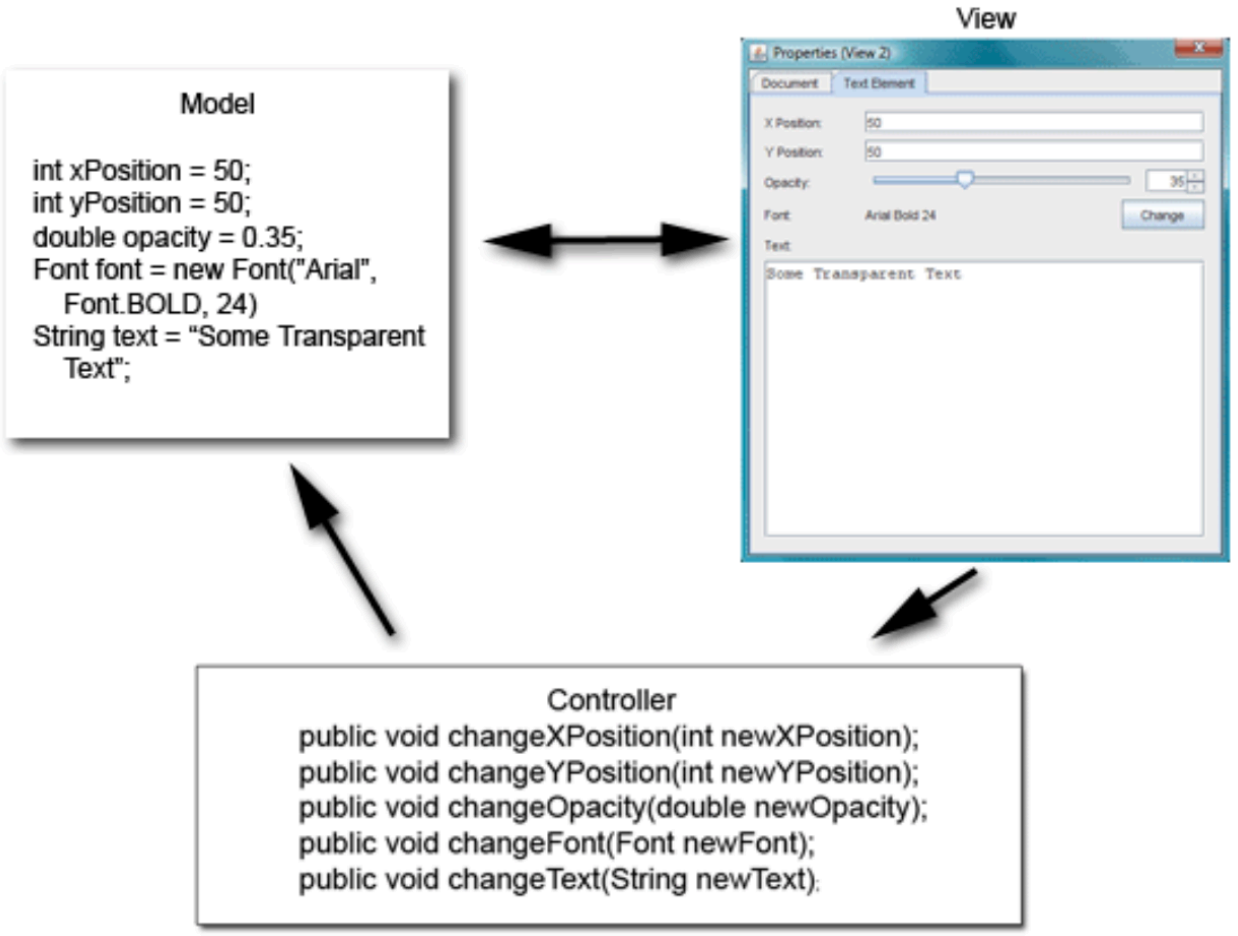
- In a stand-alone GUI client, *user interactions could be button clicks or menu selections*
- in a Web application, they appear as GET and POST HTTP requests. The actions performed by the model include activating business processes or changing the state of the model.

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

Model-View-Controller: Varianti



Esempio



https://www.oracle.com/technical-resources/articles/javase/mvc.html

Interazioni

- View riconosce l'azione fatta dall'utente
- View notifica il Controller
- Controller gestisce Model in base all'azione compiuta
- Se Model è stato cambiato, questo notifica le parti interessate (ad esempio, View) attraverso listener

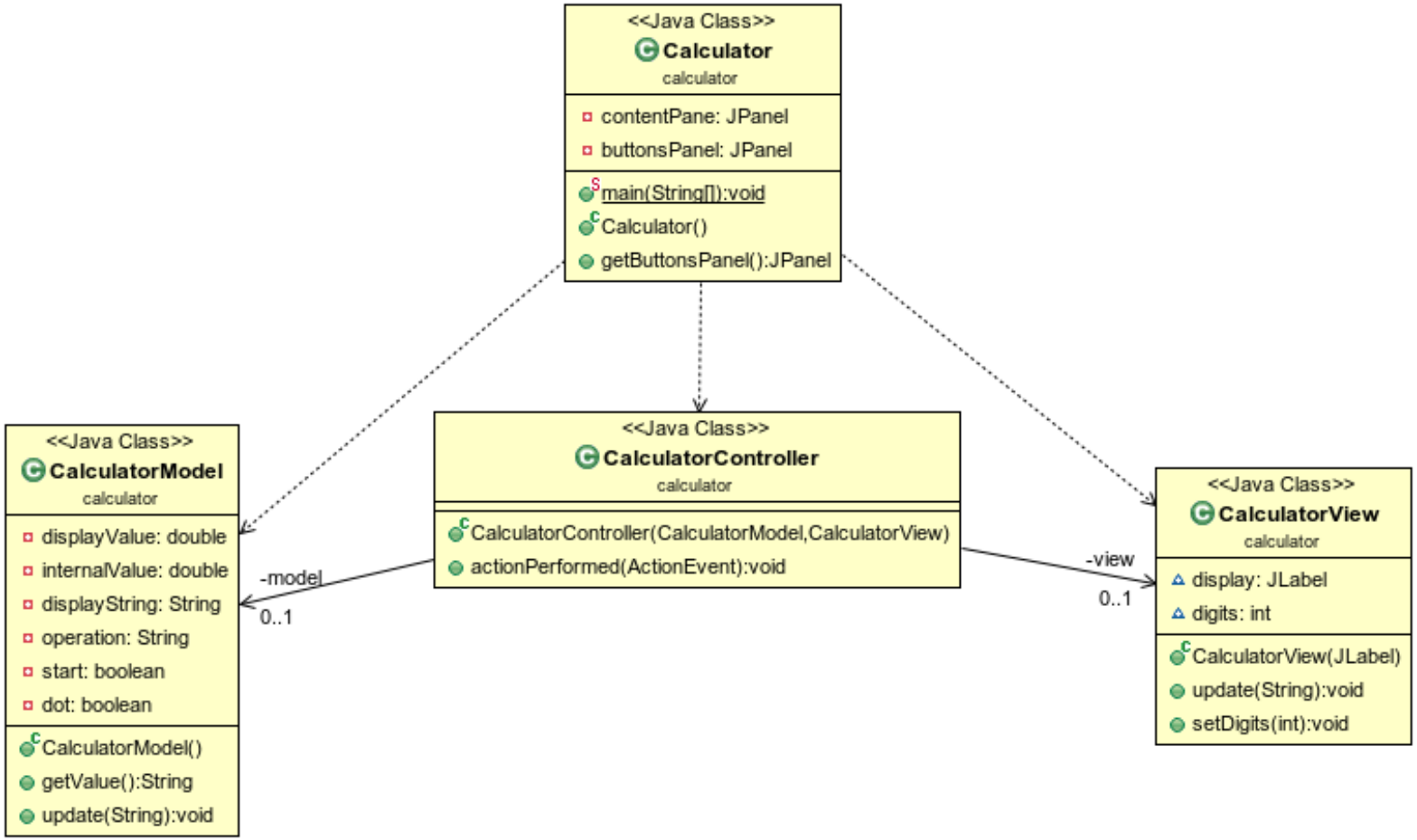
- Model non ha riferimenti a View, ma usa un approccio ad eventi per le modifiche
  - View diverse possono condividere lo stesso Model

Versione rivista

- Richiede che Controller agisca sempre da mediatore tra Model e View
  - Tipico del framework Apple Cocoa

- Le notifiche di cambiamento di stato in Model sono comunicate a View attraverso Controller
- View usa Controller per tradurre le azioni utente in azioni su Model
- Questa versione favorisce il disaccoppiamento tra View e Model

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;
  }
  …
```

Dove sono le vari parti?

Modello del Gioco: server

View: sul singoli player/client

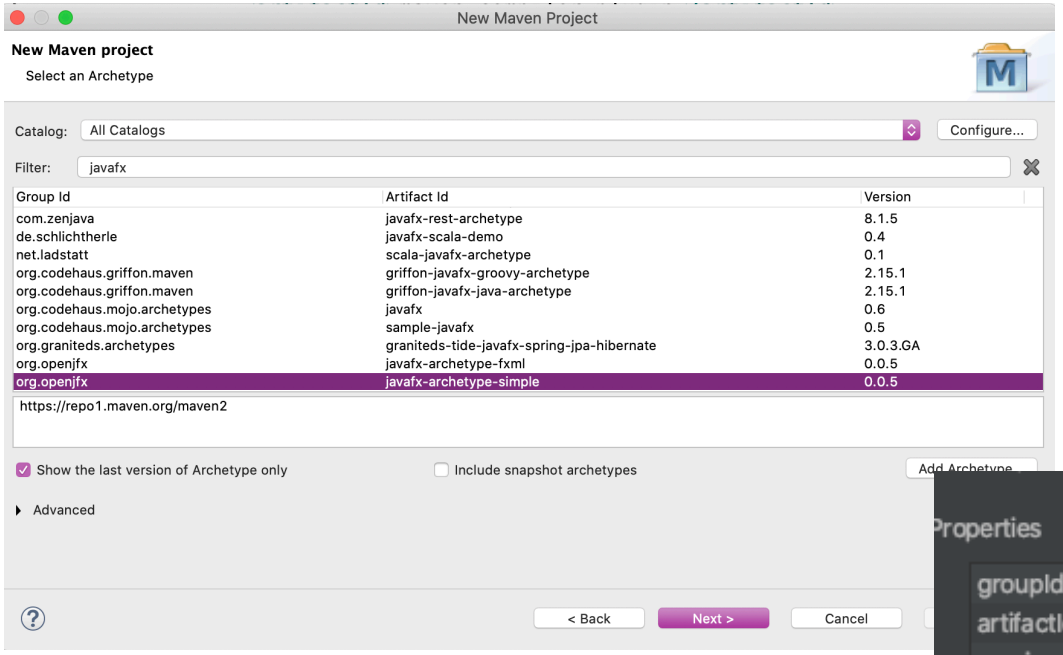
(Il server NON deve sapere se GUI / CLI)

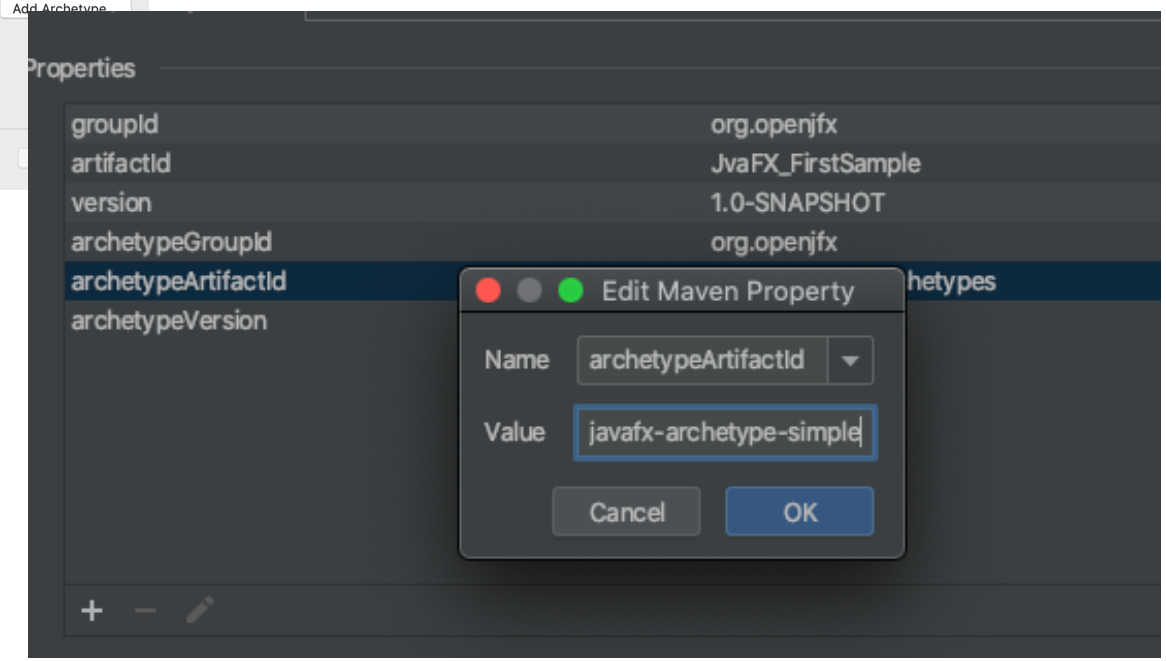
Controller.... maybe...

Mettiamo tutto insieme

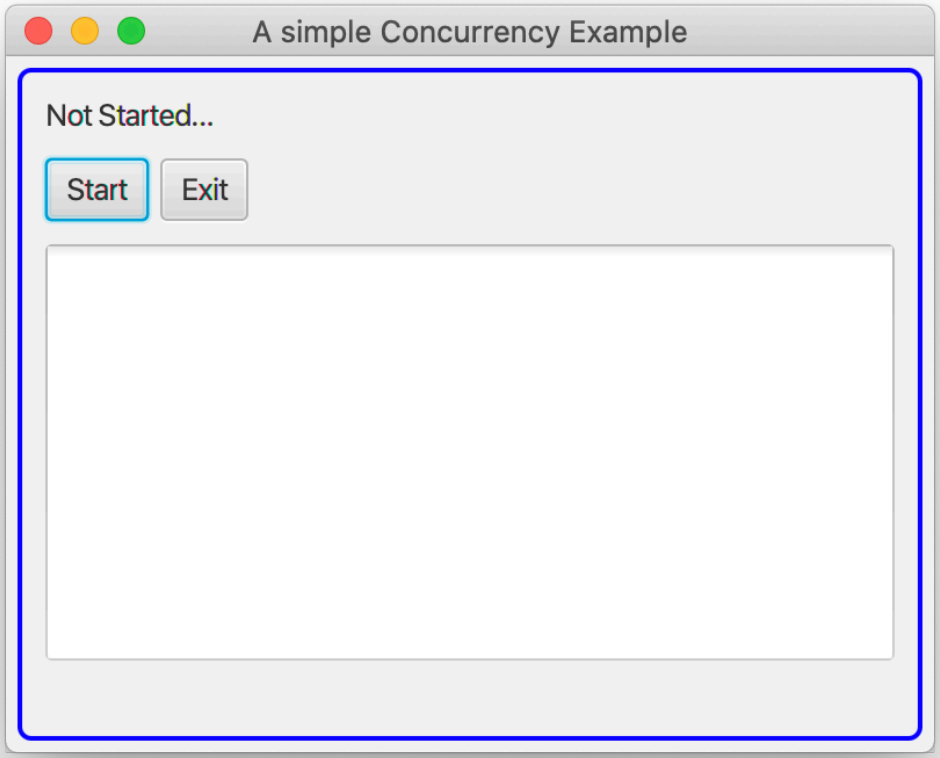
JavaFX, Socket, Thread e Maven

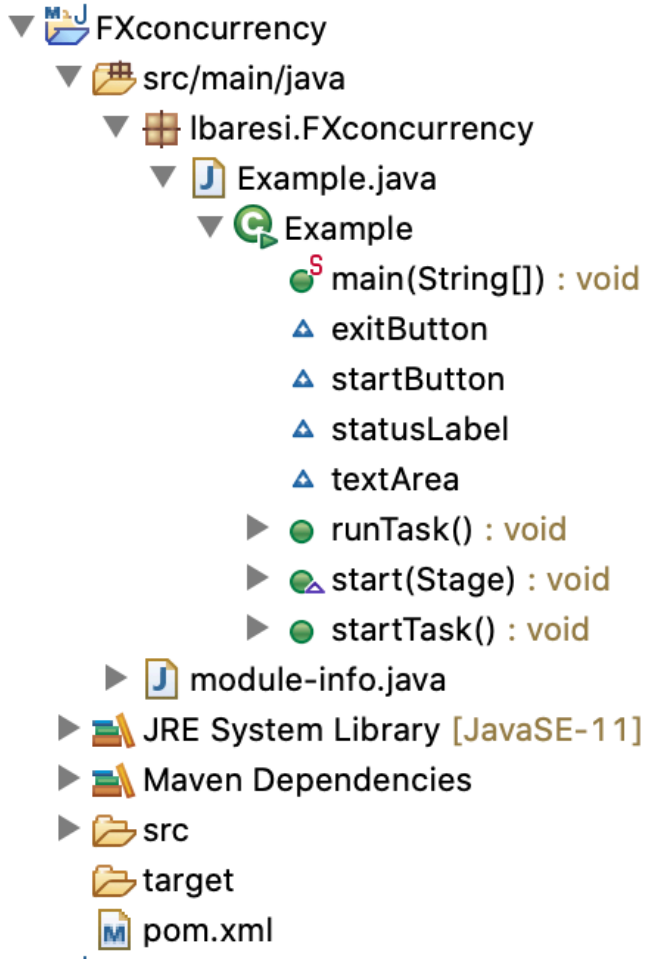
Creazione progetto Maven





Concorrenza





https://examples.javacodegeeks.com/desktop-java/javafx/javafx-concurrency-example/

Concorrenza

+ significati:

- user / rete si contendono la UI
- Thread ed accesso al modello
    - Login:
    - Utente A e Utente B via Rete -> accesso...
        - Chi vince ?
        - Codice "Thread-Safe?
            - Lock/semafori/Synchronized...

Concorrenza

Concetti base:

- Java FX Single-threaded Rendering
- "Java Thread" approccio std
- NON potete chiamare update di JAVAFx da altri thread
- "only the JavaFX application thread is allowed to make any changes to the **JavaFX Scene Graph**"
- **How?**

Concorrenza JavaFX

Platform.runLater()

- Passare al ***runLater()*** una istanza di Runnable
- Tale istanza sara' eseguita sul thread di JavaFX ("when it has time" ...)
- Dalla istanza di Runnable potete modificare il "JavaFX scene graph"

Concorrenza JavaFX

Platform.runLater() reduced sample:

```
    Thread taskThread = new Thread(...{
     ...
     {
        // it's time...

        ...
          Platform.runLater(new Runnable() {
             @Override
             public void run() {
               progressBar
                   .setProgress(reportedProgress);
        ....


        }
```

(More detail later...)

Concorrenza JavaFX

Platform.runLater() reduced sample demo

```
....
   Platform.runLater(new Runnable() {
       @Override
       public void run() {
        progressBar.setProgress(reportedProgress);
     }
   });

.....
```

Main application

```
public class Example extends Application {
   …
   Button startButton = new Button("Start");
   Button exitButton = new Button("Exit");

   @Override
   public void start(final Stage stage) {
   // Create scene

      startButton.setOnAction(new EventHandler <ActionEvent>() {
         public void handle(ActionEvent event) {startTask();}});

      exitButton.setOnAction(new EventHandler <ActionEvent>() {
         public void handle(ActionEvent event) {stage.close();}});


      …
      stage.setScene(scene);
      stage.show();
   }
}
```

startTask

```
public void startTask() {
   Runnable task = new Runnable() {
      public void run() {
         runTask();
      }
   };

   // Run the task in a background thread
   Thread backgroundThread = new Thread(task);
   // Terminate the running thread if the application exits
   backgroundThread.setDaemon(true);
   // Start the thread
   backgroundThread.start();
}
```

runTask

```
public void runTask() {
  for(int i = 1; i <= 10; i++) {
    try {
      final String status = "Processing " + i + " of " + 10;

      // Update the Label on the JavaFx Application Thread
      Platform.runLater(new Runnable() {
        @Override
        public void run() {
                statusLabel.setText(status);
                }
      });

      textArea.appendText(status+"\n");
      Thread.sleep(1000);
    }
    catch (InterruptedException e) {
      e.printStackTrace();
    }
  }
}
```

Maven

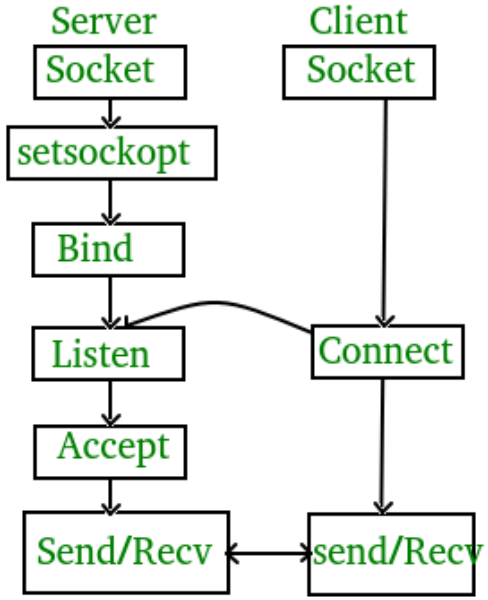
```
<plugin>
    <groupId>org.openjfx</groupId>
    <artifactId>javafx-maven-plugin</artifactId>
    <version>0.0.4</version>
    <configuration>
        <mainClass>lbaresi.FXconcurrency.Example</mainClass>
    </configuration>
</plugin>
```

Gestiamo anche i socket

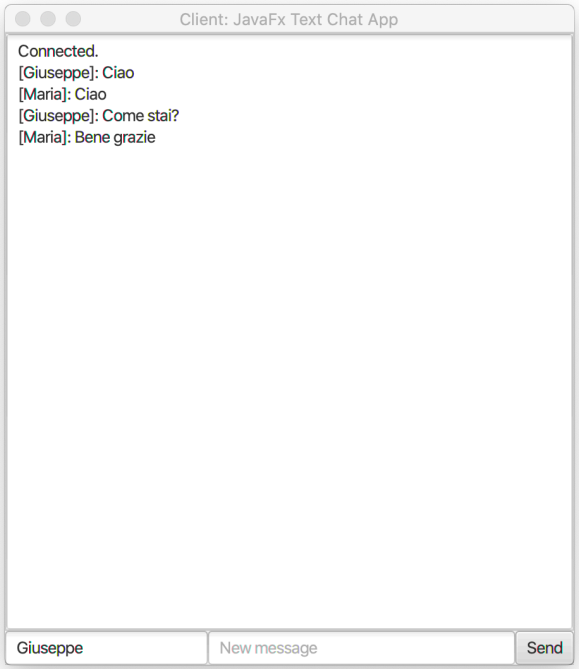
Gestiamo anche i socket

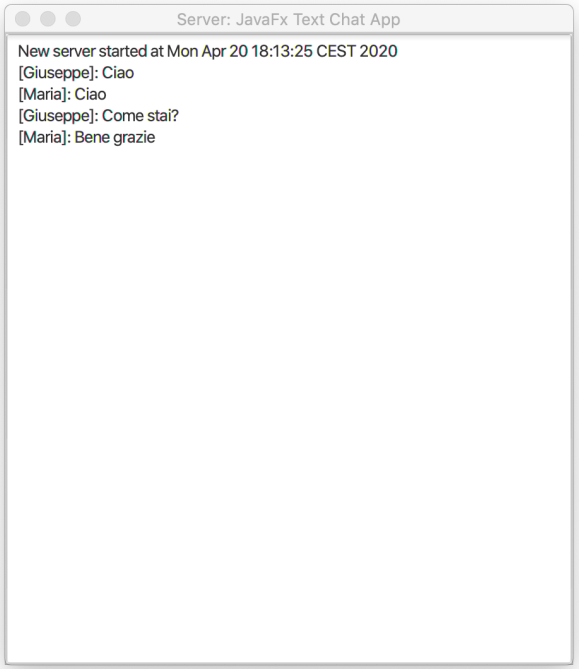
Recap:

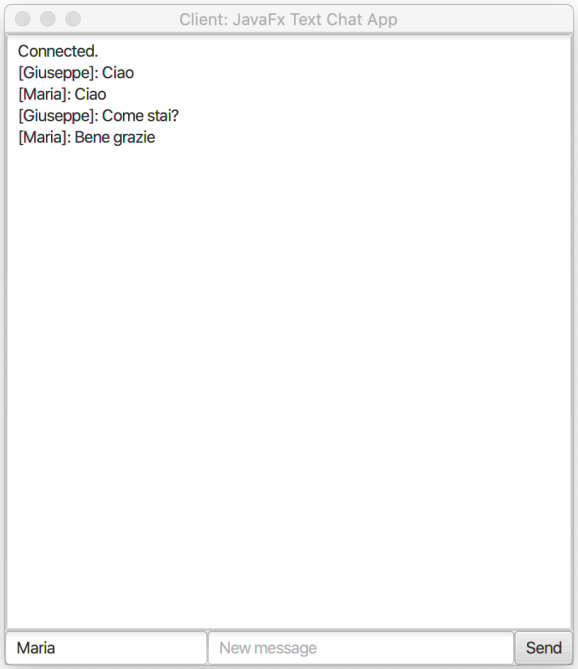
- server
    Apre socket
    Ascolta (listen)
        Se riceve ("accept")-> thread dedicato -> read
- client



Esempio



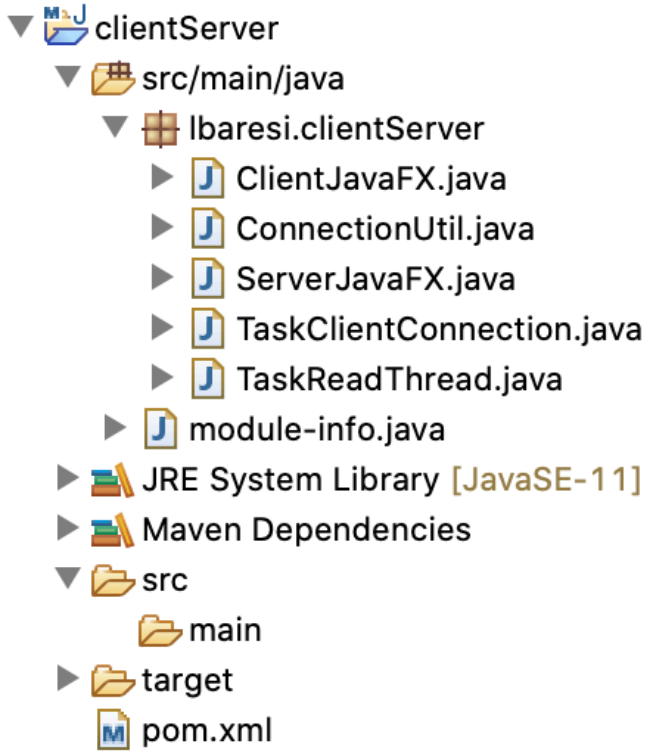




http://topman.com.np/blogs/10/how-i-create-chat-application-in-javafx-using-socket-programming

Un solo progetto Maven

- Si può avere un solo progetto
- Maven può gestire la creazione/esecuzione di programmi diversi



ConnectionUtil

- Nulla di fondamentale
- Aiuta ad essere consistenti
  - Pulizia concettuale

```
public class ConnectionUtil {
   public static String host="localhost";
   public static int port=8001;

public static  setPortFromCmdLine(int p){
        Port = p;
}

public static  setPortFromPrefsinXML(){
        Port = ....
}
```

Client

```
public class ClientJavaFX extends Application {
  …
  DataOutputStream output = null;

  @Override
  public void start(Stage primaryStage) {
    …
    Button btnSend = new Button("Send");
    btnSend.setOnAction(new ButtonListener());
    …
  try {
     // Create a socket to connect to the server
     Socket socket = new Socket(ConnectionUtil.host, ConnectionUtil.port);
     //Connection successful
     txtAreaDisplay.appendText("Connected. \n");
     // Create an output stream to send data to the server
     output = new DataOutputStream(socket.getOutputStream());

     //create a thread in order to read message from server continuously
     TaskReadThread task = new TaskReadThread(socket, this);
     Thread thread = new Thread(task);
       thread.start();
  } catch (IOException ex) {
    txtAreaDisplay.appendText(ex.toString() + '\n');
  }
}
```

TaskReadThread (I)

```
public class TaskReadThread implements Runnable {
    //private variables
    Socket socket;
    ClientJavaFX client;
    DataInputStream input;

    //constructor
    public TaskReadThread(Socket socket, ClientJavaFX client) {
        this.socket = socket;
        this.client = client;
    }

    @Override
    public void run() {…}
}
```

TaskReadThread (II)

```
public void run() {
    while (true) {
        try {
            //Create data input stream
            input = new DataInputStream(socket.getInputStream());
            //get input from the client
            String message = input.readUTF();

            //append message of the Text Area of UI (GUI Thread)
            Platform.runLater(() -> {client.txtAreaDisplay.appendText(message + "\n");});
        } catch (IOException ex) {
            System.out.println("Error reading from server: " + ex.getMessage());
            ex.printStackTrace();
            break;
        }
    }
}
```

ButtonListener

```
private class ButtonListener implements EventHandler<ActionEvent> {
    @Override
    public void handle(ActionEvent e) {
        try {
            String username = txtName.getText().trim();
            String message = txtInput.getText().trim();

            if (username.length() == 0) username = "Unknown";
            //if message is empty, just return : don't send the message
            if (message.length() == 0) return;

             //send message to server
             output.writeUTF("[" + username + "]: " + message + "");
             output.flush();

              //clear the textfield
              txtInput.clear();
        } catch (IOException ex) {System.err.println(ex);}
    }
}
```

Server

```
public class ServerJavaFX extends Application {
    public TextArea txtAreaDisplay;
    List<TaskClientConnection> connectionList = new ArrayList<TaskClientConnection>();

    public void start(Stage primaryStage) {…}

    //send message to all connected clients
    public void broadcast(String message) {
        for (TaskClientConnection clientConnection : this.connectionList) {
            clientConnection.sendMessage(message);
        }
    }
}
```

Server (II)

```
public void start(Stage primaryStage) {
  …
  new Thread(() -> {
    try {
      // Create a server socket
      ServerSocket serverSocket = new ServerSocket(ConnectionUtil.port);

      //append message of the Text Area of UI (GUI Thread)
      Platform.runLater(() -> txtAreaDisplay.appendText("Server started at " + new Date() + '\n'));

      while (true) {
        // Listen for a connection request, add new connection to the list
        Socket socket = serverSocket.accept();
        TaskClientConnection connection = new TaskClientConnection(socket, this);
        connectionList.add(connection);
        //create a new thread
        Thread thread = new Thread(connection);
        thread.start();
      }
    } catch (IOException ex) {txtAreaDisplay.appendText(ex.toString() + '\n');}
  }).start();
}
```

TaskClientConnection

```
public  class TaskClientConnection implements Runnable {
    Socket socket;
    ServerJavaFX server;
    // Create data input and output streams
    DataInputStream input;
    DataOutputStream output;

    public TaskClientConnection(Socket socket, ServerJavaFX server) {
        this.socket = socket;
        this.server = server;
    }

    @Override
    public void run() {…}
    //send message back to client
    public void sendMessage(String message) {
          try {
            output.writeUTF(message);
            output.flush();
        } catch (IOException ex) {ex.printStackTrace();}
    }
}
```

TaskClientConnection (II)

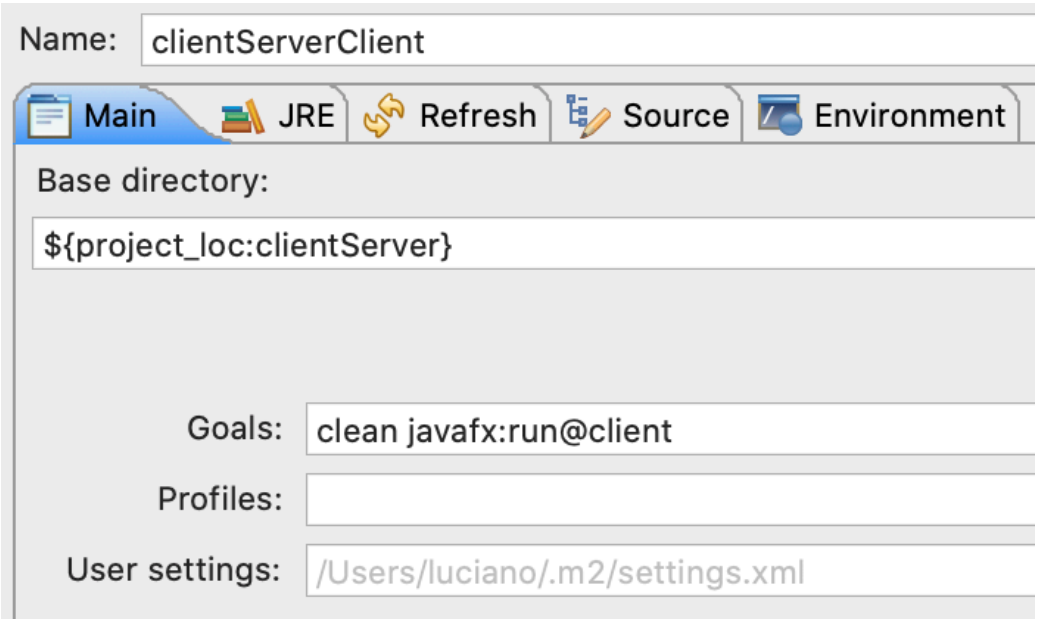
```
public void run() {
  try {
    // Create data input and output streams
    input = new DataInputStream(socket.getInputStream());
    output = new DataOutputStream(socket.getOutputStream());

    while (true) {
      // Get message from the client
      String message = input.readUTF();
      //send message via server broadcast
      server.broadcast(message);

       //append message of the Text Area of UI (GUI Thread)
       Platform.runLater(() -> {server.txtAreaDisplay.appendText(message + "\n");});
     }
  }
  catch (IOException ex) {ex.printStackTrace();}
  finally {
    try {
      socket.close();
    } catch (IOException ex) {ex.printStackTrace();}
  }
}
```

Compilazione 1

```
<plugin>
    <groupId>org.openjfx</groupId>
    <artifactId>javafx-maven-plugin</artifactId>
    <version>0.0.4</version>
    <executions>
        <execution>
            <id>server</id>
            <configuration>
                <mainClass>lbaresi.clientServer.ServerJavaFX</mainClass>
            </configuration>
        </execution>
        <execution>
            <id>client</id>
            <configuration>
                <mainClass>lbaresi.clientServer.ClientJavaFX</mainClass>
            </configuration>
        </execution>
    </executions>
</plugin>
```



Compilazione 2

```
<plugin>
    <groupId>org.codehaus.mojo</groupId>
    <artifactId>exec-maven-plugin</artifactId>
    <version>1.2.1</version>
    <executions>
        <execution>
            <id>server</id>
            <goals>
                <goal>java</goal>
            </goals>
            <configuration>
                <mainClass>lbaresi.clientServer.ServerJavaFX</mainClass>
                <includePluginDependencies>false</includePluginDependencies>
            </configuration>
        </execution>
        <execution>
            …
        </execution>
    </executions>
</plugin>
```