Esercitazioni Ing.Sw

Alessandro Margara / Gian Enrico Conti

II sem 2023

Webex:

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

Esercitazione 2

Towards VirtualView

ing. Gian Enrico Conti

Prova Finale - Ingegneria del Software - AA 2022/23

1) Partiamo da JavafxSample

https://github.com/ingconti/SampleJavaFx

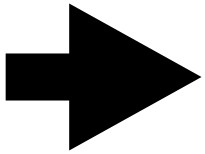
2) togliamo Codice disegno e mettiamo 1 label:

```
public class App extends Application {

    public static void main(String[] args) {
        launch(args);
    }

    @Override
    public void start(Stage primaryStage) {

        Label label = new Label("My Label");
        Scene scene = new Scene(label, 500, 600);
        primaryStage.setScene(scene);
        primaryStage.show();
    }
}
```

mettiamo M V C

- Semplificato: x ora la App **è** il controller..
- Due classi: Model e View
- La view e' la ns "Scene", adattiamo:

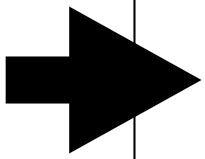
```
public class App extends Application {

    Label label = new Label("My Label");
    public static void main(String[] args) {
        launch(args);
    }

    @Override
    public void start(Stage primaryStage) {
        primaryStage.setTitle("MVC VV");
        Scene scene = new Scene(label, 200, 100);
        primaryStage.setScene(scene);
        primaryStage.show();
    }

}
```



```
public class App extends Application {

    public static void main(String[] args) {
        launch(args);
    }

    private View view = new View();

    @Override
    public void start(Stage primaryStage) {
//        Label label = new Label("My Label");
//        Scene scene = new Scene(label, 200, 100);

        primaryStage.setScene(this.view.getScene());
        primaryStage.show();
    }
}


----
public class View {
    private Label label = new Label("My Label");
     label.setFont(new Font("Arial", 30));

    private Scene scene = new Scene(label, 200, 100);

    Scene getScene(){
        return scene;
    }
}
```

Model: gioco "stupido":
Se tappi su un punto e centri
il cerchio nascosto, vinci.

Mostriamo anche io numero di tap
(Per mostra Pattern Observe.... able)



Some code for model:

```
public class GameModel {

    // for now fixed:
    private CartesianCircle circle = new CartesianCircle(20, 20, 50);
}
```

Some code for model...

```
public class CartesianPoint {

    double x,y;

    public CartesianPoint(double x, double
y){
        this.x = x;
        this.y = y;
    }

    public double
distanceFrom(CartesianPoint otherP){
        double c1 = otherP.x-this.x;
        double c2 = otherP.y-this.y;
        double d = sqrt(c1* c1 + c2 * c2);
        return d;
    }
}
```

Some code for model...

```
public class CartesianCircle {

    private CartesianPoint center;
    private double radius;

    CartesianCircle(double x,
double y, double radius){
        this.center = new
CartesianPoint(x,y);
        this.radius = radius;
    }

    boolean contains(CartesianPoint
p){
        return
center.distanceFrom(p)<this.radius;
    }
}
```

Detect click:

```
public class View {

    private Label label = new Label("My Label");
    private Scene scene = new Scene(label, 200, 100);

    public View(){
        addClickManagement();
    }

    private void addClickManagement(){
       EventHandler<MouseEvent> eventHandler = new EventHandler<MouseEvent>() {
            @Override
            public void handle(MouseEvent e) {
                double x = e.getX();
                double y = e.getY();
                System.out.println("clicked");
            }
        };
        scene.addEventFilter(MouseEvent.MOUSE_CLICKED, eventHandler);
    }
    Scene getScene(){
        return scene;
    }
}
```

Observe/**Observable**:

- La **view** e' osservabile da Controller
- Il Modello e' osservabile dalla View

```
public class View extends Observable {....

..
    // cambiamo costruttore:

    public View(Observer obs){
       addClickManagement();
       addObserver(obs);
   }



    Void handle(MouseEvent e) {
       setChanged();    // very important!
       notifyObservers(p);
```

Observe/**Observable**:

- La view e' osservabile da Controller
- Il **Modello** e' osservabile dalla View

```
public class GameModel extends Observable {

    private CartesianCircle circle;

    public GameModel(Observer obs) {
        this.circle = new CartesianCircle(20, 20, 50);
        this.addObserver(obs);
    }

    void checkPoint(CartesianPoint here){
        setChanged();    // very important!
        notifyObservers(here);
    }
}
```

- Il **Modello** e' osservabile dalla View, ma prima COMPLETIAMO LA LOGICA:
- Contiamo click
- Se nel centro, registriamo vittoria.

Ci servira anche in CartesianPoint:

```
public double distanceFrom(CartesianPoint otherP){
    double c1 = otherP.x-this.x;
    double c2 = otherP.y-this.y;
    double d = sqrt(c1* c1 + c2 * c2);
    return d;
}

E:
boolean contains(CartesianPoint p){
    return center.distanceFrom(p)<this.radius;
}

In: CartesianCircle
```

```
public class GameModel extends Observable {

    private CartesianCircle circle;
    private int clickCount = 0;
...

void checkPoint(CartesianPoint here){

    this.hasWon = circle.contains(here);
    this.clickCount++;

     String msg = (hasWon ? "WON!" : "") + Integer.toString(this.clickCount);
     setChanged();   // very important!
     notifyObservers(msg);

}
```

Observe/Observable:

- La view e' osservabile da Controller -> Controller **osserva** la View

```
public class App extends Application implements Observer {

    public static void main(String[] args) {
        launch(args);
    }

    private View view = new View(this);
    public void update(Observable obj, Object arg) {
        CartesianPoint p = (CartesianPoint)arg;
        System.out.println("got click:");
        System.out.println(p.x);
        System.out.println(p.y);
        // tell to model:
        this.model.checkPoint(p);
    }
    @Override
    public void s
....
```

Observe/Observable:

- La view e' osservabile da Controller -> Controller **osserva** la View e cambia il **MODELLO**

```
public class App extends Application implements Observer {

    public static void main(String[] args) {
        launch(args);
    }

    private View view = new View(this);
    private GameModel model = new GameModel();

    public void update(Observable obj, Object arg) {
        CartesianPoint p = (CartesianPoint)arg;
        System.out.println("got click:");
        System.out.println(p.x);
        System.out.println(p.y);

        // call model:
         model.clicked(p);
        //...
    }

....
```

Observe/Observable:

- Controller **osserva** la View e cambia il **MODELLO, OMG! Loop infinito!**

Observe/Observable: **Loop infinito!**

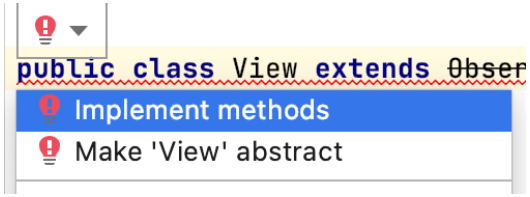
- Evento click in V
- V innesca il suo observer, i.e. APP
- App involve metodo "checkPoint"
- "checkPoint" cambia il M
- M innesca il suo observer, che e' APP !

FIX:
L' observer del modello e' **la VIEW!**

```
..
private View view = new View(this);
// NOO ! private GameModel model = new GameModel(this);
private GameModel model = new GameModel(view);
```

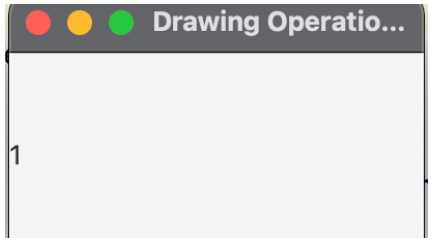
```
View is not abstract and does not override abstract method update...
```

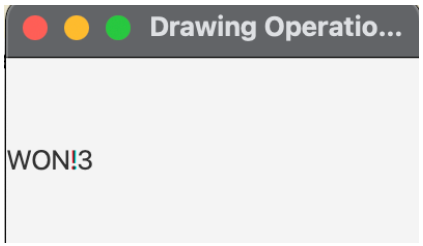
FIX:
L' observer del modello e' **la VIEW..**
IntelliJ ci aiuta...




```
@Override
public void update(Observable o, Object arg) {
    this.label.setText((String)arg); // riceviamo una stringa.

}
```

Run...





Vinto al III click...

```
public class App extends Application implements Observer {

    public static void main(String[] args) {
        launch(args);
    }

    private View view = new View(this);
    // NOO ! private GameModel model = new GameModel(this);
    private GameModel model = new GameModel(view);

    @Override
    public void start(Stage primaryStage) {
        primaryStage.setTitle("Drawing Operations Test");

        primaryStage.setScene(this.view.getScene());
        primaryStage.show();
    }

    @Override
    public void update(Observable o, Object arg) {

        //System.out.println("got it!!");
        CartesianPoint p = (CartesianPoint) arg;
        System.out.println(p.x);
        System.out.println(p.y);

        // tell to model:
        this.model.checkPoint(p);

    }
}
```

Code: CartesianCircle

```
public class CartesianCircle {

    private CartesianPoint center;
    private double radius;

    CartesianCircle(double x, double y, double radius){
        this.center = new CartesianPoint(x,y);
        this.radius = radius;
    }

    boolean contains(CartesianPoint p){
        return center.distanceFrom(p)<this.radius;
    }
}
```

Code: CartesianCircle/Point

```
public class CartesianCircle {

    private CartesianPoint center;
    private double radius;

    CartesianCircle(double x, double y, double radius){
        this.center = new CartesianPoint(x,y);
        this.radius = radius;
    }

    boolean contains(CartesianPoint p){
        return center.distanceFrom(p)<this.radius;
    }
}


--


public class CartesianPoint {

    double x,y;

    public CartesianPoint(double x, double y){
        this.x = x;
        this.y = y;
    }

    public double distanceFrom(CartesianPoint otherP){
        double c1 = otherP.x-this.x;
        double c2 = otherP.y-this.y;
        double d = sqrt(c1* c1 + c2 * c2);
        return d;
    }
}
```

Code: GameModel

```
public class GameModel extends Observable {

    private CartesianCircle circle;
    private int clickCount = 0;
    private boolean hasWon = false;

    public GameModel(View obs) {
        this.circle = new CartesianCircle(20, 20, 50);
        this.addObserver(obs);
    }


    void checkPoint(CartesianPoint here){

        this.hasWon = circle.contains(here);
        this.clickCount++;

        String msg = (hasWon ? "WON!" : "") + Integer.toString(this.clickCount);

        setChanged();   // very important!
        notifyObservers(msg);
    }


}
```

Code: View

```
public class View extends Observable implements Observer {
    private Label label;
    private Scene scene;

    public View(Observer obs){
      label = new Label("My Label");
      scene = new Scene(label, 200, 100);
      addClickManagement();
      addObserver(obs);
    }

    private void addClickManagement(){
        EventHandler<MouseEvent> eventHandler = new EventHandler<MouseEvent>() {
            @Override
            public void handle(MouseEvent e) {
                double x = e.getX();
                double y = e.getY();
                System.out.println("clicked");

                setChanged();   // very important!

                CartesianPoint p = new CartesianPoint(x,y);
                notifyObservers(p);
            }
        }
        scene.addEventFilter(MouseEvent.MOUSE_CLICKED, eventHandler);
    }

    Scene getScene(){
        return scene;
    }

    @Override
    public void update(Observable o, Object arg) {
        this.label.setText((String)arg);
    }
}
```