

# Anatomia di un'applicazione

# Slot Machine

- 1) Scrivere un'applicazione che implementi una slot machine. Tutto il codice deve essere documentato con Javadoc. L'applicazione presenterà una finestra che ricordi vagamente la seguente immagine



2) I contatori sono due:

Credito (indica i soldi disponibili, espressi in centesimi, inizialmente è 0)

Punteggio (inizialmente è 0)

3) Le monete inizialmente sono 3. Sono dei cerchi su ciascuno dei quali è riportata la dicitura “1 Euro”.

4) Cliccando su una moneta, questa sparisce e il credito viene aumentato di 100.

5) I bottoni sono :

Nuova partita

Spin (disabilitato se il punteggio è zero)

Pay (disabilitato se il credito è zero)

6) Le ruote dei simboli sono tre, uguali tra loro. Ciascuna contiene gli stessi sei simboli (delle figure geometriche stilizzate: barra inclinata a destra, rombo, cerchio, ecc., scegliete voi). Ogni ruota mostra un solo simbolo alla volta.

7) Cliccando sul tasto “Nuova partita”, se il credito è inferiore a 100 appare una finestra di pop-up che dice “non hai credito sufficiente”. Altrimenti il credito viene diminuito di 100 e il punteggio viene settato a 128.

8) Se Il tasto Spin è abilitato, cliccandolo i simboli delle tre ruote vengono scelti in modo casuale. Ad ogni pressione del tasto “Spin” il punteggio viene dimezzato (ma se è 1 diventa 0).

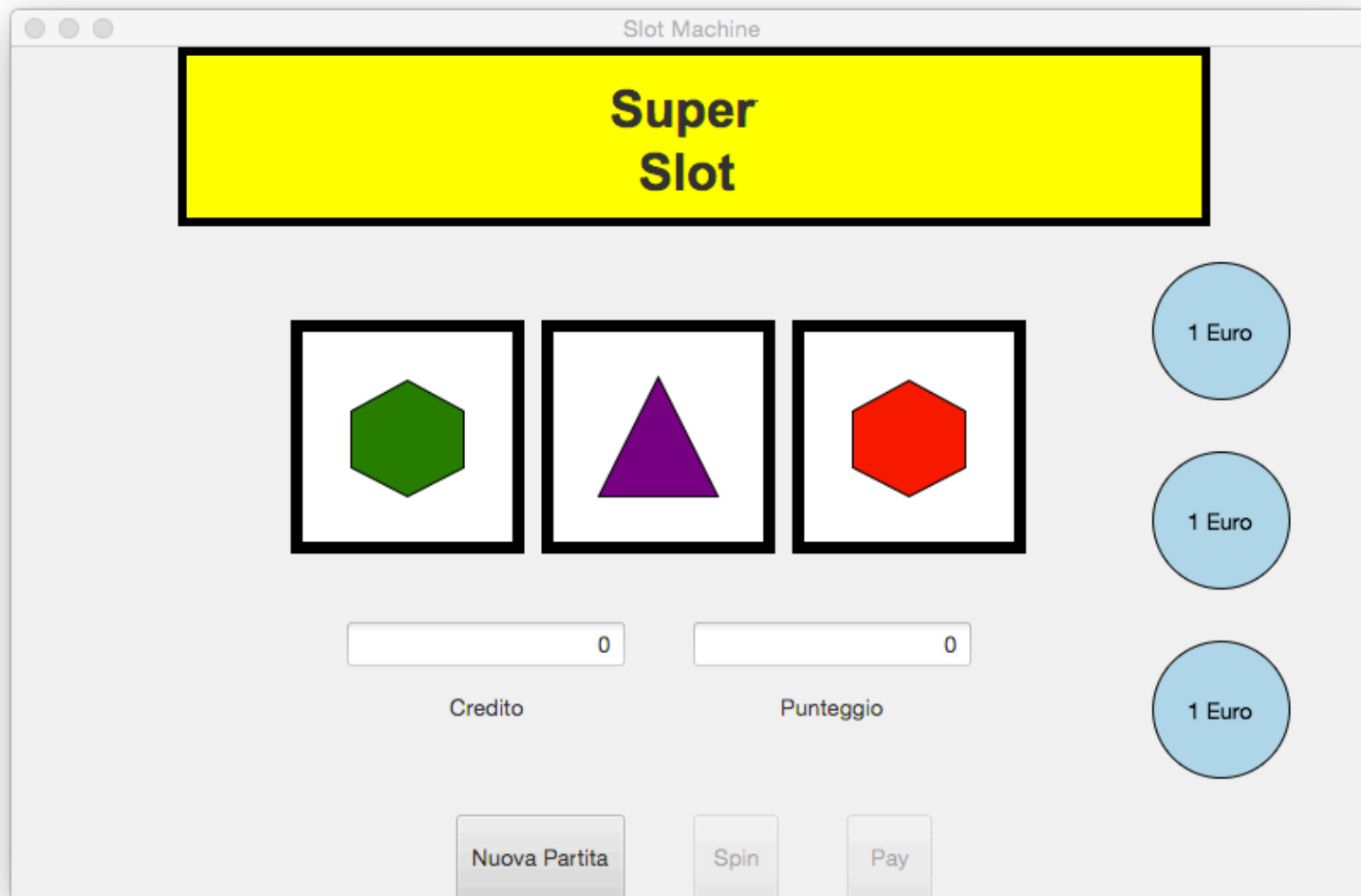
9) Cliccando su una delle ruote dei simboli, il suo simbolo viene modificato (ma solo se il punteggio non è zero) scegliendolo in modo casuale (quelli delle altre ruote restano immutati). Il punteggio viene dimezzato.

10) Se i simboli mostrati dalle tre ruote sono uguali, appare una finestra di pop-up che dice “Hai vinto”, il credito viene incrementato di un valore pari al punteggio moltiplicato per 100, il punteggio diventa zero.

11) Cliccando sul tasto “Pay” appare un pop-up che dice “Hai vinto XX Euro”, dove XX è il credito diviso 100. Il sistema viene resettato nella condizione iniziale.

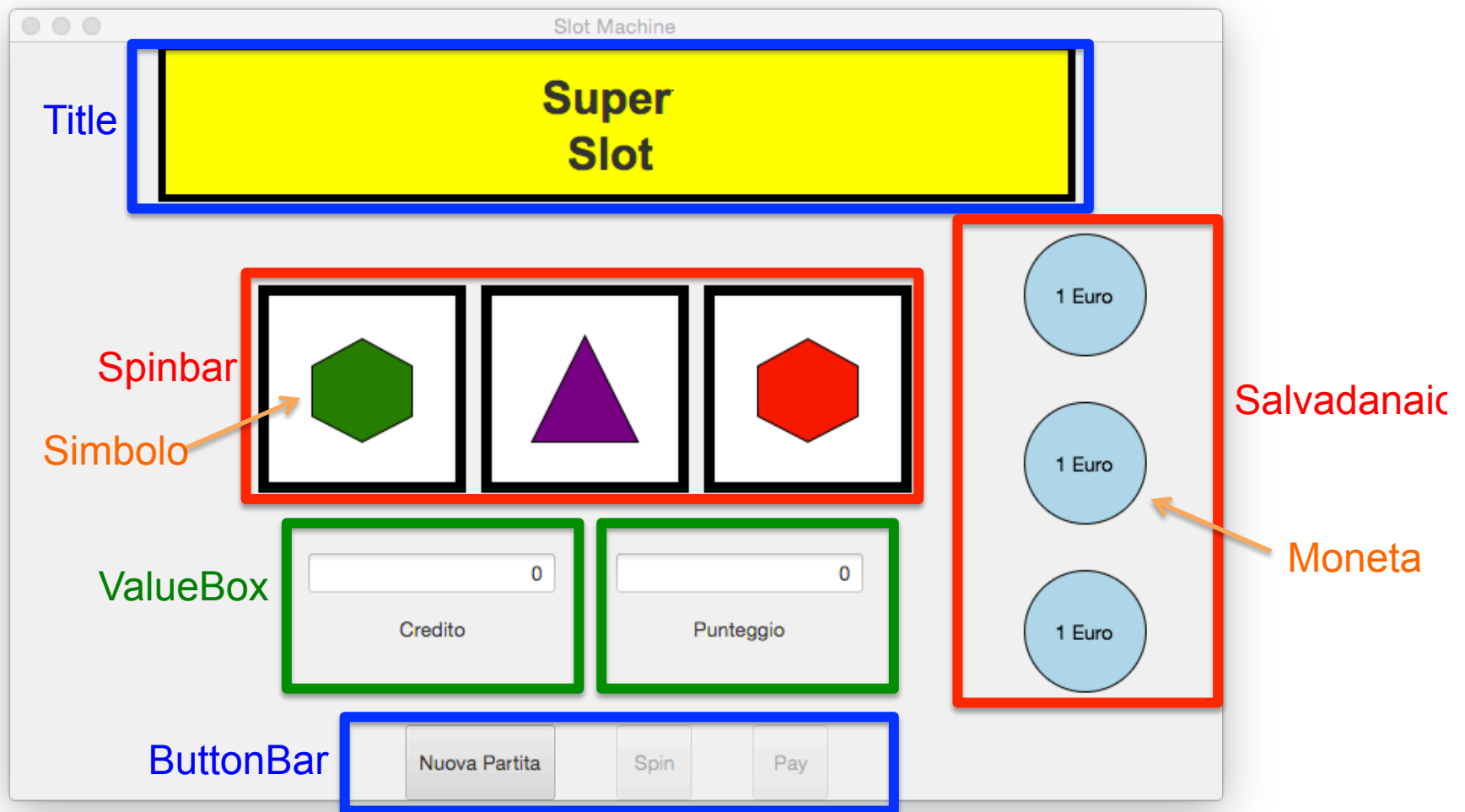
# Soluzione

1. Disegniamo come sarà l'aspetto della nostra applicazione



# Soluzione: Componenti logiche

Individuiamo le componenti logiche



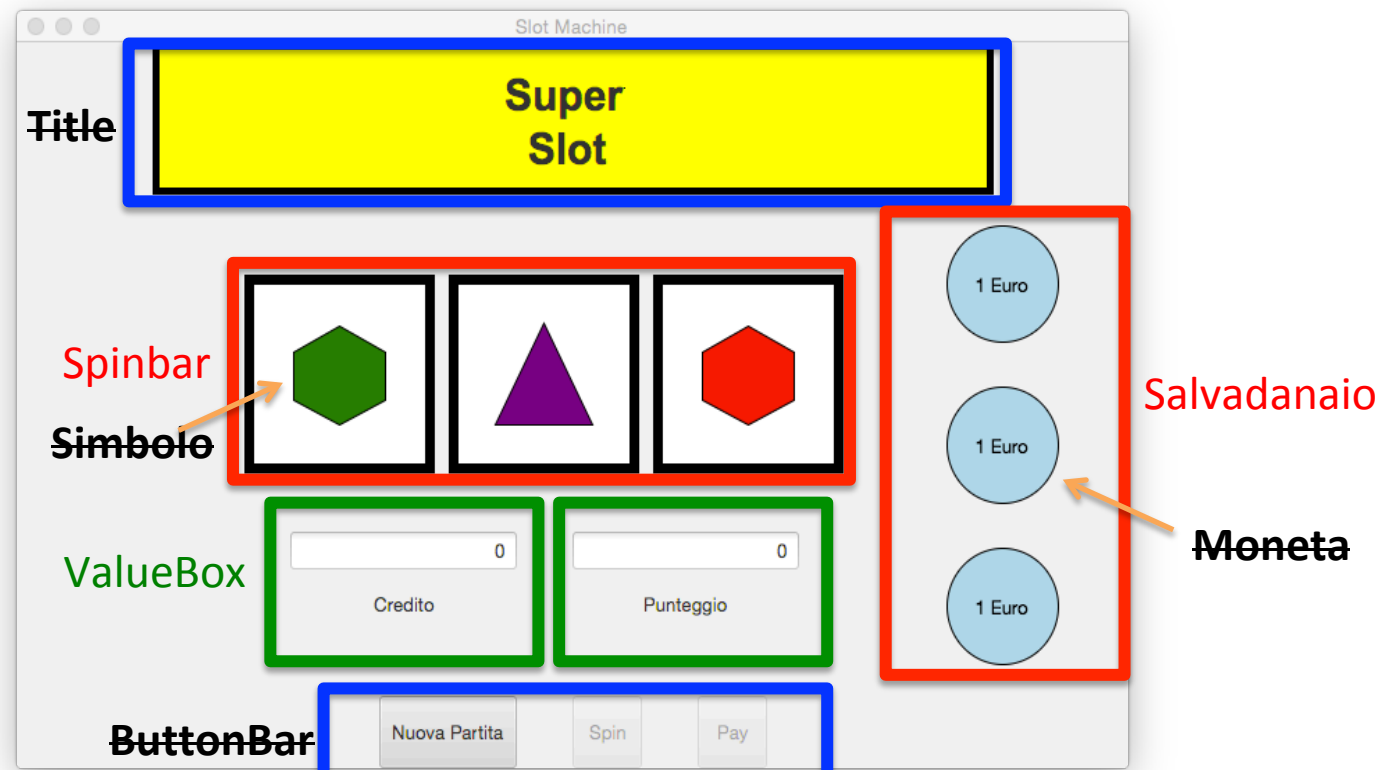


# SlotMachine - 1

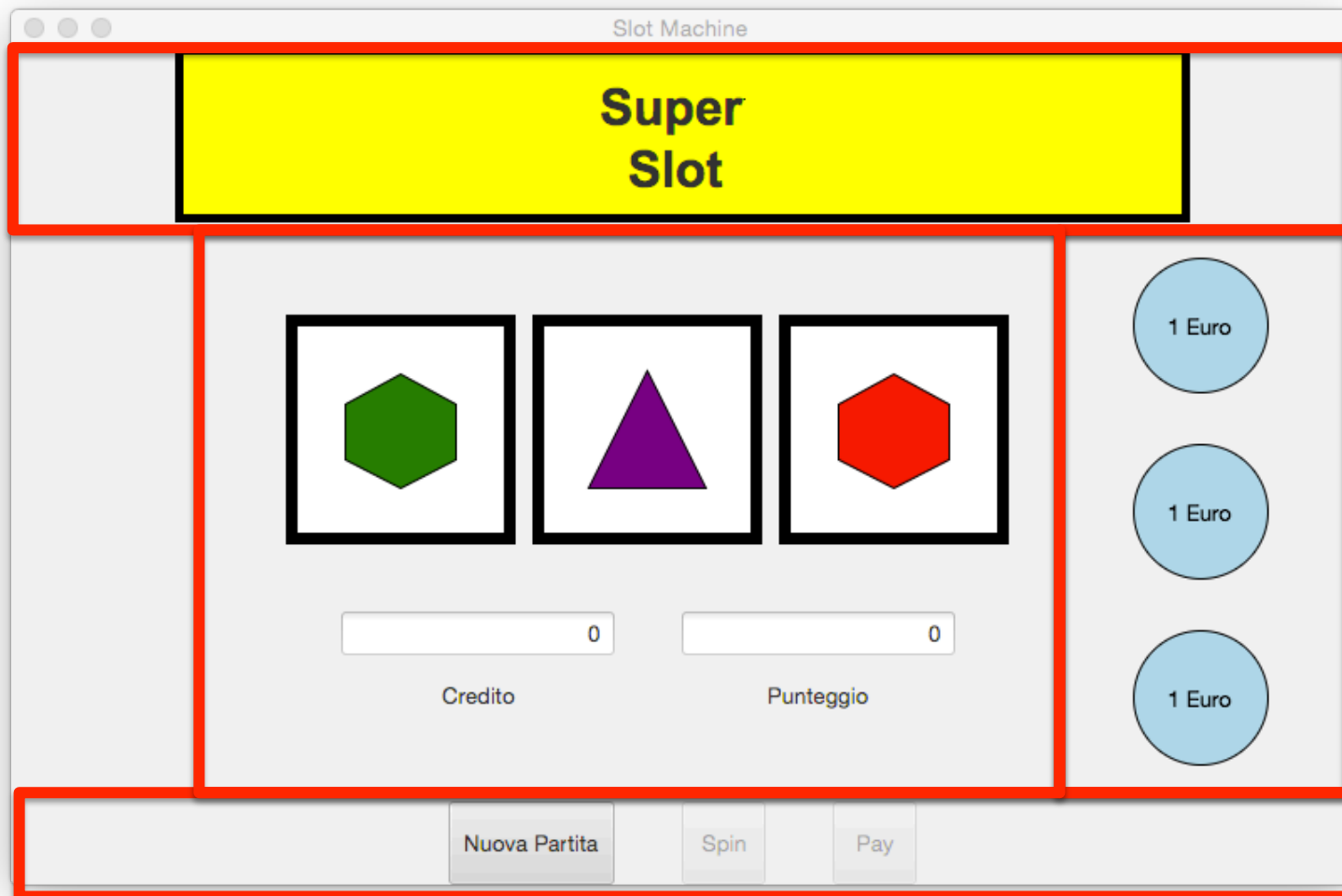
```
public class SlotMachine extends Application {  
    public static final int NUM_MONETE = 3; // numero di monete disponibili  
    public static final int NUM_SPINNERS = 3; // numero di simboli sulla slot machine  
    public static final int NUM_TIPI = 6; // numero di diversi tipi di simbolo  
    public static final int NPOINTS_PER_MONETA = 100; // numero di punti per moneta  
    public static final int COSTO_PARTITA = 100; // numero di punti per partita  
    public static final int PUNTI_PER_PARTITA = 128; // numero di punti per partita  
    ... (IV )... (methods) ...  
    public void start(Stage primaryStage) {  
        Scene scene = new Scene(this.prepareSceneContent(), 800, 500);  
        mainWindow = primaryStage;    primaryStage.setTitle("Slot Machine");  
        primaryStage.setScene(scene    primaryStage.centerOnScreen();  
        primaryStage.show();  
    }  
    public static void main(String[] args) { launch(args); }  
}
```

# SlotMachine – 2: instance variables

```
Stage mainWindow = null;  
ValueBox creditBox = null;  
ValueBox punteggioBox = null;  
Spinbar spinbar = null;  
Salvadanaio salvadanaio = null;
```



# Layout



# Title

```
public class Title extends Group {  
  
    public Title() {  
        Label lab = new Label("Super\n Slot");  
        lab.setAlignment(Pos.CENTER);  
        lab.setFont(Font.font("Arial", FontWeight.BOLD, 30));  
        lab.setLayoutX(250);  
        lab.setLayoutY(15);  
        Rectangle rect = new Rectangle(600, 100);  
        rect.setFill(Color.YELLOW);  
        rect.setStroke(Color.BLACK);  
        rect.setStrokeWidth(5);  
        this.getChildren().addAll(rect,lab);  
    }  
}
```



```
class MyCircle extends Circle{
    MyCircle() {
        super(35);
        setLayoutX(65);
        setLayoutY(65);
        setStroke(Color.BLACK);
    }
}
```

```
class MyRect extends Rectangle{
    MyRect() {
        super(70,70);
        setLayoutX(30);
        setLayoutY(30);
        setStroke(Color.BLACK);
    }
}
```

```
class MyTriangle extends Polygon{
    MyTriangle() {
        super(35.0,0.0, 0.0,70.0,
70.0,70.0);
        setLayoutX(30);
        setLayoutY(30);
        setStroke(Color.BLACK);
    }
}
```

```
class MyHexagon extends Polygon{
    MyHexagon() {
        super(35.0,2.0, 68.0,20.0, 68.0,53.0,
35.0,70.0, 2.0,53.0, 2.0,20.0 );
        setLayoutX(30);
        setLayoutY(30);
        setStroke(Color.BLACK);
    }
}
```

S  
i  
m  
b  
o  
l  
o

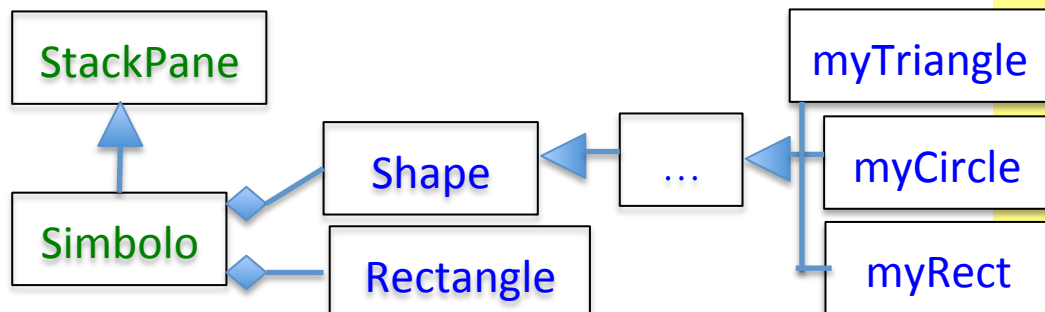
```

public class Simbolo extends StackPane {
    String classi[]={"MyRect","MyTriangle","MyHexagon","MyCircle"};
    public Simbolo(EventHandler simbolListener, int tipo, int colorIndex){
        addEventFilter(MouseEvent.MOUSE_CLICKED,simbolListener);
        setId(""+tipo+colorIndex);
        Shape tmp = null;
        String packname=this.getClass().getPackage().getName();
        String className=packname+"."+classi[tipo];
        try {
            tmp=(Shape)(Class.forName(className).newInstance());
        } catch (ClassNotFoundException | InstantiationException
                | IllegalAccessException ex) {
            System.err.println("Impossibile istanziare la classe "+className);
        }
        tmp.setFill(colorArray[colorIndex]);
    }

```



# Simbolo

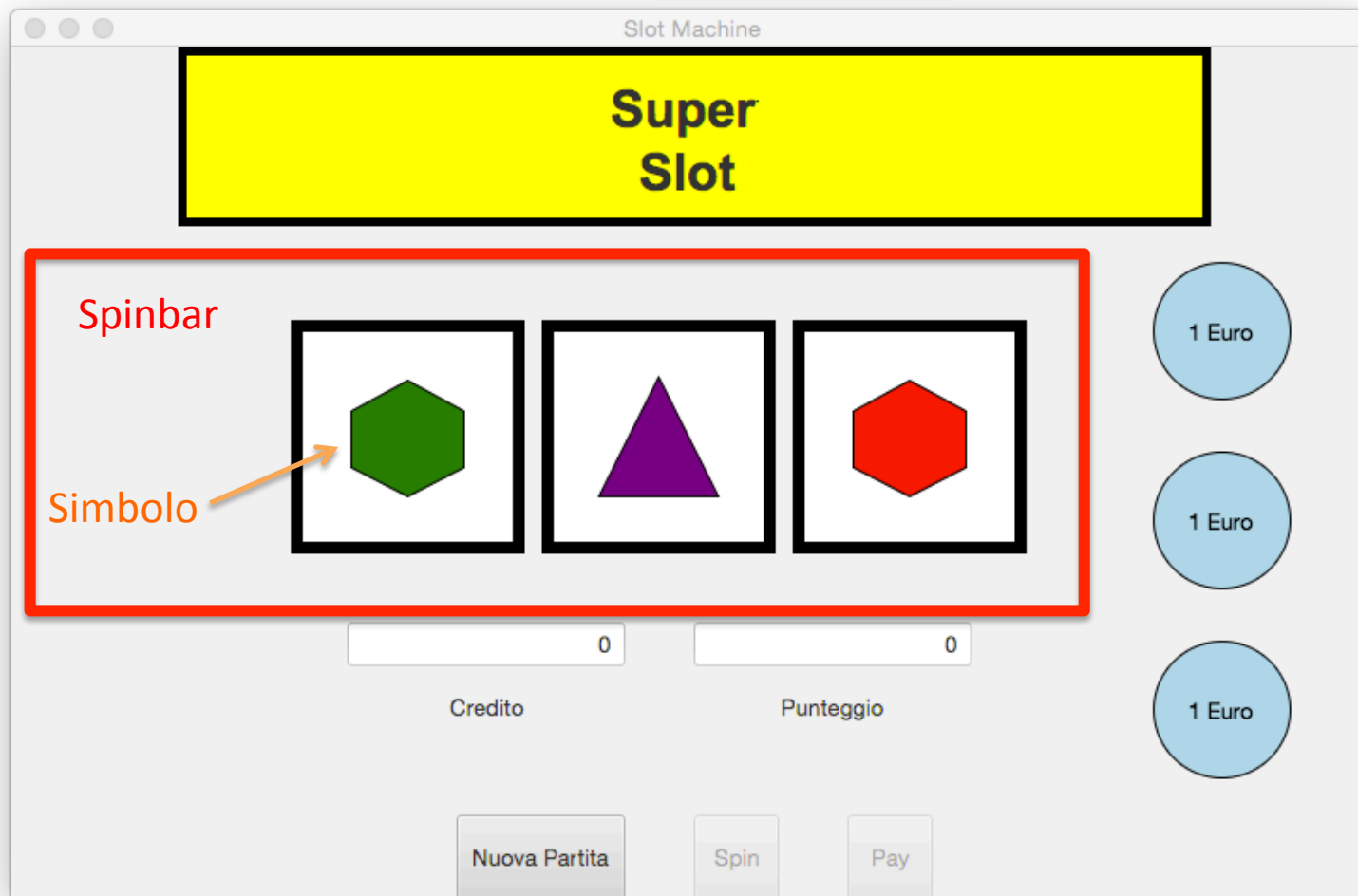
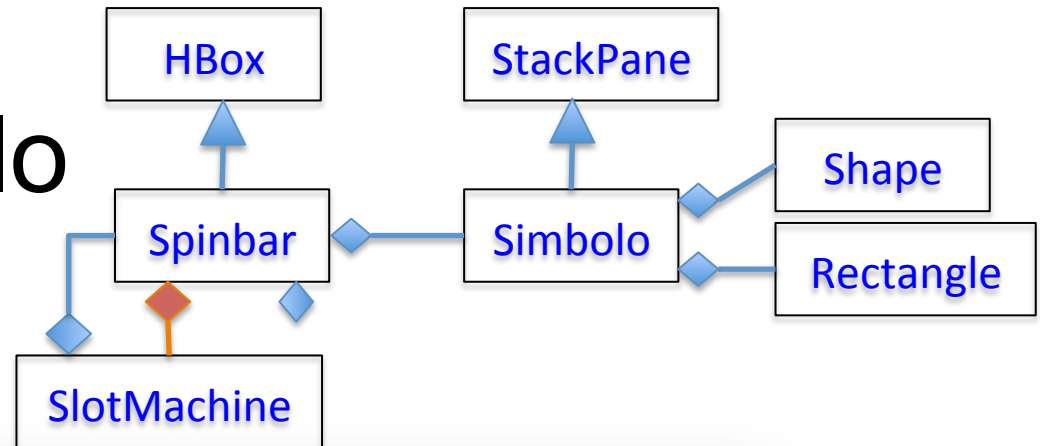


```

// ===== enclosing rectangle
Rectangle rect= new Rectangle(130,130);
rect.setFill(Color.WHITE);
rect.setStrokeWidth(7);
rect.setStroke(Color.BLACK);
// ===== put things together
getChildren().addAll(rect,tmp);

```

# Spinbar e Simbolo



# Spinbar

```
public class Spinbar extends Hbox implements EventHandler {  
    Simbolo simbolo[] = new Simbolo[SlotMachine.NUM_SPINNERS];  
    SlotMachine sm = null;  
    public Spinbar(SlotMachine sm)  
        setAlignment(Pos.CENTER);  
        // spazio orizzontale tra  
        // le componenti  
        setSpacing(10);  
        this.sm = sm;  
        initialize();  
}
```

```
public void initialize() {  
    getChildren().clear();  
    for (int i = 0; i < SlotMachine.NUM_SPINNERS; i++) {  
        int tipo = SlotMachine.randomGenerator.  
            nextInt(SlotMachine.NUM_SYMBOL_TYPES);  
        int colore = SlotMachine.randomGenerator.  
            nextInt(SlotMachine.NUM_SYMBOL_COLORS);  
        simbolo[i] = new Simbolo(tipo, colore);  
        simbolo[i].addEventFilter(  
            MouseEvent.MOUSE_CLICKED, this);  
    } getChildren().addAll(simbolo);  
    // evita di iniziare con tutti i simboli già uguali!  
    if (areSymbolsEqual()) initialize();  
}
```

```
public boolean areSymbolsEqual() {  
    for (int i = 1; i < SlotMachine.NUM_SPINNERS; i++)  
        if (!simbolo[0].getId().equals(simbolo[i].getId())) return false;  
    return true; }  
}
```





```
public class Spinbar extends HBox {
```

```
    Simbolo simbolo[] = new Simbolo[SlotMachine.NUM_SPINNERS];
```

```
    SlotMachine sm = null;
```

```
    public Spinbar(SlotMachine sm) {
```

```
        setAlignment(Pos.CENTER);
```

```
        // spazio orizzontale tra
```

```
        // le componenti
```

```
        setSpacing(10);
```

```
        this.sm = sm;
```

```
        initialize();
```

```
    }
```

# Nota

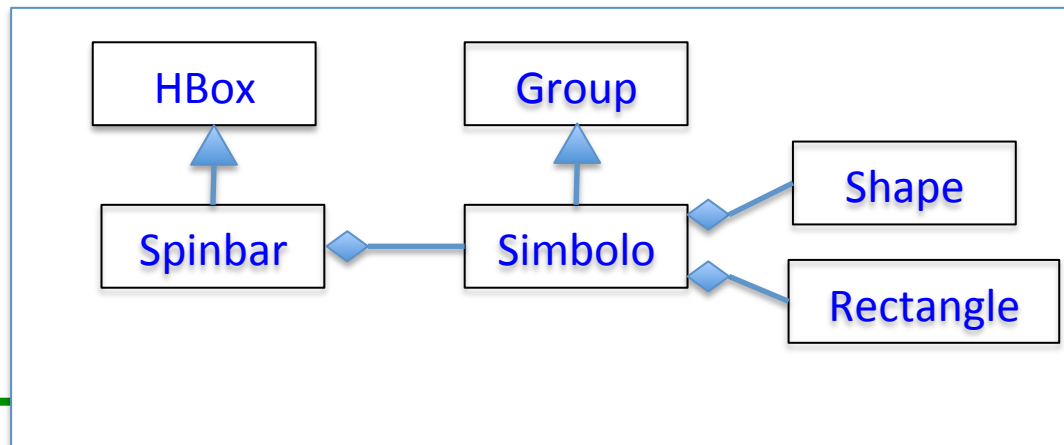
Le inizializzazioni fatte  
sulle Instance Variables  
vengono eseguite  
all'inizio del costruttore

# Spinbar

```
public void handle(Event t) {  
    if (!sm.payPoints()) return; // do not do any action if no points available  
    // scopri qual'è la posizione del simbolo nella spinbar  
    Simbolo s = (Simbolo) (t.getSource());  
    for (int i = 0; i < SlotMachine.NUM_SPINNERS; i++) {  
        if (s == simbolo[i]) spinElement(i);  
    }  
    t.consume();  
    if (areSymbolsEqual()) {  
        sm.declareVictory();  
    }  
}
```

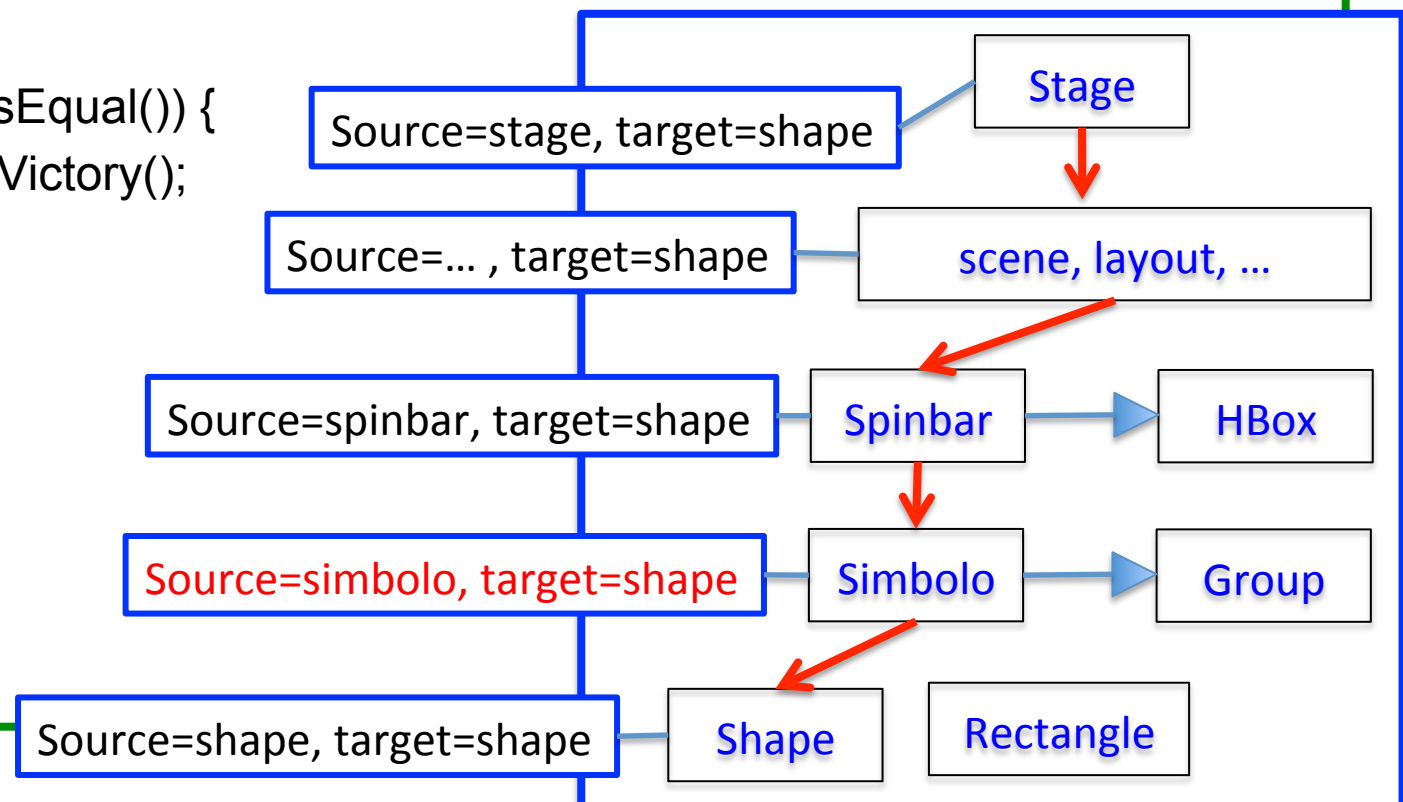
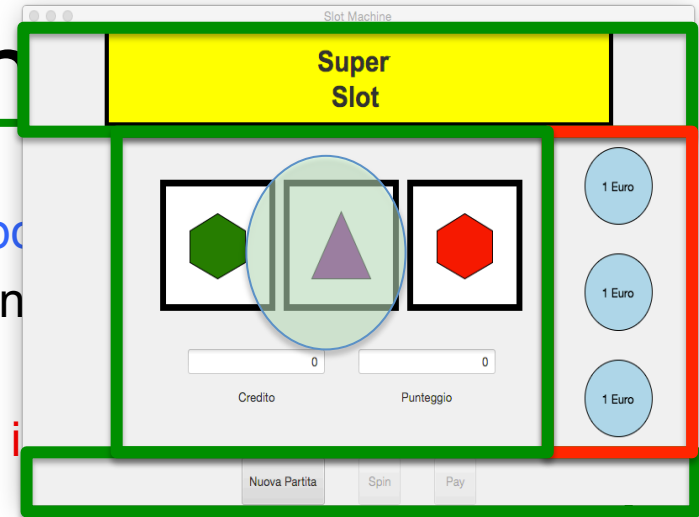
SlotMachine

declareVictory()  
payPoints()



# Spinbar.SymbolListener

```
public void handle(Event t) {
    if (!sm.payPoints()) return; // disable action if no po
    // scopri qual'è la posizione del simbolo nella spin
    Simbolo s = (Simbolo) (t.getSource());
    for (int i = 0; i < SlotMachine.NUM_SPINNERS; i)
        if (s == simbolo[i]) spinElement(i);
    }
    t.consume();
    if (areSymbolsEqual()) {
        sm.declareVictory();
    }
}
}
```



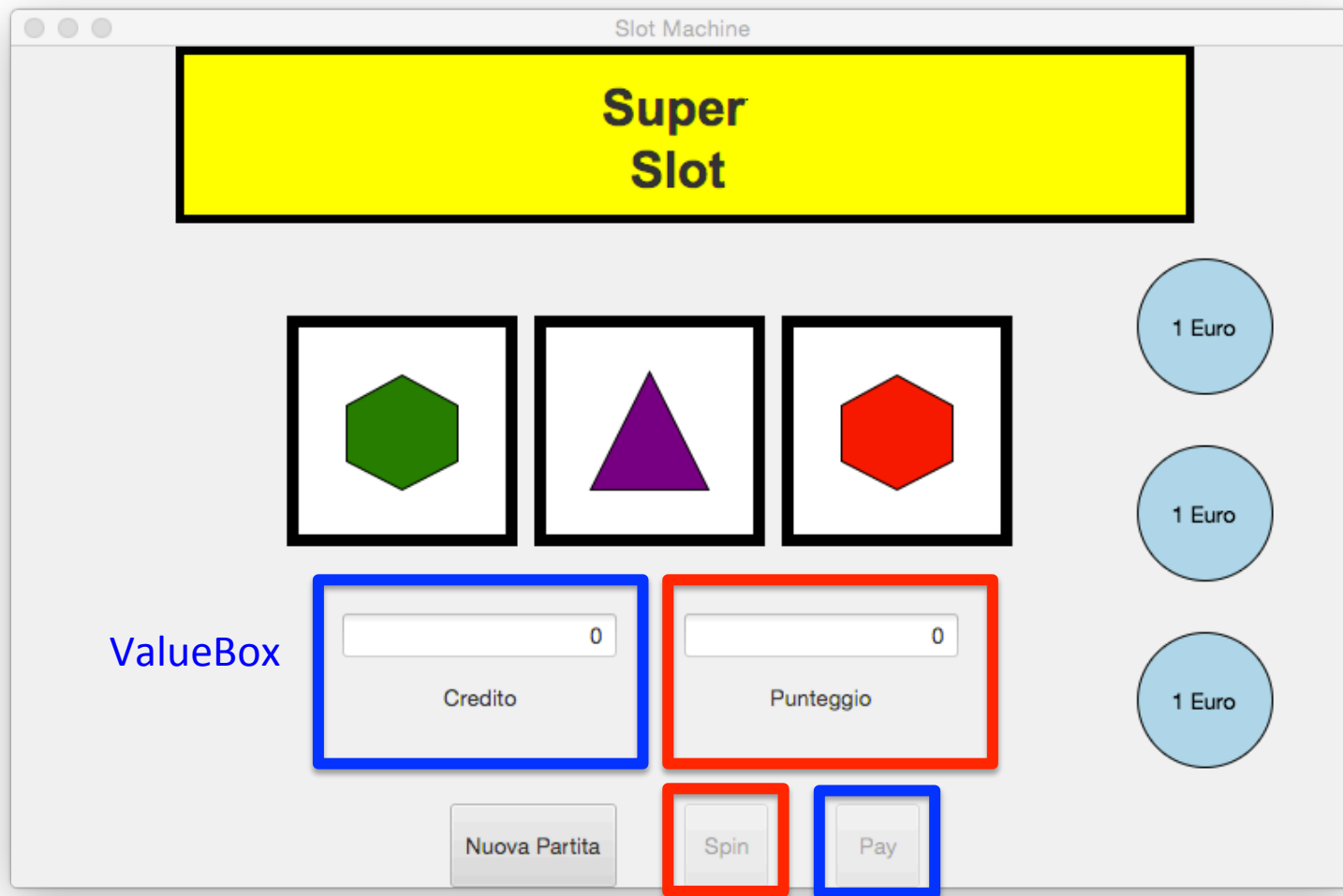
# Spinbar



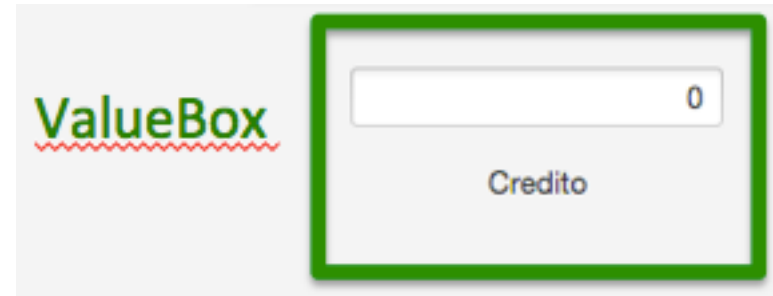
```
public void spinElement(int i) {
    int tipo = SlotMachine.randomGenerator.
        nextInt(SlotMachine.NUM_TIPI);
    simbolo[i] = new Simbolo(tipo);
    //System.out.println("replace simbolo " + i);
    this.getChildren().remove(i);
    this.getChildren().add(i, simbolo[i]);
}

public void spinAll() {
    if (!sm.payPoints()) return;
    for (int i = 0; i < SlotMachine.NUM_SPINNERS; i++) {
        spinElement(i);
    }
    if (areSymbolsEqual()) sm.declareVictory();
}
```

# ValueBox: associazione logica



# ValueBox



```
public class ValueBox extends VBox {  
    TextField txt = null;  
    private int value=0;  
    Button associatedButton=null;  
    public ValueBox(String label,  
        Button associatedButton) {  
        this.associatedButton=associatedButton;  
        txt=new TextField("--");  
        txt.setEditable(false);  
        txt.setAlignment(Pos.CENTER_RIGHT);  
        Label space=new Label(" ");  
        Label lbl=new Label(label);  
        this.getChildren().addAll(txt,space,lbl);  
        this.setAlignment(Pos.CENTER);  
    }  
}
```

Structure

```
public void reset() {  
    setValue(0);  
}  
public void setValue(int value) {  
    this.value=value;  
    txt.setText(String.valueOf(value));  
    associatedButton.setDisable(value==0);  
}  
public int getValue() {return value;}  
public void incrementValue(int delta){  
    setValue(value+delta);  
}  
}
```

Logic

# SlotMachine – 3 – business methods

```
public boolean payPoints() {
    int punti = punteggioBox.getValue();
    if (punti == 0) {
        return false;
    }
    punteggioBox.setValue(punti / 2);
    return true;
}
```

```
public void declareVictory() {
    int points = punteggioBox.getValue();
    punteggioBox.setValue(0);
    showPopup("Hai vinto!");
    creditBox.incrementValue(points * 100);
}
```

SlotMachine

declareVictory()  
payPoints()

```
/**
 * resetta le varie componenti per tornare
 * allo stato iniziale
 */
void reset() {
    creditBox.reset();
    punteggioBox.reset();
    salvadanaio.initialize();
    spinbar.initialize();
}
```

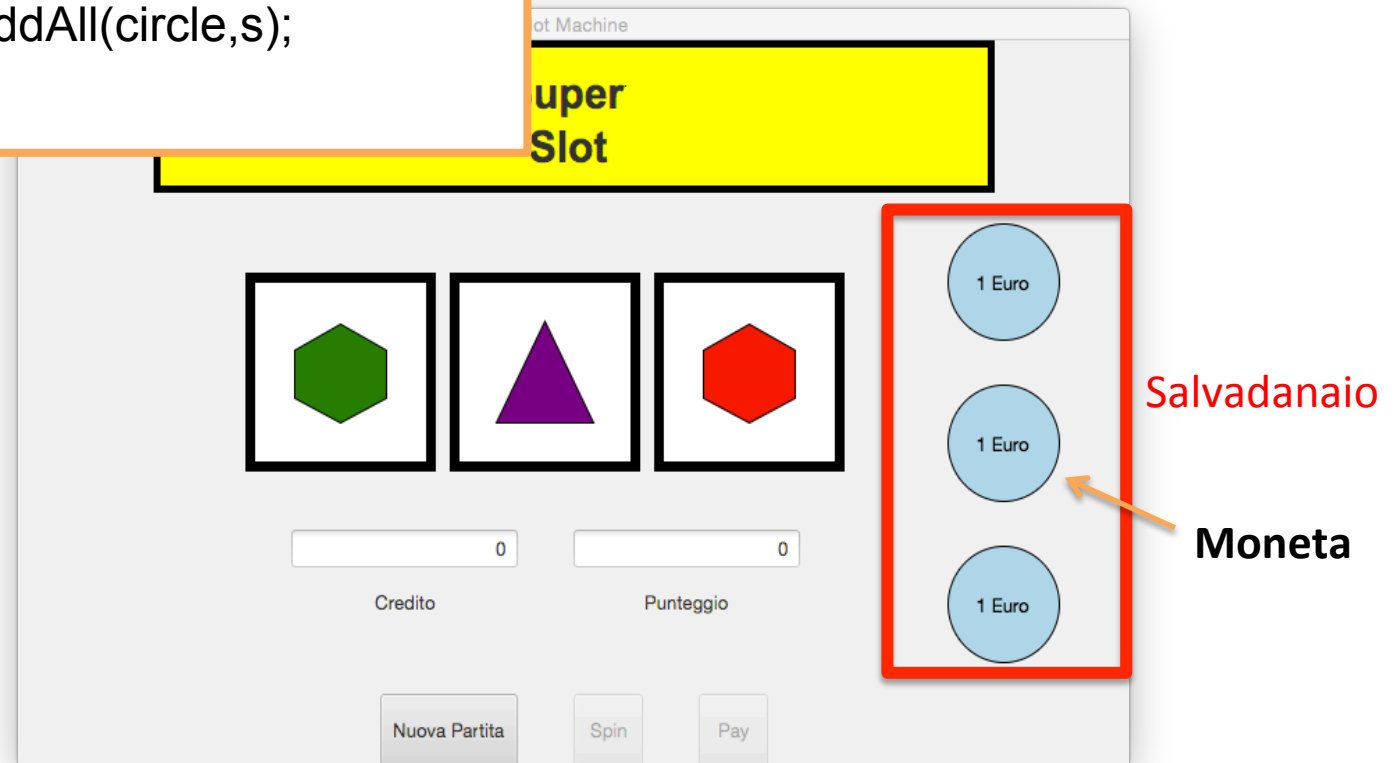
# SlotMachine – 4 – modal dialog

```
/**
 * Creates a modal pop-up window, i.e. a window that blocks actions on the
 * window which generate it until the pop-up is closed
 * @param message message to be shown in the popup
 */
public void showPopup(String message) {
    Label label= new Label(message);
    label.setAlignment(Pos.CENTER);
    label.setFont(Font.font("Arial", FontWeight.BOLD, 20));
    Scene sc = new Scene(label, 500, 200);
    Stage stage = new Stage();
    stage.setScene(sc);
    stage.setX(100);
    stage.setY(100);
    stage.initModality(Modality.WINDOW_MODAL);
    stage.initOwner(mainWindow);
    stage.show();
}
```



# Moneta

```
public class Moneta extends  
javafx.scene.Group {  
    public Moneta(){  
        Circle circle = new Circle(40);  
        circle.setFill(Color.LIGHTBLUE);  
        circle.setStroke(Color.BLACK);  
        Text s = new Text ("1 Euro");  
        s.setLayoutX(-20);  
        s.setLayoutY(5);  
        this.getChildren().addAll(circle,s);  
    }  
}
```

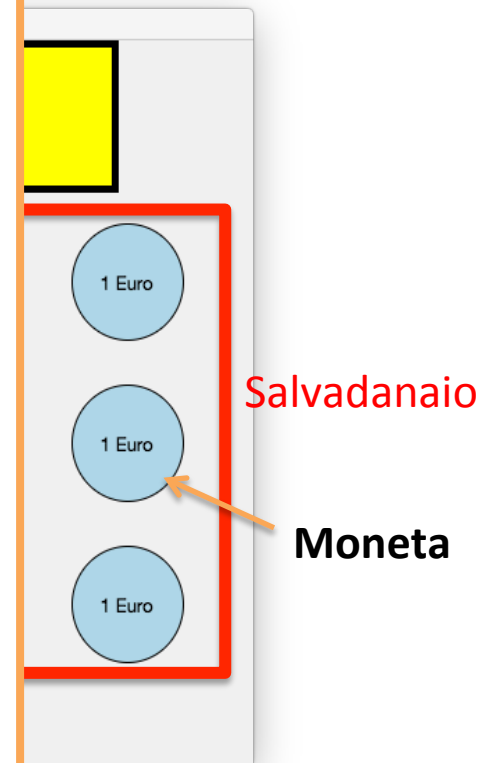


```

public class Salvadanaio extends VBox{
    SlotMachine sm=null;
    ListenerMonete monetaListener = null;
    public Salvadanaio(SlotMachine sm){
        this.sm=sm;
        setAlignment(Pos.CENTER);
        setSpacing(30); setPadding(new Insets(10, 50, 10, 10));
        monetaListener=new ListenerMonete();
        initialize();
    }
    public void initialize() {
        getChildren().clear();
        for (int i = 0; i < SlotMachine.NUM_MONETE; i++) {
            Moneta m= new Moneta();
            m.addEventFilter(MouseEvent.MOUSE_CLICKED,monetaListener);
            addMoneta(m);
        }
        public void addMoneta(Moneta m){
            getChildren().add(m);
        }
    }
}

```

# Salvadanaio-1

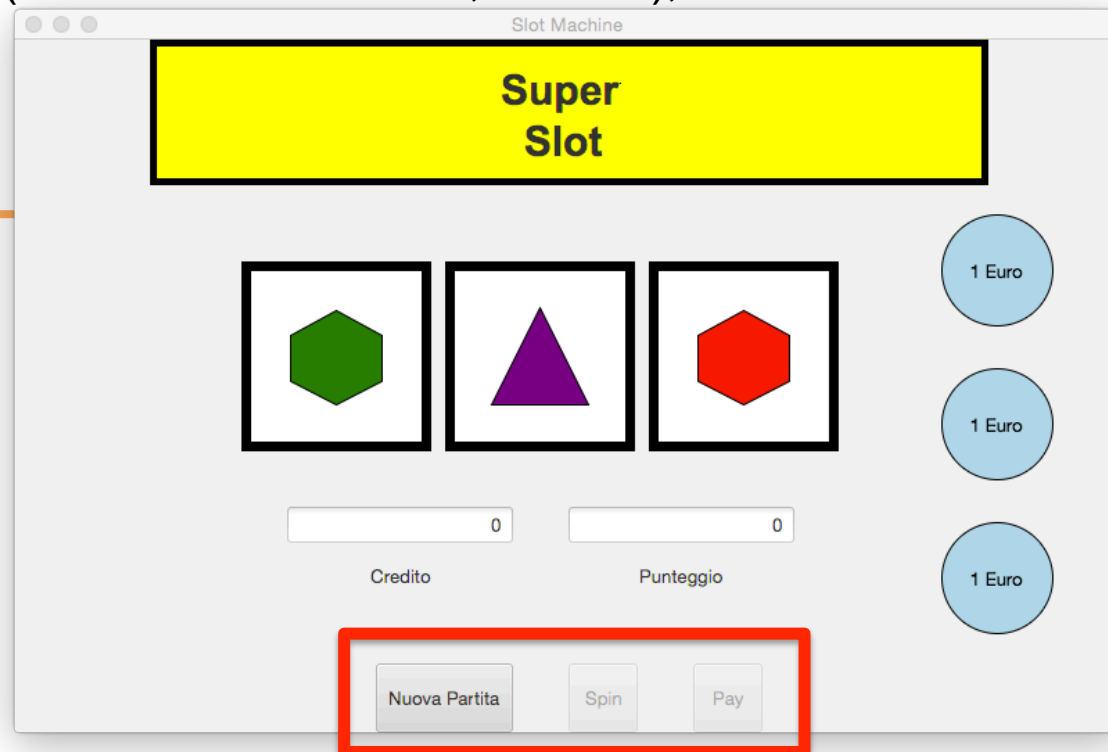


# Salvadanaio.ListenerMonete

```
public class ListenerMonete implements EventHandler {  
  
    public void handle(Event t) {  
        Moneta m = (Moneta) (t.getSource());  
        m.setVisible(false);  
        sm.creditBox.incrementValue(SlotMachine.NPOINTS_PER_MONETA);  
        t.consume();  
    }  
}
```

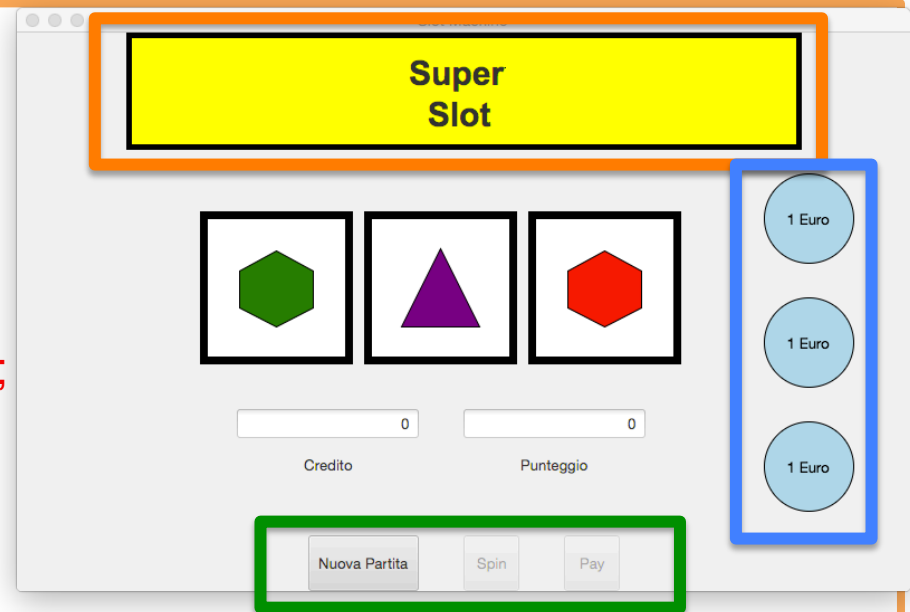
# SlotMachine – 5 : My Button

```
class MyButton extends Button {  
  
    MyButton(String label, boolean isDisabled, EventHandler listener) {  
        super(label);  
        setMinSize(50, 50);  
        setDisable(isDisabled);  
        addEventHandler(ActionEvent.ACTION, listener);  
    }  
}
```



# SlotMachine – 6 - prepareSceneContent

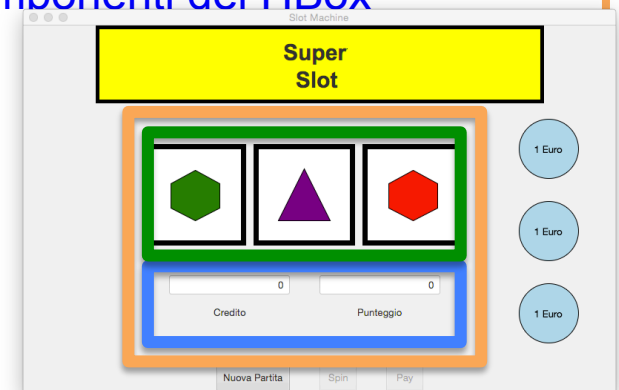
```
BorderPane prepareSceneContent() {  
    BorderPane border = new BorderPane();  
    // ===== TOP: titolo  
    Group g = new Title();  
    border.setTop(g);  
    BorderPane.setAlignment(g, Pos.CENTER);  
    // ===== RIGHT: le monete  
    salvadanaio = new Salvadanaio(this);  
    border.setRight(salvadanaio);  
    // ===== BOTTOM : i bottoni di controllo  
    HBox buttonbar = new HBox();  
    MyButton spinButton = new MyButton("Spin", true, new ListenerSpinButton());  
    MyButton payButton = new MyButton("Pay", true, new ListenerPayButton());  
    MyButton nuovaPartitaButton = new MyButton("Nuova Partita", false, new  
        ListenerNuovaPartitaButton());  
    buttonbar.getChildren().addAll(nuovaPartitaButton, spinButton, payButton);  
    buttonbar.setSpacing(40); // spazio orizzontale tra le componenti del HBox  
    buttonbar.setAlignment(Pos.CENTER);  
    border.setBottom(buttonbar);  
}
```



# SlotMachine – 6 - prepareSceneContent

```
// ===== CENTER: Spinbar e contatori
VBox centralBox = new VBox(); // componente che conterrà Spinner e Contatori
centralBox.setAlignment(Pos.CENTER);
centralBox.setSpacing(40); // spazio verticale tra le componenti del VBox
// spazio verticale tra la componente al centro e quella soprastante:
centralBox.setPadding(new Insets(0, 0, 0, 100));
spinbar = new Spinbar(this);
spinbar.setAlignment(Pos.CENTER);
HBox boxContatori = new HBox(); // contenitore dei contatori
creditBox = new ValueBox("Credito", payButton);
punteggioBox = new ValueBox("Punteggio", spinButton);
boxContatori.getChildren().addAll(creditBox, punteggioBox);
boxContatori.setAlignment(Pos.CENTER);
boxContatori.setSpacing(40); // spazio orizzontale tra le componenti del HBox
centralBox.getChildren().addAll(spinbar, boxContatori);
border.setCenter(centralBox);
reset(); // inizializza tutte le componenti
return border;
```

```
}
```



# SlotMachine – 7 – ButtonListeners 1

```
class ListenerNuovaPartitaButton implements EventHandler {  
    /**  
     * Controlla se è possibile avviare una nuova partita,  
     * e se sì regola i conti e inizializza  
     * @param t L'evento scatenante  
     */  
    public void handle(Event t) {  
        if (creditBox.getValue() < COSTO_PARTITA) {  
            showPopup("Non hai credito sufficiente");  
        } else {  
            spinbar.initialize();  
            creditBox.incrementValue(-COSTO_PARTITA);  
            punteggioBox.setValue(PUNTI_PER_PARTITA);  
        }  
    }  
}
```

# SlotMachine – 8 – ButtonListeners 2

```
class ListenerPayButton implements EventHandler {  
    /**  
     * Paga la vincita e resetta allo stato iniziale  
     * @param t L'evento scatenante  
     */  
    public void handle(Event t) {  
        int euro = creditBox.getValue() / 100;  
        String message = "Hai vinto " + euro + " Euro";  
        showPopup(message);  
        reset();  
    }  
}
```

```
class ListenerSpinButton implements EventHandler {  
    /**  
     * Richiedi che sia effettuato uno spin  
     * @param t L'evento scatenante  
     */  
    public void handle(Event t) {  
        spinbar.spinAll();  
    }  
}
```