

Programmazione

LAB 02

<http://www.iet.unipi.it/m.cimino/prg/>

Mario G.C.A. Cimino
Dipartimento di Ingegneria dell'Informazione

Seconda applicazione - stile e comportamento

2 of 7

1. Da NetBeans importare il codice svolto precedentemente, *prg-lab11-project.zip*, da *File* ⇒ Import Projects ⇒ From ZIP... (l'operazione complementare è *File* ⇒ Export Projects ⇒ To ZIP...)
2. **Specifiche:** aggiungere lo stile, un campo di testo e un pulsante come in figura. Premendo il pulsante viene inserita una moneta. Il campo di testo indica il numero di monete presenti nel salvadanaio.

3. **Specifiche:** si vuole anche poter inserire più monete in un colpo solo.
In tal caso si digita il numero di monete da inserire preceduto da '+'.
Classi suggerite: metodi setLayoutX, setStyle, setY



setOnAction, startsWith; classi ActionEvent, Integer.

4. **Specifiche:** archiviare il numero di monete su file binario (su un solo byte), quando si chiude l'applicazione. Ripristinare il numero di monete quando l'applicazione viene aperta. **Classi/metodi suggeriti:** read/write, FileOutputStream

5. Fare in modo che il colore della moneta cambi per segnalare situazioni di eccesso di monete (colore verde) oppure di problemi di accesso a file (rosso)

Classi

suggerite:

setFill, Color.

Riferimenti extra su
tinyurl.com

/javafx-cssref
/javafx-forms
/javafx-cssadv
/javafx-charts
/javafx-csscharts
/java-lambdaexp
/progr-fun
/progr-dec
/progr-imp
/java-iobasic
/java-bytestream
/java-trywithres



Salvadanaio\src\Salvadanaio.java

```
1 import javafx.application.*;
2 import javafx.stage.*;
3 import javafx.scene.*;
4 import javafx.scene.shape.*;
5 import javafx.scene.image.*;
6 import javafx.scene.control.*;
7 import javafx.scene.paint.*;
8 import javafx.event.*;
9 import java.io.*;
10
11 public class Salvadanaio extends Application {
12
13     private int monete = 0;
14
15     private Label la;
16     private Circle ci;
17     private ImageView im;
18     private TextField tf;
19     private Button bu;
20
21     public void start(Stage stage) {
22
23         la = new Label("Salvadanaio");
24         ci = new Circle(120, 65, 20, Color.GOLD);
25         im = new ImageView("file:../../myfiles/foto.png");
26         tf = new TextField("0");
27         bu = new Button("Inserisci");
28
29         la.setLayoutX(20);
30         la.setStyle("-fx-font-size: 40px; -fx-text-fill: green; -fx-font-weight: bold;");
31         im.setY(80);
32         tf.setMaxWidth(60);
33         tf.setLayoutX(110); tf.setLayoutY(370);
34         tf.setStyle("-fx-font-size: 20px; -fx-text-fill: blue;"); // (00)
```

```

35     bu.setLayoutX(60); bu.setLayoutY(420);
36     bu.setStyle("-fx-font-size: 30px;");
37
38     // (01)
39     bu.setOnAction((ActionEvent ev)->{inserisciMonete()});
40     stage.setOnCloseRequest((WindowEvent we) -> {conservaSalvadanaio();});
41
42     Group root = new Group(la, ci, im, tf, bu);
43     Scene scene = new Scene(root, 280, 500);
44     stage.setTitle("Applicazione JavaFX");
45     stage.setScene(scene);
46     stage.show();
47
48     prelevaSalvadanaio();
49 }
50
51 private void inserisciMonete() {
52     String x = tf.getText();
53     if (x.startsWith("+"))
54         monete = monete + Integer.parseInt(x);
55     else
56         monete++;
57     tf.setText(Integer.toString(monete));
58     if (monete > 255)
59         ci.setFill(Color.GREEN);
60 }
61
62 private void conservaSalvadanaio() { //(02)
63     try (FileOutputStream fout = new FileOutputStream("./myfiles/salvadanaio.bin")){ //(03)
64         fout.write(monete); //(04)
65     } catch (IOException ex) {
66         ci.setFill(Color.RED); // (05)
67         System.out.println("errore: impossibile conservare il salvadanaio!");
68     }
69 }

```

```

70
71 private void prelevaSalvadanaio() { //(02)
72     try (FileInputStream fin = new FileInputStream("./myfiles/salvadanaio.bin")){ //(03)
73         monete = fin.read(); //(04)
74     } catch (IOException ex) {
75         ci.setFill(Color.RED); // (05)
76         System.out.println("errore: impossibile prelevare il salvadanaio!");
77     }
78     tf.setText(Integer.toString(monete));
79 }
80 }
81
82 /*
83 Note:
84 (00) JavaFX CSS Reference Guide
85 https://docs.oracle.com/javafx/2/api/javafx/scene/doc-files/cssref.html
86 Forms
87 https://docs.oracle.com/javafx/2/get_started/form.htm
88 Advanced CSS
89 https://docs.oracle.com/javafx/2/css_tutorial/jfxpub-css_tutorial.htm
90 Charts
91 http://docs.oracle.com/javafx/2/charts/jfxpub-charts.htm
92 https://docs.oracle.com/javafx/2/charts/css-styles.htm
93
94 (01) Gestione degli eventi
95 In Java8 sono generati oggetti evento di vari tipo quando l'utente
96 esegue qualche azione sull'interfaccia grafica (GUI). Con i metodi "setOn"
97 di un elemento della GUI si puÃ² scegliere il tipo di evento da catturare,
98 l'oggetto evento da considerare, e il codice da eseguire (gestore).
99 CiÃ² si puÃ² esprimere in forma funzionale compatta, detta espressione lamda
100 https://docs.oracle.com/javase/tutorial/java/javaOO/lambdaexpressions.html#lambda-
expressions-in-gui-applications
101 Oppure in forma imperativa (molto piÃ¹ complicata)
102 Le espressioni labda sono tipiche della programmazione funzionale,
103 un approccio dichiarativo. Ci sono linguaggi puramente funzionali

```

```
104 http://en.wikipedia.org/wiki/Functional_programming
105 https://en.wikipedia.org/wiki/Declarative_programming
106 https://en.wikipedia.org/wiki/Imperative_programming
107
108 (02) Flussi
109 Uno stream Ã“ un flusso unidirezionale di informazioni da/verso una sorgente
110 esterna, a cui si accede in modo sequenziale
111 Lessons: Basic I/O
112 https://docs.oracle.com/javase/tutorial/essential/io/index.html
113 I flussi File permettono di connettersi a File. I flussi file di byte
114 vengono gestiti tramite le classi FileInputStream e FileOutputStream.
115 https://docs.oracle.com/javase/tutorial/essential/io/bytestreams.html
116
117 (03) Il 'try-with-resources' Ã“ uno statement 'try' che permette di mettere tra
118 parentesi tande risorse che poi verranno automaticamente chiuse (sono oggetti
119 che devono essere chiusi dopo averli usati, => interfaccia AutoCloseable).
120 Si evitano metodi close e finally. In assenza di eccezioni si usa senza catch
121 versione del codice senza try-with-resources
122 try {
123     FileOutputStream fout = new FileOutputStream("./myfiles/salvadanaio.bin")
124     fout.write(monete);
125     fout.close();
126 } catch...
127 https://docs.oracle.com/javase/tutorial/essential/exceptions/tryResourceClose.html
128
129 (04) I metodi write/read dei flussi file operano sul singolo byte, per cui
130 viene archiviato solo il primo byte dell'intero, fino ad un massimo di 255.
131 Il colore verde della moneta segnala che il salvadanaio e' pieno.
132
133 (05) E' importante segnalare sull'interfaccia grafica eventuali problemi.
134 La console testuale serve solo per sviluppatori e non per l'utente.
135 Il colore rosso segnala che c'e' stato qualche errore a livello di file
136 */
```