

Programmazione

LAB 01

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

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

Presentazione e introduzione ai laboratori

2 of 10

1. Presentazione. Ricevimento subito dopo i laboratori, oppure giovedì dalle 17.30 in poi, su prenotazione via email;
2. Seguire i laboratori è fortemente consigliato per acquisire un buon metodo di lavoro; chi intende dare l'esame ai primi appelli può costruire progressivamente il progetto, chi intende darlo successivamente può svolgere gli esercizi seguendo step-by-step lo svolgimento dei tutorial da parte del docente; lo studente lavoratore può stare al passo con il corso mantenendosi in contatto con un collega che segue.
3. Nei laboratori si farà molta pratica di programmazione Java sperimentando le principali librerie del linguaggio per costruire incrementalmente delle applicazioni con interfaccia grafica, in grado di archiviare dati su file system o su database management system, di distribuire in rete parte della elaborazione o della archiviazione, di interfacciarsi con altre applicazioni tramite XML, di elaborare ingressi ed uscite multimediali. Si apprenderà come consultare la documentazione, come rilevare errori e collaudare il software.

4. Sito web del corso:

<http://elearn.ing.unipi.it/course/view.php?id=430>

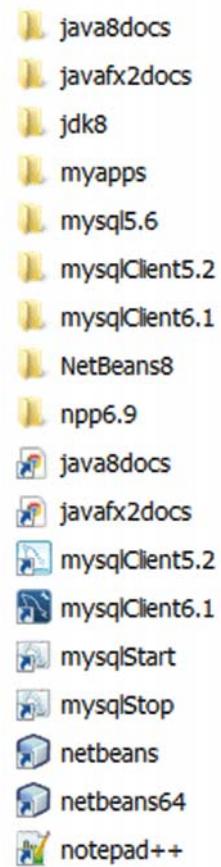
5. Download del pacchetto all-in-one, per sistemi Windows, contenente: Java 8; MySQL; Java API (documentazione di tutte le librerie); Notepad++; NetBeans (editor avanzato per laboratori e progetto).

6. Salvare sul desktop il pacchetto all-in-one e decomprimerlo in C. Apparirà una cartella C:\prg\ con il contenuto visualizzato in figura →

7. Nella cartella *myapps* si metteranno tutti i programmi sviluppati. Per compilarli ed eseguirli da riga di comando si dovrà invocare esplicitamente la versione di java contenuta nella cartella *jdk8*. Es. collocando un file *Test.java* in C:\myapps, questo si compila ed esegue così:

C:\prg\jdk8\bin\javac *.java

C:\prg\jdk8\bin\java Test



Ambiente di sviluppo e applicazione Hello World

8. Aprendo l'editor *NetBeans* si potrà invece gestire anche compilazione ed esecuzione all'interno dell'ambiente integrato.

9. Inizieremo costruendo una semplice applicazione con interfaccia grafica, mostrando la potenza e la espressività delle API Java FX.

10. Dall'ambiente *NetBeans*: File → New Project; Categories: JavaFX; Projects: JavaFX Application; Next.

11. Project Name: Salvadanaio;
Project Location: C:\prg\myapps;
Create Application Class: Salvadanaio;
Finish;

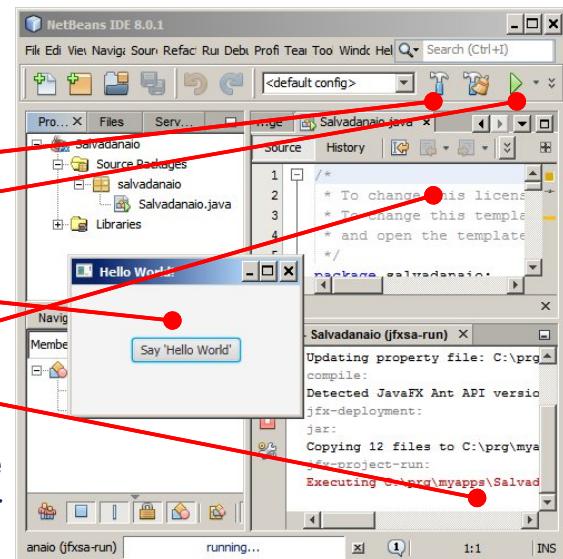
11. Compilare, e poi Eseguire

12. Si apre una semplice finestra

13. Finestra messaggi

14. Finestra del codice

15. Chiudere l'applicazione demo e vuotare il file *Salvadanaio.java* per scrivere nuovo codice.

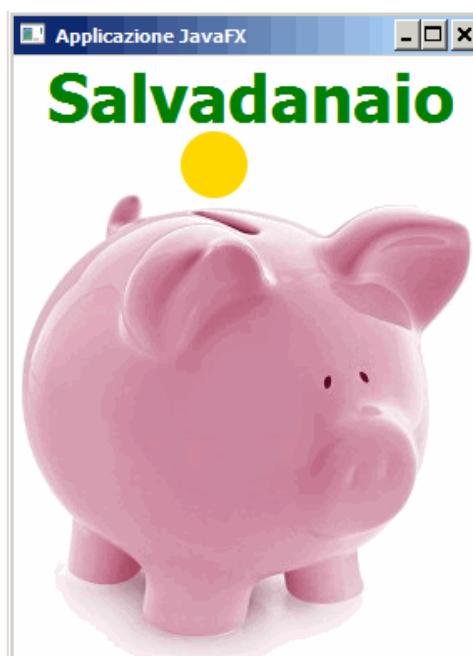
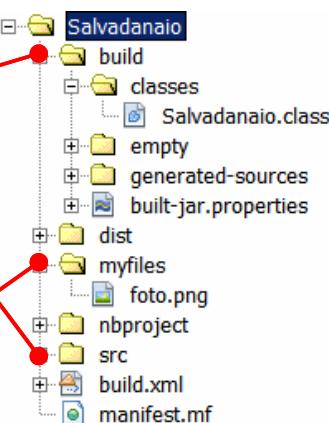


17. Specifiche: costruire la finestra raffigurata in basso, consultando le API JAVA FX (vedere prossima slide). **Alcune classi suggerite:** Application, Label, Group, Scene, Circle (tinyurl.com/javafx-controls).

18. NetBeans colloca i file class in *build* e i file java in *src*

18. Scaricare *foto.png* da <http://www.iet.unipi.it/m.cimino/prg/foto.png> e per esso creare la cartella *myfile*.

19. Si può compilare ed eseguire anche da *classes* digitando:



C:\prg\jdk8\bin\javac *.java
C:\prg\jdk8\bin\java Salvadanaio

tinyurl.com/javafx-intro

Consultazione delle API

20. Consultare spesso le API, per abituarsi alla gerarchia di classi

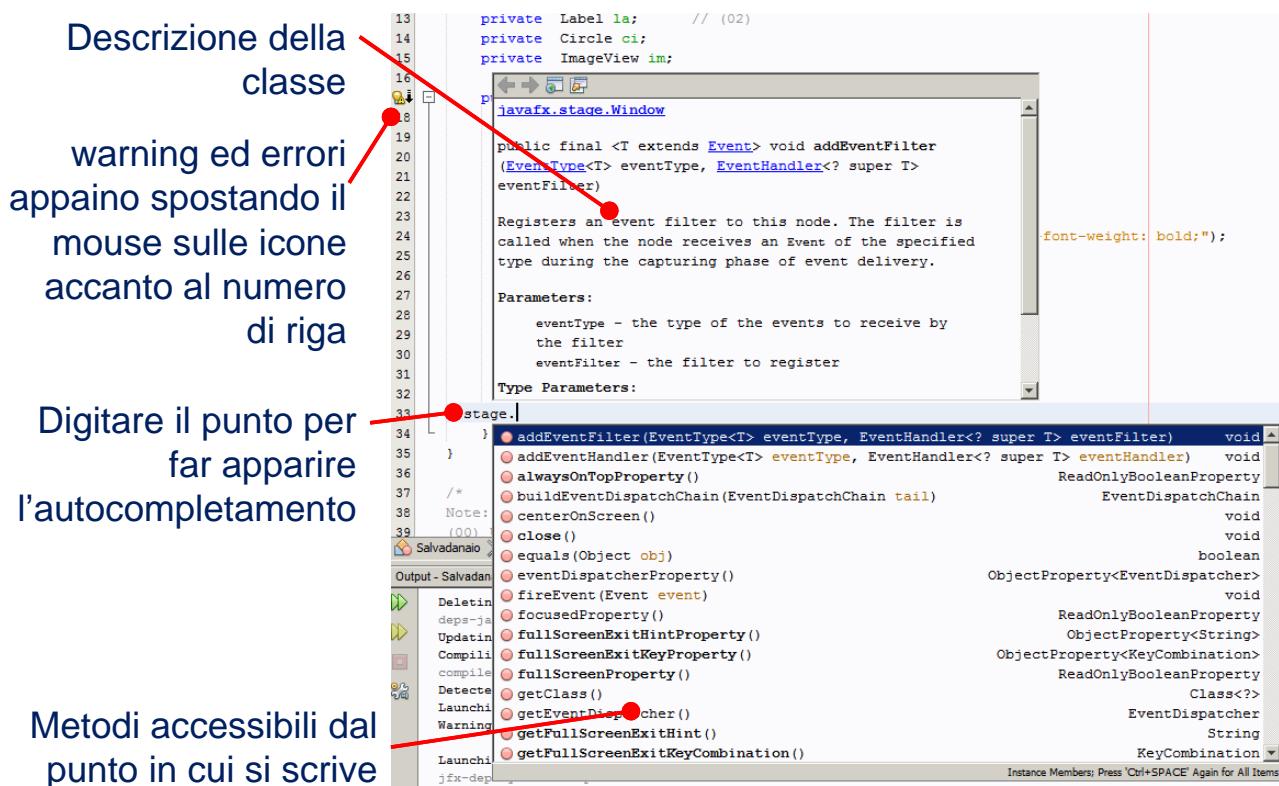
Package di appartenenza della classe

Tutte le classi (in ordine alfabetico)

Descrizione, esempi, elenco di attributi e metodi

The screenshot shows the JavaFX 8 API documentation for the `Application` class. The left sidebar lists packages like `javafx.animation`, `javafx.application`, `javafx.beans`, etc. The main content area shows the `Class Application` page, which includes the inheritance chain from `Object` to `Application`, a list of direct subclasses (`Preloader`), and a detailed description of the `Application` class. Red circles highlight the package list, the class name in the sidebar, and the class name in the main content area.

21. L'autocompletamento si basa sulle API, segnala quelle deprecated.



Metodo di studio per chi è assente a un laboratorio

1. All'esame si adopera il pacchetto 'all-in-one' senza accesso Web. È bene consultare da subito la documentazione 'Java doc' che trovate in esso, e i suggerimenti dell'ambiente NetBeans.
2. Si parte sempre dalle **specifiche**, fornite in forma visuale o testuale, con **alcune classi suggerite**.
3. Dai nomi delle classi suggerite, nell'ordine, si consulta la Java doc, e si cercano in essa esempi base, con i quali scrivere da sé il codice.
4. Notare i suggerimenti dell'ambiente di sviluppo e cercare di risolverli prima di proseguire. Compilare ed eseguire il codice man mano, per vederne l'evoluzione, facendo pratica con l'ambiente di sviluppo.
5. Non inserire tutti gli import trovati negli esempi, ma lasciare che l'ambiente segnali qualcosa e poi trovare il giusto package attraverso la Java doc, confrontandosi solo dopo con la soluzione.
6. Se l'applicazione prevede più componenti dello stesso package (es. più componenti dell'interfaccia, più flussi), vengono suggerite solo alcune classi. Provare inizialmente a cercare da sé le altre classi sfogliando il package nella documentazione.

```

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.control.*;
6 import javafx.scene.paint.*;
7 import javafx.scene.image.*;
8
9 public class Salvadanaio extends Application { //01
10
11     private Label la; //02
12     private Circle ci;
13     private ImageView im;
14
15     public void start(Stage stage) { //03
16
17         la = new Label("Salvadanaio"); //04
18         ci = new Circle(120,65,20, Color.GOLD);
19         im = new ImageView("file:../../myfiles/foto.png"); //05
20         //im = new ImageView("http://www.iet.unipi.it/m.cimino/prg/foto.png");//06
21         //07
22         Group root = new Group(la, ci, im); //08
23         Scene scene = new Scene(root, 280,500); //09
24
25         stage.setTitle("Applicazione Java FX"); //10
26         stage.setScene(scene); //11
27         stage.show(); //12
28     }
29 }
30
31 /*
32 Note:

```

33 (01) Una applicazione Java con interfaccia grafica si realizza estendendo la
34 classe Application. Il metodo start parte automaticamente all'avvio e va
35 ridefinito (non e' obbligatorio un main)
36 <https://docs.oracle.com/javafx/2/api/javafx/application/Application.html>
37 (02) Struttura dell'interfaccia grafica o view
38 Principali componenti grafici di Java FX:
39 https://docs.oracle.com/javafx/2/ui_controls/jfxpub-ui_controls.htm
40 (03) Il metodo start viene chiamato automaticamente all'avvio, consente
41 di inizializzare l'applicazione
42 (04) Istanziazione degli oggetti grafici
43 (05) Il file foto.png ha un percorso relativo alla cartella classes (in Windows,
44 mentre su MACOS/Unix e' relativo alla cartella Salvadanaio).
45 Compilando da riga di comando il percorso e' relativo sempre al punto in
46 cui si generano i class (dentico per i vari sistemi operativi). Es.
47 compilando da src il percorso e' "file:../../myfiles/foto.png".
48 (06) In alternativa si puo' usare un percorso web
49 (07) aggiunta degli elementi nella interfaccia:
50 (08) Un gruppo e' un contenitore di oggetti osservabili dentro la scena
51 (09) Una scena e' il contenitore principale della finestra, il nome discende
52 dal mondo delle classi per lo sviluppo di video game, librerie 3D
53 <https://docs.oracle.com/javase/8/javafx/scene-graph-tutorial/scenegraph.htm>
54 (10) Lo stage (palcoscenico) e' la finestra, il luogo dove si svolge la scena
55 (11) Inserisce la scena nello stage
56 (12) Mostra la finestra
57 */

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 */
```

Programmazione

LAB 03

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

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

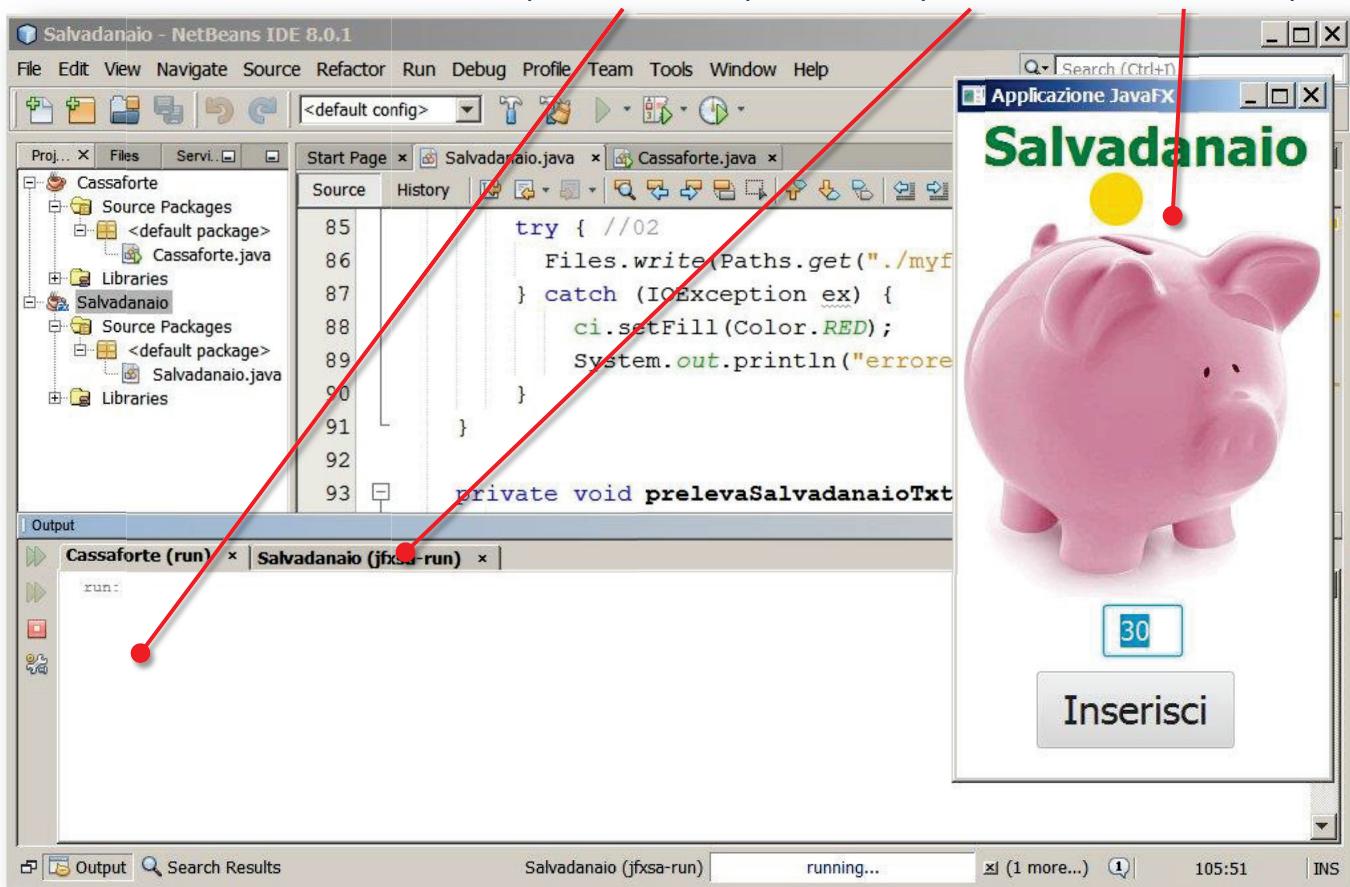
Terza applicazione - serializzazione su file e socket

2 of 11

1. **Specifiche:** consentire la scrittura e lettura della variabile intera, rimuovendo lo stato verde della moneta.
2. La scrittura su file tramite un solo byte è riduttiva. In generale i *flussi file di byte* FileOutputStream e FileInputStream sono scomodi da gestire da soli, perché a basso livello e richiederebbero dei cicli ‘for’ per trasformare il dato in un formato seriale adatto per entrare in un flusso (serializzare).
3. Ci sono i *flussi oggetto*, ObjectInputStream e ObjectOutputStream, che permettono di serializzare automaticamente un oggetto. Il flusso oggetto può poi essere concatenato ad un flusso file per salvare o caricare lo stato di un oggetto da file.
4. Il file binario non è direttamente leggibile da un utente, né esportabile ad altre applicazioni non Java. È possibile a tale scopo adoperare la classe *Files*. Si adoperino dei metodi che permettono di serializzare automaticamente il dato come una sequenza di caratteri, senza implementare dei cicli for.

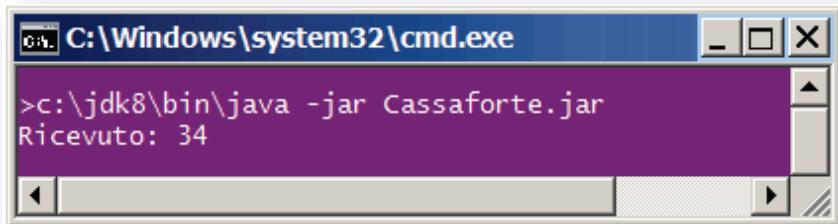
4. Implementare l'invio del dato ad un server remoto, creando una applicazione server *Cassaforte* che riceve un valore intero adoperando la classe *ServerSocket*. Per inviare il dato dal lato applicazione, adoperare la classe *Socket*.
5. L'applicazione *Cassaforte* andrà avviata prima dell'applicazione *Salvadanaio* in modo che sia pronta a ricevere connessioni remote sulla porta 8080. Per la serializzazione dell'intero, connettere i flussi oggetto alla connessione di rete, e gestire la trasmissione/ricezione in maniera identica a quanto visto precedentemente per i flussi file binari.
6. Creare un nuovo progetto di tipo Java Application (senza interfaccia grafica), di nome *Cassaforte*. Prima di cliccare su *Finish* modificare il campo *Create Main Class* a 'Cassaforte' (quindi senza il prefisso del package).
7. Prelevare il file JAR (Java ARchive, un formato zip comodo per distribuire una applicazione composta da molti file class) nella cartella dist. Tale cartella viene generata dopo aver eseguito il build.

Esecuzione locale di Server (con console) e Client (console + interfaccia)



Esecuzione remota di cassaforte (classe server) con console

1. Scaricare sullo host **server** la Java Runtime Environment (jre) e l'applicazione Cassaforte:
<http://www.iet.unipi.it/m.cimino/prg/res/jre8.zip>
<http://www.iet.unipi.it/m.cimino/prg/res/Cassaforte.jar>
2. Supponiamo di estrarre jre8.zip in C, e Cassaforte.jar sul desktop
Da shell posizionarsi sul desktop e digitare
C:\jre8\bin\java -jar "Cassaforte.jar"
3. Sullo host server
connettersi all'indirizzo
www.myipaddress.com
per avere l'indirizzo web.
Supponiamo che tale
indirizzo pubblico appartenga direttamente allo host server.
4. Sullo host client: modifichiamo in Salvadanaio la prima riga del metodo *inviaSalvadanaioBin*, sostituendo “localhost” con l'indirizzo IP dello host server. Eseguendo Salvadanaio e poi terminando si potrà vedere sulla console dell'applicazione Cassaforte il valore della variabile monete.



```
C:\prg\myapps\Cassaforte\src\Cassaforte.java
1 import java.net.*;
2 import java.io.*;
3
4 public class Cassaforte { // (00)
5     public static void main(String[] args) { //01
6         int result = 0;
7         try ( ServerSocket servs = new ServerSocket(8080); // (02)
8              Socket s = servs.accept(); // (03)
9              ObjectInputStream oin = new ObjectInputStream(s.getInputStream());
10            ) { result = (int) oin.readObject();
11            } catch (IOException | ClassNotFoundException e) {e.printStackTrace();}
12            System.out.println("Ricevuto: " + result);
13        }
14    }
15 /*
16 (00)
17 L'applicazione Salvadanaio va lanciata dopo aver lanciato l'applicazione
18 Cassaforte. In basso a netbeans si vedono due console distinte.
19 Oppure aprire una finestra shell, collocarsi nella
20 cartella dove si trova Cassaforte.class e digitare:
21 c:\prg\jkd8\bin\java Cassaforte
22 quindi lanciare l'applicazione client.
23 Alla chiusura del client apparira' un messaggio sulla console di Cassaforte
24 (01)
25 Un socket e' un canale di comunicazione bidirezionale tra due programmi che
26 girano su host (computer) connessi in rete. Uno di essi (server) sta in ascolto
27 su una certa porta logica (un identificativo), l'altro (client) si connette
28 su quella porta e si stabilisce cosi il canale di comunicazione, che rilascia
29 dei flussi oggetto nel quale far passare i dati.
```

```

31     https://docs.oracle.com/javase/8/docs/api/java/net/ServerSocket.html
32     https://docs.oracle.com/javase/8/docs/api/java/net/Socket.html
33     https://docs.oracle.com/javase/tutorial/networking/sockets/index.html
34
35 (02)
36     Crea un TCP server socket, in grado di ricevere richieste di connessione e
37     restituire un socket.
38
39 (03)
40     L'applicazione si ferma ed attende richieste di connessione, questo metodo
41     e' quindi bloccante. Se la connessione va a buon fine viene restituito un socket
42     che poi restituisce un ObjectInputStream. E' anche possibile avere un
43     ObjectOutputStream tramite getOutputStream() dato che il canale e' bidirezionale
44 */

```

C:\prg\myapps\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 import java.nio.file.*;
11 import java.net.*;
12
13 public class Salvadanaio extends Application {
14
15     private int monete = 0;
16
17     private Label la;
18     private Circle ci;
19     private ImageView im;
20
21     private TextField tf;
22     private Button bu;
23
24     public void start(Stage stage) {
25
26         la = new Label("Salvadanaio");
27         ci = new Circle(120, 65, 20, Color.GOLD);
28         im = new ImageView("file:../../myfiles/foto.png");
29         tf = new TextField("0");
30         bu = new Button("Inserisci");
31
32         la.setLayoutX(20);
33         la.setStyle("-fx-font-size: 40px; -fx-text-fill: green; -fx-font-weight: bold;");
34         im.setY(80);
35         tf.setMaxWidth(60);
36         tf.setLayoutX(110); tf.setLayoutY(370);
37         tf.setStyle("-fx-font-size: 20px; -fx-text-fill: blue;");
38         bu.setLayoutX(60); bu.setLayoutY(420);
39         bu.setStyle("-fx-font-size: 30px;");
40
41         bu.setOnAction((ActionEvent ev)->{inserisciMonete();});
42         stage.setOnCloseRequest((WindowEvent we) ->
43 {conservaSalvadanaioBin();conservaSalvadanaioTxt();inviaSalvadanaioBin();});
44
45         Group root = new Group(la, ci, im, tf, bu);
46         Scene scene = new Scene(root, 280, 500);
47         stage.setTitle("Applicazione JavaFX");
48         stage.setScene(scene);
49         stage.show();
50
51         prelevaSalvadanaioTxt();
52         prelevaSalvadanaioBin();
53     }
54
55     private void inserisciMonete() {

```

```

55     String x = tf.getText();
56     if (x.startsWith("+"))
57         monete = monete + Integer.parseInt(x);
58     else
59         monete++;
60     tf.setText(Integer.toString(monete));
61 }
62
63 private void conservaSalvadanaioBin() {
64     try ( FileOutputStream fout = new FileOutputStream("./myfiles/salvadanaio.bin");
65          ObjectOutputStream oout = new ObjectOutputStream(fout); ) { //01
66         oout.writeObject(monete);
67     } catch (IOException ex) {
68         ci.setFill(Color.RED);
69         System.out.println("errore: impossibile conservare il salvadanaio!");
70     }
71 }
72
73 private void prelevaSalvadanaioBin() {
74     try ( FileInputStream fin = new FileInputStream("./myfiles/salvadanaio.bin");
75          ObjectInputStream oin = new ObjectInputStream(fin); ) { //01
76         monete = (int) oin.readObject();
77     } catch (IOException | ClassNotFoundException ex) {
78         ci.setFill(Color.RED);
79         System.out.println("errore: impossibile prelevare il salvadanaio!");
80     }
81     tf.setText(Integer.toString(monete));
82 }
83
84 private void conservaSalvadanaioTxt() {
85     String x = Integer.toString(monete);
86     try { //02
87         Files.write(Paths.get("./myfiles/salvadanaio.txt"), x.getBytes());
88     } catch (IOException ex) {
89         ci.setFill(Color.RED);
90         System.out.println("errore: impossibile conservare il salvadanaio!");
91     }
92 }
93
94 private void prelevaSalvadanaioTxt() {
95     try { //02
96         String x = new String(Files.readAllBytes(Paths.get("./myfiles/salvadanaio.txt")));
97         monete = Integer.parseInt(x);
98         tf.setText(x);
99     } catch (IOException ex) {
100         ci.setFill(Color.RED);
101         System.out.println("errore: impossibile prelevare il salvadanaio!");
102     }
103 }
104
105 private void inviaSalvadanaioBin() { // (4)
106     try ( Socket s = new Socket("localhost", 8080); // (5)
107          ObjectOutputStream oout = new ObjectOutputStream (s.getOutputStream());
108      ) { oout.writeObject(monete);
109      } catch (IOException e) { e.printStackTrace(); }
110     System.out.println("invia stato");
111 }
112 }
113
114 /*
115 Note:
116 (01)
117     I flussi file di byte sono scomodi da gestire. Ci sono i flussi oggetto che
118     permettono di serializzare nel flusso gli oggetti e i tipi primitivi:
119     ObjectInputStream e ObjectOutputStream. Il flusso oggetto puo' essere agganciato
120     ad un flusso file per salvare o caricare lo stato di un oggetto.
121     https://docs.oracle.com/javase/7/docs/api/java/io/ObjectOutputStream.html
122
123 (02)
124     La classe Files consiste esclusivamente di metodi statici operanti su file,
125     https://docs.oracle.com/javase/8/docs/api/java/nio/file/Files.html
126

```

```
127 (04)
128 L'applicazione Salvadanaio va lanciata dopo aver lanciato l'applicazione Cassaforte.
129 Un socket e' un canale di comunicazione bidirezionale tra due programmi che
130 girano su host (computer) connessi in rete. Uno di essi (server) sta in ascolto
131 su una certa porta logica (un identificativo), l'altro (client) si connette
132 su quella porta e si stabilisce così il canale di comunicazione, che rilascia
133 dei flussi oggetto nel quale far passare i dati.
134 https://docs.oracle.com/javase/8/docs/api/java/net/ServerSocket.html
135 https://docs.oracle.com/javase/8/docs/api/java/net/Socket.html
136
137 https://docs.oracle.com/javase/tutorial/networking/sockets/index.html
138
139 (05)
140 Crea un socket e cerca di connetterlo ad una porta di un certo host.
141 Se la connessione va a buon fine viene restituito un ObjectOutputStream.
142 E' anche possibile avere un ObjectInputStream tramite getInputStream()
143 dato che il canale e' bidirezionale.
144
145 Se il server Cassaforte e' avviato su un host diverso, occorre fornire
146 l'indirizzo IP di tale host invece di "localhost".
147 */
148
```

Programmazione

LAB 04

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

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

Quarta applicazione - server ciclico monoprocesso

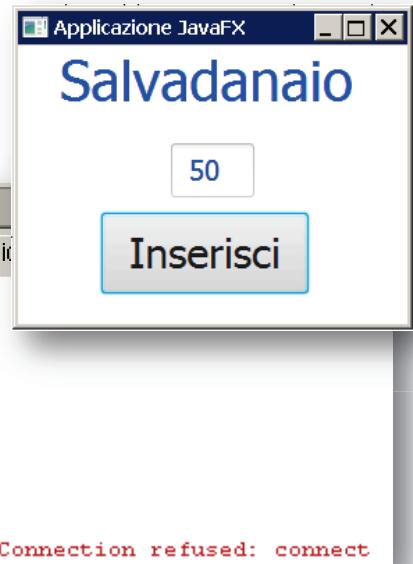
2 of 14

1. Modificare il server in modo che sia ciclico, ossia in grado di accettare più richieste in sequenza senza terminare.
2. Modificare il server in modo che compia un'operazione di qualche secondo ognqualvolta si crea una connessione.
3. Modificare il client in modo che faccia molte richieste al server, ad intervalli di qualche decimo di secondo.
4. Per gestire richieste ravvicinate sulla porta di un server c'è la coda di backlog (arretrato) la cui dimensione indica il numero massimo di richieste TCP che possono essere automaticamente accettate ed accodate senza che la accept() sia ancora stata invocata.
5. In Java esiste il costruttore ServerSocket(int port, **int backlog**) per configurare le dimensioni della coda di backlog. Settare la coda a 7.
6. Catturare l'eccezione generata quando la coda di backlog è piena, e far terminare il client all'occorrenza di tale eccezione.

Idea di massima:

- Il client invia 5000 richieste in rapida successione
- Il server si ferma per 3 secondi ad ogni ricezione

Risultato: dopo 8 invii il client riceve un rifiuto di connessione, in quanto il server ha saturato la coda di backlog.



A quel punto il client termina, ma il server ogni 3 secondi preleva un dato conservato nella coda di backlog.

Quinta applicazione - server ciclico multithread

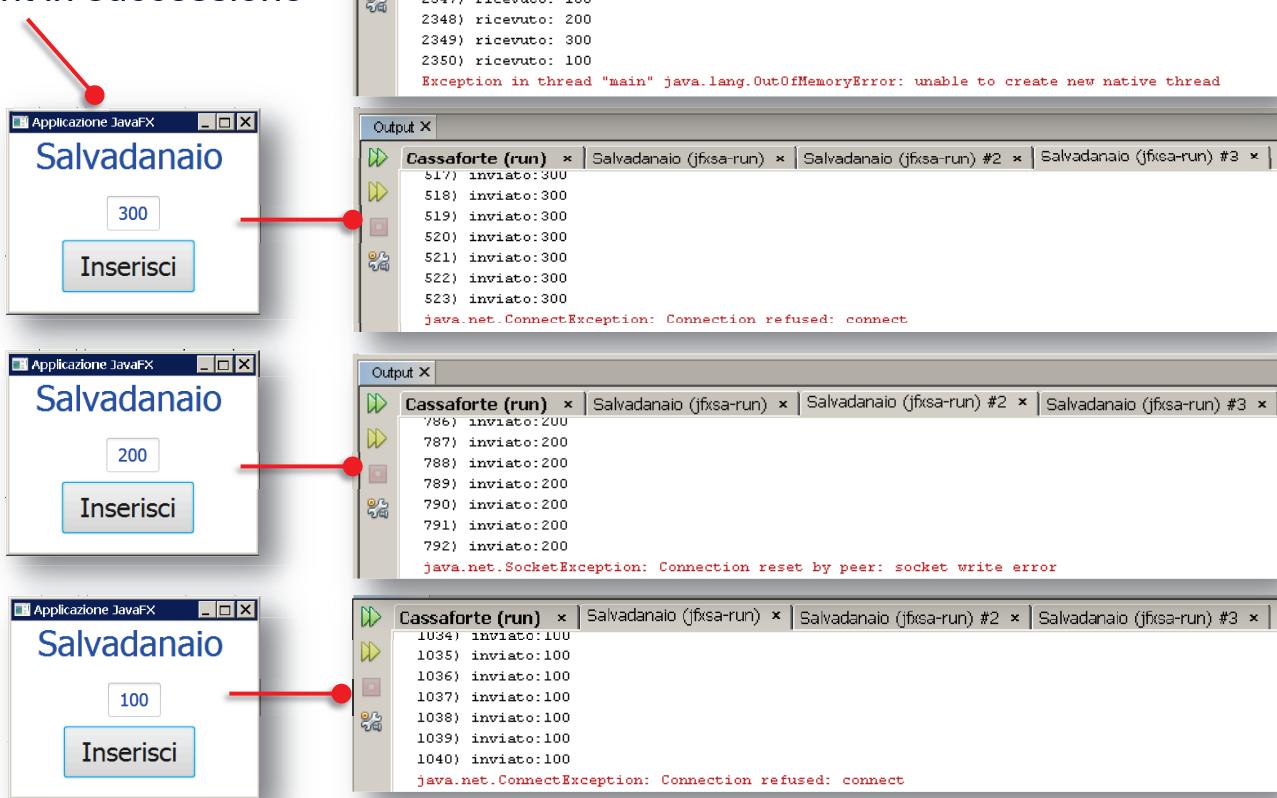
7. Si riprogetti il server con i **thread** per assegnare, ad ogni richiesta di servizio, un flusso sequenziale di controllo indipendente. Verificare sperimentalmente che in tal caso non si crea backlog.
8. In generale, possono esserci altri problemi di scalabilità sul server: es. dopo molte richieste si arriva al numero massimo di thread istanziabili da un processo, se questi non vengono completati in un tempo opportuno. Riprodurre sperimentalmente questo fenomeno.
9. Altro problema di scalabilità: se un thread usa una quantità modesta di memoria RAM per un tempo considerevole, un numero elevato di essi può saturare la memoria RAM disponibile per il processo.

Idea di massima:

- Il client invia 5000 richieste in rapida successione (come prima)
- Il server crea un thread che si ferma per 5 minuti ad ogni ricezione

Risultato: la coda di backlog non si saturerà mai, ma dopo aver creato circa 2350 thread (Windows) il server lancerà una eccezione e si fermerà. A quel punto tutti i thread non potranno più inviare dati e si fermeranno.

Avviare il server,
quindi avviare tre
client in successione



```
C:\prg\myapps\Salvadanaio\src\Salvadanaio.java

1 import javafx.application.*;
2 import javafx.stage.*;
3 import javafx.scene.*;
4 import javafx.scene.control.*;
5 import javafx.event.*;
6 import java.io.*;
7 import java.net.*;
8
9 public class Salvadanaio extends Application {
10
11     private int monete = 0;
12
13     private Label la;
14     private TextField tf;
15     private Button bu;
16
17     public void start(Stage stage) {
18
19         la = new Label("Salvadanaio");
20         tf = new TextField("0");
21         bu = new Button("Inserisci");
22
23         la.setLayoutX(30);
24         la.setStyle("-fx-font-size: 40px; -fx-text-fill: blue;");
25         tf.setMaxWidth(60);
26         tf.setLayoutX(110); tf.setLayoutY(70);
27         tf.setStyle("-fx-font-size: 20px; -fx-text-fill: blue;");
28         bu.setLayoutX(60); bu.setLayoutY(120);
29         bu.setStyle("-fx-font-size: 30px;");
30
31         bu.setOnAction((ActionEvent ev)->{
32             inserisciMonete();
33             inviaSalvadanaioBinCiclico();
```

```

34     });
35
36     Group root = new Group(la, tf, bu);
37     Scene scene = new Scene(root, 280, 200);
38     stage.setTitle("Applicazione JavaFX");
39     stage.setScene(scene);
40     stage.show();
41 }
42
43 private void inserisciMonete() {
44     monete = monete + Integer.parseInt(tf.getText());
45     tf.setText(Integer.toString(monete));
46 }
47
48 private void inviaSalvadanaioBinCiclico() {
49     for (int n=1; n<5000; n++) {
50         try ( Socket s = new Socket("localhost", 8080);
51             ObjectOutputStream oout = new ObjectOutputStream (s.getOutputStream()));
52         ){ oout.writeObject(monete);
53             System.out.println(n + " inviato:" + monete);
54         } catch (Exception e) {e.printStackTrace(); System.exit(1);}
55     }
56 }
57 }
58 }
```

Server mono-processo

C:\prg\myapps\Cassaforte\src\Cassaforte.java

```

1 import java.net.*;
2 import java.io.*;
3
4 public class Cassaforte {
5     public static void main(String[] args) {
6         int n=0;
7         try ( ServerSocket servs = new ServerSocket(8080, 7) ){ //0
```

```

8         while(true) {
9             try ( Socket s = servs.accept();
10                 ObjectInputStream oin = new ObjectInputStream(s.getInputStream()));
11             ){ System.out.println(++n + " ricevuto: " + (int) oin.readObject()); }
12             Thread.sleep(3000); //1
13         }
14     } catch (Exception e) {e.printStackTrace();}
15 }
16 }
17
18 /*
19 (0) coda di backlog lunga 7 (di default e' circa 50 sui sistemi Windows)
20 (1) simulazione di una elaborazione; il server è in grado di accettare più
21 richieste in sequenza fino alla dimensione della coda di backlog
22 */
23
```

Server multi-thread

C:\prg\myapps\Cassaforte\src\Cassaforte.java

```

1 import java.net.*;
2 import java.io.*;
3
4 public class Cassaforte {
5     private static int n = 0;
6
7     public static void main(String[] args) {
8         try ( ServerSocket servs = new ServerSocket(8080, 7) ){ //0
9             while(true) {
10                 Socket s = servs.accept(); //1
11                 Thread t = new Thread() { //1
12                     public void run() { //2
13                         try ( ObjectInputStream oin = new ObjectInputStream(s.getInputStream()) ) {
14                             System.out.println(++n + " ricevuto: " + (int) oin.readObject());
```

```

15             Thread.sleep(300000); //3
16         } catch (Exception e) {e.printStackTrace();}
17     }
18     t.start(); //4
19 }
20 } catch (IOException e) {e.printStackTrace();}
21 }
22 }
23 }
24 */
25 /*
26 Note:
27 (0) Il parametro 7 del costruttore ServerSocket e' la coda di backlog,
28     il default sotto windows e' 50.
29 (1) Dopo ogni richiesta viene lanciato un thread per gestirla. Il Thread viene
30     eseguito come classe anonima, in cui si istanzia un costruttore e si
31     ridefinisce il metodo run. Le classi anonime sono comode quando si istanzia
32     un oggetto di classe gia' esistente con modifiche minime. In tal caso non
33     vale la pena di definire nomi di nuove classi, ma basta estendere quella
34     esistente in modo anonimo.
35     https://docs.oracle.com/javase/tutorial/java/javaOO/anonymousclasses.html
36 (2) Ridefinisco il metodo run() della classe Thread, che contiene il corpo del
37     thread che viene eseguito
38 (3) Crea una attesa di 5 minuti, sufficiente per creare il massimo numero di
39     thread e far scattare l'eccezione
40 (4) il metodo start fa partire effettivamente il nuovo thread chiamando run
41
42 Approfondimenti
43 Java Networking
44 https://docs.oracle.com/javase/tutorial/networking/sockets/index.html
45
46 Coda di backlog
47 https://docs.oracle.com/javase/8/docs/api/java/net/ServerSocket.html#ServerSocket-int-int-
48
49 */

```

Creazione di un JAR da riga di comando

10 of 14

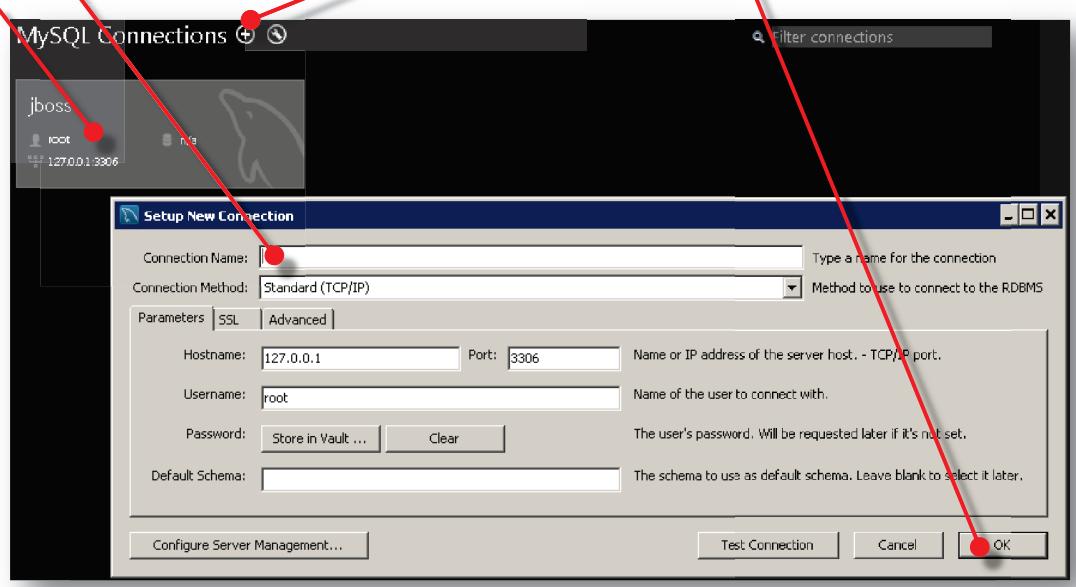
Per creare il jar da riga di comando:

- creare un file di testo mainClass.txt contenente una riga indicante la classe con il metodo main, nel formato: *Main-Class: Salvadanaio ↵*(accapo)
- eseguire i comandi in figura

Basta distribuire solo il jar e la cartella myfiles con l'immagine

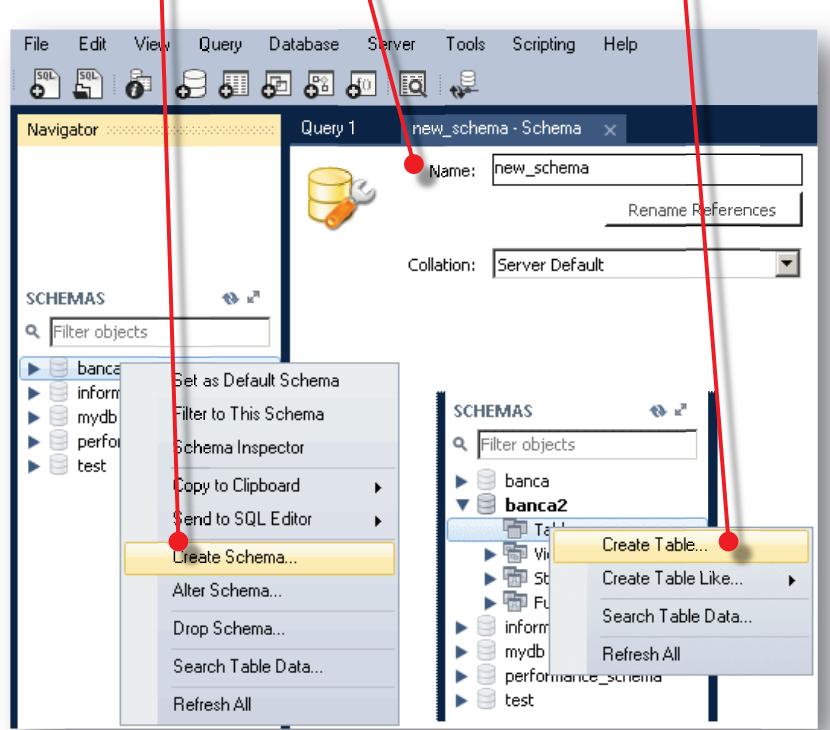
Tramite i package è possibile organizzare i file class in sottocartelle.
<http://docs.oracle.com/javase/tutorial/deployment/jar/basicsindex.html>

1. Archiviare il valore delle monete in un database MySQL.
2. Avvio di MySQL Server: doppio click su *mysql/Start*, dopo qualche secondo apparirà una console con la porta di ascolto (3306).
3. In *MySQL Client 6.1* cliccare sul simbolo '+' di *MySQL Connections*, digitare un nome della connessione e poi premere OK, quindi doppio click sul rettangolo relativo alla connessione.



Sesta applicazione

4. Con il tasto destro su uno dei database elencati nella tab *schemas*, selezionare *Create Schema*, digitare il nome *salvadanaio* e premere *Apply*, *Apply*, *Finish*.
5. Selezionare il nuovo db, tasto destro su *Tables*, selezionare *Create Table*



- Inserire il nome della tabella, e i campi *id* (VARCHAR 45) e *saldo* (INT).

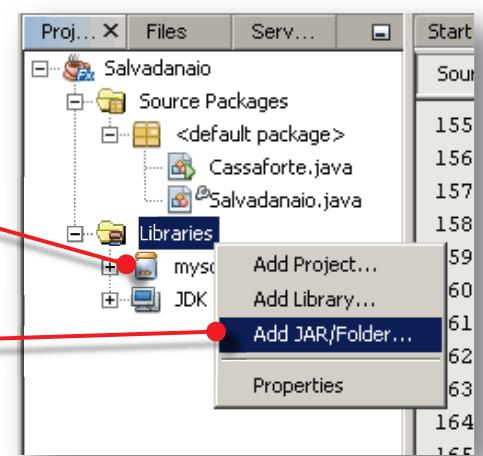
The screenshot shows the MySQL Workbench interface. At the top, there's a 'Query 1' tab and a 'banca2 - Schema' tab. Below that is the 'conticorrenti - Table' tab. The 'Table Name' is set to 'conticorrenti', 'Schema' to 'banca2', 'Collation' to 'Schema Default', and 'Engine' to 'InnoDB'. The 'Comments' field is empty. The table structure is defined with two columns: 'id' (VARCHAR(45)) and 'saldo' (INT). The 'id' column has a primary key (PK) and non-null (NN) constraints. The 'saldo' column has a null constraint (NN). The 'Comments' field is empty.

Below the table definition, the 'SCHEMAS' tree shows 'banca' and 'banca2' schemas, with 'Tables' expanded to show 'conticorrenti'. A context menu is open over the 'conticorrenti' table entry, with the 'Select Rows...' option highlighted.

On the right, a query editor window contains the SQL command: `1 • SELECT * FROM banca2.conticorrenti;`. The results grid shows three rows of data:

	<code>id</code>	<code>saldo</code>
1	bobo@bobo.uk	120
2	tato@tato.uk	100
3	NULL	

- Per esportare uno script SQL in grado di ricostruire il db, selezionare in alto a sinistra, nella scheda MANAGEMENT, la voce *Data Export*, quindi selezionare il db da esportare, selezionare *Export to self-contained file*, cliccare su *Start Export*. Viene salvato un file sql.
- Per ricostruire il database su un altro server, selezionare nella medesima scheda la voce *Data Import/Restore*, quindi *Import from self-contained file*, quindi cliccare su *Start Import*. Quindi cliccare sul tasto di refresh nella scheda SCHEMA.
- Per poter collegare la libreria `mysql-connector-java-5.1.34-bin.jar` presente in `c:\prg\exe` all'interno di NetBeans, tasto destro su *Libraries* della scheda Project, selezionare *Add JAR/Folder*



- In alternativa, da riga di comando digitare:
`C:\prg\jdk8\bin\javac -classpath mysql-connector-java-5.1.34-bin.jar *.java`
`C:\prg\jdk8\bin\java -classpath mysql-connector-java-5.1.34-bin.jar; Test.java`

Programmazione

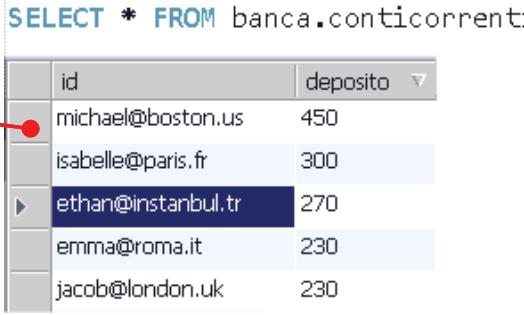
LAB 05

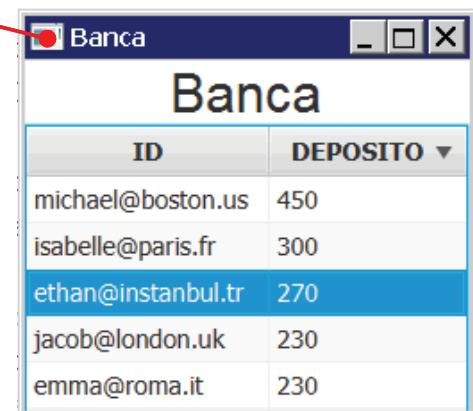
<http://www.iet.unipi.it/m.cimino/prg/>

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

Sesta applicazione - Java-database

2 of 11

- Sviluppare un'applicazione JavaFX in grado di interrogare un database MySQL e di rappresentare i risultati su una tabella visuale come in figura.
- Sul database MySQL esiste una tabella *conticorrenti*, con attributi *id* VARCHAR(45) e *deposito* INT(11). 
- Sull'interfaccia grafica si adoperi la classe *Label* per il titolo, e la classe *TableView* come tabella visuale, posizionati in verticale tramite il gestore di layout *Vbox*. (tinyurl.com/javafx-vbox)
- Per tenere sincronizzata l'interfaccia con i dati presenti nel database, creare una classe “bean” *Cliente*, che incapsula ogni riga del risultato di una query SQL in un oggetto Java.
- In tal modo il programmatore Java ha una interfaccia ad oggetti dei dati relazionali, e grazie alla struttura standard dei *bean* essi sono direttamente osservabili da altre classi.



- Per costruire un *bean*, dichiarare una classe dal nome a piacere e con attributi dal nome a piacere (es. *x*, *y*,...), di tipo Simple Data Objects (es. Integer, String, ...) e con metodi dal nome *getX*, *setX*, *getY*, *setY*,...
- Infine, per definire gli attributi non adoperare direttamente *Integer*, *String*, ... ma delle classi omologhe del package javafx.beans.property, come *SimpleIntegerProperty*, *SimpleStringProperty*...
- In tal modo, i bean saranno automaticamente dotati di osservabilità e di binding con altre classi di JavaFX (come la TableView). (tinyurl.com/javafx-tabview)

Es.

```
public static class Cliente {
    private final SimpleStringProperty id;
    private final SimpleIntegerProperty deposito;

    private Cliente (String i, int d) {
        id = new SimpleStringProperty(i);
        deposito = new SimpleIntegerProperty(d);
    }

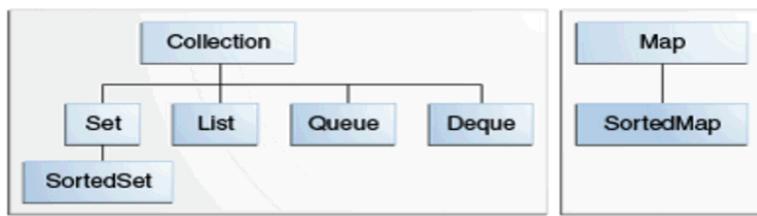
    public String getId() { return id.get(); }
    public int getDeposito() { return deposito.get(); }
}
```

docs.oracle.com/javase/8/javafx/properties-binding-tutorial/binding.htm

docs.oracle.com/javase/tutorial/java/data/index.html

Introduzione a Java Collections a Java Generics

- In Java sono disponibili i principali algoritmi e strutture dati sotto il nome di *Collection*: *Set*, *List*, *Queue*, *Map*,...



docs.oracle.com/javase/tutorial/collections/interfaces/index.html

- Inoltre essi sono definiti come tipi generici, ossia il tipo viene definito all'atto di istanziare la struttura dati, come un parametro speciale tra parentesi angolate. In tal modo il programmatore può definire algoritmi (es. *sort*) in modo generico per un tipo *T*.

docs.oracle.com/javase/7/docs/api/java/util/ArrayList.html

- Es. per dichiarare ed usare una lista senza generics, occorre adoperare il cast, e si ha un controllo del tipo più debole a compile time:

```
List list = new ArrayList(); list.add("hello");
String s = (String) list.get(0);
```

- Invece con i generics non serve cast e c'è maggiore controllo del tipo:

```
List<String> list = new ArrayList<>(); list.add("hello");
String s = list.get(0);
```

docs.oracle.com/javase/tutorial/java/generics/why.html

- JDBC sta per Java DataBase Connectivity e fornisce classi e metodi per gestire in modo programmatico database relazionali, eseguendo query SQL e recuperando i risultati. I package di JDBC 4 sono `java.sql` e `javax.sql`.

docs.oracle.com/javase/tutorial/jdbc/overview/index.html

docs.oracle.com/javase/tutorial/jdbc/basics/index.html

- `DriverManager` è una classe per gestire più JDBC driver. Un JDBC driver è un adattatore lato client che converte richieste da Java ad un protocollo proprietario che può essere compreso dal DBMS. Si tratta di un software scritto dal costruttore del DBMS e fornito come libreria jar: `mysql-connector-java-5.1.34-bin.jar`

`DriverManager` prende una url e restituisce una connection

```
Connection c = DriverManager.getConnection(
    "jdbc:mysql://localhost:3306/banca", "root", "");
```

docs.oracle.com/javase/8/docs/api/java/sql/DriverManager.html

- La classe `Statement` rappresenta uno statement SQL, per eseguire il quale occorre una connessione, fornita dal Driver. L'esecuzione di uno `Statement` può produrre un `ResultSet`, con le tuple restituite dal DBMS. La classe `PreparedStatement` rappresenta query parametriche.

docs.oracle.com/javase/tutorial/jdbc/basics/processingsqlstatements.html

docs.oracle.com/javase/8/docs/api/java/sql/Statement.html

C:\prg\myapps\Salvadanaio\src\Salvadanaio.java

```
1 import javafx.application.*;
2 import javafx.beans.property.*;
3 import javafx.collections.*;
4 import javafx.geometry.*;
5 import javafx.scene.*;
6 import javafx.scene.control.*;
7 import javafx.scene.control.cell.*;
8 import javafx.scene.layout.*;
9 import javafx.scene.text.*;
10 import javafx.stage.*;
11 import java.sql.*;
12
13 public class Salvadanaio extends Application {
14
15     private TableView<Cliente> tb = new TableView<>(); //1
16     private ObservableList<Cliente> ol; //2
17
18
19     public void start(Stage stage) {
20
21         final Label label = new Label("Banca");
22         label.setFont(new Font("Arial", 30));
23
24         TableColumn idCol = new TableColumn("ID"); //3
25         idCol.setCellValueFactory(new PropertyValueFactory<>("id")); //4
26
27         TableColumn depositoCol = new TableColumn("DEPOSITO"); //3
28         depositoCol.setCellValueFactory(new PropertyValueFactory<>("deposito")); //4
29
30         // caricaClientiPredefiniti();
31         caricaClientiDB();
32         // caricaBenestantiDB();
33         // registraClienteDB();
34         // eliminaClienteDB();
35         // aggiornaClienteDB();
36
37         tb.setItems(ol); //5
```

```

38     tb.getColumns().addAll(idCol, depositoCol); //6
39
40     final VBox vbox = new VBox(); //7
41     vbox.getChildren().addAll(label, tb);
42     vbox.setAlignment(Pos.CENTER);
43
44     Scene scene = new Scene(new Group(vbox));
45     stage.setTitle("Banca");
46     stage.setScene(scene);
47     stage.show();
48 }
49
50 public void caricaClientiPredefiniti() {
51     ol = FXCollections.observableArrayList( //8
52         new Cliente("jacob@london.uk", 0),
53         new Cliente("isabelle@paris.fr",0),
54         new Cliente("ethan@istanbul.tr", 0),
55         new Cliente("emma@roma.it", 0),
56         new Cliente("michael@boston.us", 0)
57     );
58 }
59
60 public void caricaClientiDB() {
61     ol = FXCollections.observableArrayList();
62     try ( Connection co = DriverManager.getConnection("jdbc:mysql://localhost:3306/banca", "root", ""));
63     Statement st = co.createStatement(); //10
64     ) {
65     ResultSet rs = st.executeQuery("SELECT * FROM conticorrenti"); //11
66     while (rs.next()) //12
67         ol.add(new Cliente(rs.getString("id"), rs.getInt("deposito")));
68     } catch (SQLException e) {System.out.println(e.getMessage());}
69 }
70
71 public void caricaBenestantiDB() {
72     ol = FXCollections.observableArrayList();
73     try ( Connection co = DriverManager.getConnection("jdbc:mysql://localhost:3306/banca", "root", ""));
74
75     PreparedStatement ps = co.prepareStatement("SELECT id, deposito FROM conticorrenti WHERE
deposito > ?"); //10
76     ) {
77     ps.setInt(1,250);
78     ResultSet rs = ps.executeQuery(); //11
79     while (rs.next()) //12
80         ol.add(new Cliente(rs.getString("id"), rs.getInt("deposito")));
81     } catch (SQLException e) {System.out.println(e.getMessage());}
82
83 public void registraClienteDB() {
84     ol = FXCollections.observableArrayList();
85     try ( Connection co = DriverManager.getConnection("jdbc:mysql://localhost:3306/banca", "root", ""));
86     PreparedStatement ps = co.prepareStatement("INSERT INTO conticorrenti VALUES (?, ?)"); //10
87     ) {
88     ps.setString(1,"socrates@athens.gr"); ps.setInt(2,260);
89     System.out.println("rows affected: " + ps.executeUpdate()); //11
90     caricaClientiDB();
91     } catch (SQLException e) {System.out.println(e.getMessage());}
92 }
93
94 public void eliminaClienteDB() {
95     ol = FXCollections.observableArrayList();
96     try ( Connection co = DriverManager.getConnection("jdbc:mysql://localhost:3306/banca", "root", ""));
97     PreparedStatement ps = co.prepareStatement("DELETE FROM conticorrenti WHERE id = ?"); //10
98     ) {
99     ps.setString(1,"socrates@athens.gr");
100    System.out.println("rows affected:" + ps.executeUpdate()); //11
101    caricaClientiDB();
102    } catch (SQLException e) {System.out.println(e.getMessage());}
103 }
104
105 public void aggiornaClienteDB() {
106     ol = FXCollections.observableArrayList();
107     try ( Connection co = DriverManager.getConnection("jdbc:mysql://localhost:3306/banca", "root", ""));
108     PreparedStatement ps = co.prepareStatement("UPDATE conticorrenti SET deposito = deposito +
WHERE id = ?"); //10
109     ) {

```

```

110     ps.setInt(1,200); ps.setString(2,"michael@boston.us");
111     System.out.println("rows affected:" + ps.executeUpdate()); //11
112     caricaClientiDB();
113 } catch (SQLException e) {System.err.println(e.getMessage());}
114 }
115
116 public static class Cliente { //13
117
118     private final SimpleStringProperty id;
119     private final SimpleIntegerProperty deposito;
120
121     private Cliente(String i, int d) {
122         id = new SimpleStringProperty(i);
123         deposito = new SimpleIntegerProperty(d);
124     }
125
126     public String getId() { return id.get(); }
127     public int getDeposito() { return deposito.get(); }
128 }
129 }
130
131 /*
132 Note:
133 (1) Classe TableView
134     https://docs.oracle.com/javafx/2/ui\_controls/table-view.htm
135
136 (2) la ObservableList è una struttura dati a lista che aggredisce i beans e permette
137     il tracciamento automatico delle modifiche ai dati in essa contenuti
138
139 (3) Una TableView è formata da un insieme di TableColumn, il cui costruttore prende
140     come parametro il nome della colonna
141
142 (4) Per ogni colonna andranno costruite le celle, e riempite con una specifica
143     proprietà dei bean. Il metodo setCellValueFactory specifica una fabbrica di
144     celle per la colonna, implementata usando la classe PropertyValueFactory
145     che prende come parametro il nome della proprietà del bean.
146
147 (5) Dopo aver associato una proprietà del bean ad ogni colonna della tabella visuale,

```

```

148     si può associare la lista osservabile dei bean alla tabella visuale, attraverso
149     il metodo setItems.
150
151 (6) Le colonne precedentemente create vanno aggiunte alla tabella visuale.
152
153 (7) VBox è un gestore di layout che posiziona verticalmente i suoi nodi figlio
154     https://docs.oracle.com/javafx/2/api/javafx/scene/layout/VBox.html
155
156 (8) La classe FXCollections è una classe Utility che consiste in metodi statici
157     che creano delle copie 1:1 di metodi di java.util.Collections.
158     https://docs.oracle.com/javase/8/javafx/api/javafx/collections/FXCollections.html
159
160 (9) Download
161     * MySQL Community Server 5.6.21 (zip version, 32 bit)
162     http://dev.mysql.com/downloads/mysql/
163
164     * MySQL Workbench 5.6
165     http://dev.mysql.com/downloads/workbench/
166
167     * Connettori per MySQL
168     mysql-connector-java-5.1.34.zip
169     http://dev.mysql.com/downloads/file.php?id=454397
170
171 (10) Uno Statement è una interfaccia che rappresenta uno statement SQL.
172     Per eseguire statement SQL occorre una connessione, che viene fornita dal Driver.
173     L'esecuzione di uno Statement può produrre un oggetto ResultSet, che contiene
174     le tuple identificate dall'esecuzione dello statement sul DBMS,
175     https://docs.oracle.com/javase/tutorial/jdbc/basics/processingsqlstatements.html
176
177     La classe Statement rappresenta query SQL non parametriche, mentre la classe
178     PreparedStatement rappresenta statement precompilati, con parametri di ingresso
179     posizionati ma dal valore non definito.
180     http://docs.oracle.com/javase/8/docs/api/java/sql/Statement.html
181     http://docs.oracle.com/javase/tutorial/jdbc/basics/prepared.html
182
183     Mentre uno statement viene compilato dal DBMS ad ogni esecuzione, il prepared
184     statement è compilato solo una volta ed è conveniente nel caso di esecuzioni
185     multiple della stessa query con parametri diversi.

```

```
186
187 (11)
188 Per eseguire uno statement, il metodo executeQuery() restituisce un solo oggetto
189 ResultSet, es. prodotto da una SELECT, mentre executeUpdate restituisce un
190 intero rappresentante il numero di righe coinvolte da una INSERT, DELETE, o UPDATE.
191 https://docs.oracle.com/javase/8/docs/api/java/sql/ResultSet.html
192
193 (12)
194 Si può accedere ad un Resulset attraverso un cursore, ossia un puntatore a una riga.
195 Il puntatore inizialmente è' posizionato prima della prima riga.
196
197 (13) Una classe interna statica fornisce dei valori da considerarsi costanti e
198 condivisi: non si necessita di un oggetto della outer-class per creare un
199 oggetto della inner-class.
200 L'uso dei modificatori private/public ecc. è lo stesso che per i membri:
201 https://docs.oracle.com/javase/tutorial/java/javaOO/innerclasses.html
202 */
203
```

Programmazione

LAB 06

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

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

Settima applicazione: Serializzazione binaria

2 of 16

- Supponiamo di avere il seguente modello dei dati, raffigurato in UML

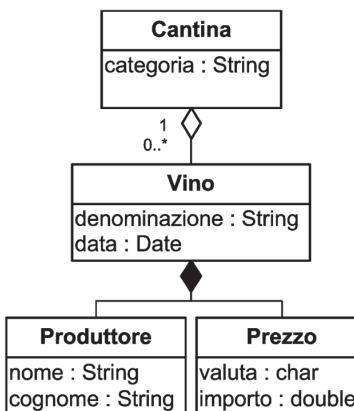


Fig.1 Diagramma delle classi

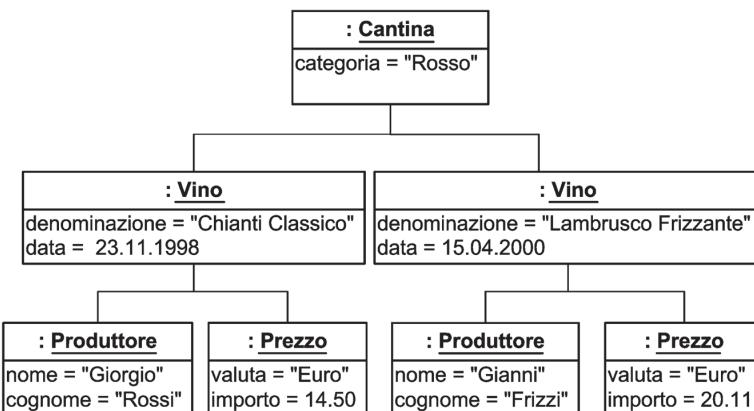


Fig.2 Diagramma degli oggetti (istanze)

- Una *Cantina* contiene diversi tipi di *Vino* di una medesima *categoria*. Ciascun tipo di vino, caratterizzato da una *denominazione* e una *data* di produzione, si compone anche di un *Produttore* e un *Prezzo*.
- Un *Produttore* è caratterizzato da *nome* e *cognome*, mentre il *Prezzo* dalla *valuta* e dall'*importo*.
- Si vuole serializzare tale struttura complessa in formato binario, per essere archiviata su file o inviata su socket.

- Affinchè una classe sia serializzabile, in Java occorre dichiarare che implementa l'interfaccia *Serializable*. E' una interfaccia senza metodi e serve solo come marcatore per contrassegnare le classi che devono potersi serializzare.

```
class Cantina implements Serializable {
    public String categoria;
    public Vino[] vini;
    public Cantina (String c, Vino[] v) { categoria = c; vini = v; }
}
class Vino implements Serializable {...
```

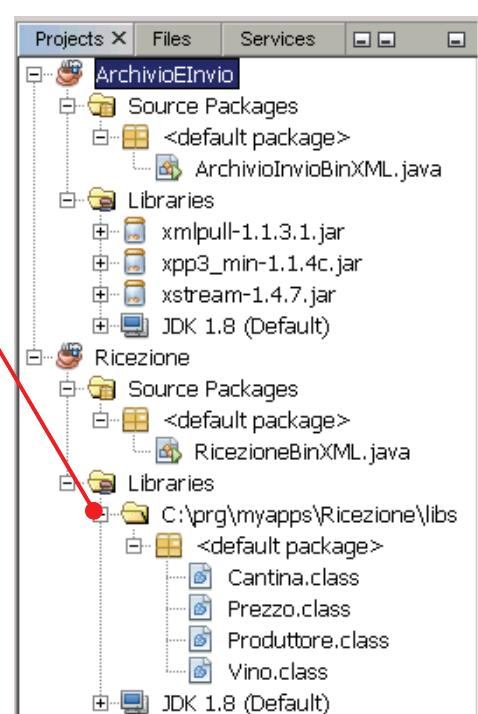
- Il metodo *readObject* / *writeObject* deserializzano / serializzano gli oggetti, percorrendone tutta la struttura gerarchica; se una parte di tali oggetti non è (de-)serializzabile viene lanciata una eccezione *NotSerializable*.
- Quando si serializza un oggetto, si considerano solo le informazioni che caratterizzano l'istanza; quindi nessuna info sui metodi, le costanti. Al momento della deserializzazione sarà ricreata una copia dell'istanza solo se è localmente disponibile il file ".class", altrimenti si lancia l'eccezione *ClassNotFoundException*.

Settima applicazione: Serializzazione binaria

- L'applicazione è fatta da due progetti: *ArchivioEInvio* e *Ricezione*. Dopo aver compilato *ArchivioEInvio* copiare i file *Cantina.class*, *Prezzo.class*, *Produttore.class* e *Vino.class* nella sottocartella *libs* del progetto *Ricezione*, e aggiungere tale cartella come libreria. In tal modo la classe *Ricezione* potrà deserializzare gli oggetti *Cantina*.
- Se compilata/eseguita da riga di comando, la classe *Ricezione* può vedere la cartella *libs* nel seguente modo (file make.bat):


```
C:\prg\jdk8\bin\javac -classpath "C:\prg\myapps\Ricezione\libs" *.java
C:\prg\jdk8\bin\java -classpath ".;C:\prg\myapps\Ricezione\libs" RicezioneBin
```
- Senza classpath non si compila il codice che usa quelle classi, e non si deserializza l'oggetto a tempo di esecuzione. Analogamente rimuovendo l'interfaccia *Serializable* in qualche classe.

[https://en.wikipedia.org/wiki/Classpath_\(Java\)](https://en.wikipedia.org/wiki/Classpath_(Java))





1. Lo sviluppo di un'applicazione Java distribuita comporta l'utilizzo di diverse librerie e l'integrazione con applicazioni esterne (Database Management System, File System, Applicazioni Desktop e Web,...).
2. La tecnologia Java, i cui componenti sono eseguiti sulla Java Virtual Machine, permette di realizzare applicazioni indipendenti dal sistema operativo.

3. Java fornisce anche un valido supporto alla serializzazione degli oggetti, ossia alla loro codifica in un formato lineare che possa passare in uno stream.
4. Tuttavia, questa codifica si basa su un formato binario legato a Java. Ciò significa che occorrono N interfacce per N applicazioni “non Java” e – dualmente – altrettante interfacce per gli altri framework di sviluppo.
5. Il linguaggio XML (Extensible Markup Language), è un metalinguaggio derivato da una famiglia di linguaggi di markup nati per codificare i documenti web in plain text e trasportarli su HTTP.
6. Lo standard XML del W3C permette a tali documenti di essere processati da applicazioni di qualsiasi natura, definendone la struttura mediante XML Schema, e realizzando quindi applicazioni device independent.
7. <http://www.w3.org/XML> <http://www.w3.org/standards/xml/core>

3. La possibilità di definire nuovi elementi (estendibilità) di XML, rispetto ad HTML, lo rende un metalinguaggio. Attraverso XML, ogni organizzazione può definirsi i propri documenti per i propri processi, processabili dagli elaboratori ed intelligibili per i progettisti.
4. Ciò consente di realizzare la integrazione fra le applicazioni: poter scambiarsi protocolli, formati, messaggi...

XStream

5. Fornisce una implementazione alternativa a `java.io.ObjectInputStream` e `java.io.ObjectOutputStream`, permettendo ad un oggetto Java di essere automaticamente serializzato come stringa codificata in XML (metodo `toXML()`) , ed a quest'ultima di essere deserializzata in un oggetto Java (metodo `fromXML()`).
6. Non richiede alcun mapping tra gli oggetti Java ed elementi XML, il cui schema è semplice da interpretare.

7. Tipico utilizzo:
 - a) Persistenza: archiviare la struttura XML su file/db
 - b) Trasporto: trasmettere l'oggetto tra due piattaforme applicative
 - c) Configurazione: inizializzazione degli oggetti da parte dell'utente
 - d) Test di unità: costruire dei test case I/O

Esempio di formato XML del diagramma degli oggetti Cantina

```
<?xml version="1.0" encoding="UTF-8"?>  
<!-- Formato n.1 --&gt;<br/><Cantina>  
  <categoria>Rosso</categoria>  
  <Vino>  
    <denominazione>Chianti Classico</denominazione>  
    <data giorno = "23" mese = "11" anno = "1998"/>  
    <Produttore formato = "nome cognome">Giorgio Rossi</Produttore>  
    <Prezzo importo = "14.50 Euro"/>  
  </Vino>  
  <Vino>  
    <denominazione>Lambrusco Frizzante</denominazione>  
    <data giorno = "15" mese = "04" anno = "2000"/>  
    <Produttore formato = "nome cognome">Gianni Frizzi</Produttore>  
    <Prezzo importo = "20.11 Euro"/>  
  </Vino>  
</Cantina>
```

<? Processing instruction?>
direttiva per l'
applicazione

formato é un
attributo dell'
elemento
Produttore

Altro esempio di formato XML del diagramma degli oggetti Cantina

```
<?xml version="1.0" encoding="UTF-8"?>
<!!-- cantina.xml -->
<Cantina categoria = "Rosso">
  <Vino>
    <denominazione>Chianti Classico</denominazione>
    <data>1998-11-23</data>
    <Produttore>
      <nome>Giorgio</nome>
      <cognome>Rossi</cognome>
    </Produttore>
    <Prezzo valuta = "EUR">
      <importo>14.50</importo>
    </Prezzo>
  </Vino>
  <Vino>
    <denominazione>Lambrusco Frizzante</denominazione>
    <data>2000-04-15</data>
    <Produttore>
      <nome>Gianni</nome>
      <cognome>Frizzi</cognome>
    </Produttore>
    <Prezzo valuta = "EUR">
      <importo>20.11</importo>
    </Prezzo>
  </Vino>
</Cantina>
```

Quale è il formato migliore?

Quello che rispetta le **regole di buona progettazione XML**

Regole di buona progettazione XML

Nella progettazione di una struttura dati XML, ad ogni nuovo dato occorre scegliere se si tratterà come attributo o elemento

- I. È un nuovo **elemento** il dato che è **strutturato su linee multiple**, o **cambia spesso**, o può assumere **una moltitudine di valori**
- II. È un nuovo **attributo** il dato che è **stringa o numero semplice**, e non **cambia frequentemente**, e può assumere **un set limitato di valori**.

Con “cambiare frequentemente” si intende che un valore di default settato sulla applicazione che genera i dati va bene per la maggioranza dei documenti generati in un certo contesto. Esempi: unità di misura (“kg”, “m”, “°C”), valute monetarie (“eur”, “usd”), formati (“gg/mm/aaaa”), categorie statiche come il fuso orario (“UTC+1”).

- III. Eccezione: più dati che individualmente potrebbero modellarsi come attributi, si devono però modellare come elementi se occorre specificare la presenza in un certo ordine, o sono alternativi, oppure possono apparire più di una volta.

```
C:\prg\myapps\Ricezione\src\RicezioneBinXML.java
1 import java.net.*;
2 import java.io.*;
3
4 public class RicezioneBinXML {
5     public static void main(String[] args) {
6         Cantina c = null;
7         try ( ServerSocket servs = new ServerSocket(8080);
8              Socket so = servs.accept();
9              ObjectInputStream oin = new ObjectInputStream(so.getInputStream());
10             Socket sd = servs.accept();
11             DataInputStream din = new DataInputStream(sd.getInputStream()) //1
12         ) { c = (Cantina) oin.readObject();
13             System.out.println(c.categoria + ":" + c.vini.length);
14             System.out.println(din.readUTF());
15         } catch (Exception e) {e.printStackTrace();}
16     }
17 }
18
19 /*
20 Note:
21 (1) Un DataInputStream legge tipi di dato primitivi Java da un input stream
22      http://docs.oracle.com/javase/7/docs/api/java/io/DataInputStream.html
23 */
```

```
C:\prg\myapps\ArchivioEInvio\src\ArchivioInvioBinXML.java
1 import com.thoughtworks.xstream.*; // (00)
2 import com.thoughtworks.xstream.converters.basic.*;
3 import java.io.*;
4 import java.net.*;
5 import java.util.*;
6 import java.nio.file.*;

7 import java.text.*;
8
9 class Produttore implements Serializable { // (01)
10    public String nome, cognome;
11    public Produttore(String n, String c) { nome = n; cognome = c; }
12 }
13
14 class Prezzo implements Serializable {
15    public String valuta;
16    public double importo;
17    public Prezzo(String v, double i) { valuta = v; importo = i; }
18 }
19
20 class Vino implements Serializable {
21    public String denominazione;
22    public Date data;
23    public Produttore Produttore;
24    public Prezzo Prezzo;
25    public Vino(String dn, Date dt, Produttore pd, Prezzo pz) {
26        denominazione = dn; data = dt; Produttore = pd; Prezzo = pz; }
27 }
28
29 class Cantina implements Serializable {
30    public String categoria;
31    public Vino[] vini;
32    public Cantina (String c, Vino[] v) { categoria = c; vini = v; }
33 }
34
35 public class ArchivioInvioBinXML {
36
37    public static void serializzaBin(Cantina c) {
38        try ( ObjectOutputStream outf =
39                  new ObjectOutputStream( new FileOutputStream("cantina.bin") ) // (02)
40                  ObjectOutputStream outts =
41                  new ObjectOutputStream( (new Socket("localhost",8080) ).getOutputStream())
42        ) { outf.writeObject(c); outts.writeObject(c); } catch (Exception e) {}
```

```

43     try ( ObjectInputStream oinf =
44         new ObjectInputStream( new FileInputStream("cantina.bin") ) ; // (03)
45     ) { c = (Cantina)oinf.readObject(); } catch (Exception e) {e.printStackTrace();}
46     System.out.println(c.categoria + ": " + c.vini.length);
47 }
48
49 public static void serializzaXML(Cantina c) {
50     XStream xs = new XStream(); // (03)
51     xs.useAttributeFor(Cantina.class, "categoria"); // (04)
52     xs.useAttributeFor(Prezzo.class, "valuta");
53     xs.registerConverter(new DateConverter("yyyy-MM-dd", null));
54     String x = xs.toXML(c); // (06)
55     try { Files.write(Paths.get("cantina.txt"), x.getBytes());
56         x = new String(Files.readAllBytes(Paths.get("cantina.txt")));
57     } catch (Exception e) {}
58     c = (Cantina)xs.fromXML(x); // (07)
59     System.out.println(c.categoria + ": " + c.vini.length);
60     try ( DataOutputStream dout = // (08)
61         new DataOutputStream( new Socket("localhost",8080) .getOutputStream())
62     ) { dout.writeUTF(x);} catch (Exception e) {e.printStackTrace();}
63 }
64
65 public static void main(String[] args) throws Exception {
66     Cantina c =
67         new Cantina( "Rosso", new Vino[]{
68             new Vino( "Chianti Classico",
69                 new SimpleDateFormat("yyyy-MM-dd").parse("1998-11-23"), // (09)
70                 new Produttore("Giorgio","Rossi"),
71                 new Prezzo("EUR",14.50)
72             ),
73             new Vino( "Lambrusco Frizzante",
74                 new SimpleDateFormat("yyyy-MM-dd").parse("2000-04-15"), // (09)
75                 new Produttore("Gianni","Frizzi"),
76                 new Prezzo("EUR",20.11)
77         })
78     );
79
80     ArchivioInvioBinXML.serializzaBin(c);
81     ArchivioInvioBinXML.serializzaXML(c);
82 }
83
84 /*
85 (00)
86 Librerie Xstream per (de-)serializzare in formato XML, adoperate nel metodo
87 serializzaXML
88 http://xstream.codehaus.org/
89 Si tratta di librerie esterne. In Java ci sono delle librerie interne per XML,
90 (package javax.xml), ma sono piu' complesse da adoperare
91 http://docs.oracle.com/javase/tutorial/jaxp/
92
93 (01)
94 Affinché una classe sia serializzabile, in Java occorre dichiarare che implementa
95 l'interfaccia Serializable. È una interfaccia senza metodi e serve solo come marcatore
96 per contrassegnare le classi che devono potersi serializzare.
97 https://docs.oracle.com/javase/8/docs/api/java/io/Serializable.html
98 Per semplicità, definire classi serializable con attributi pubblici e senza metodi.
99
100 (02)
101 I metodi readObject e writeObject (de-)serializzano gli oggetti, percorrendone
102 tutta la struttura; se una parte di tali oggetti non è (de-)serializzabile
103 viene lanciata una eccezione NotSerializable. Quando si serializza un oggetto,
104 si serializza solo le informazioni che caratterizzano l'istanza, quindi nessuna
105 info sui metodi, le costanti. Al momento della deserializzazione sarà ricreata
106 una copia dell'istanza solo se è disponibile il file ".class", altrimenti si
107 lancia l'eccezione ClassNotFoundException
108
109 (03)
110 L'oggetto xstream rappresenta un flusso in cui possono viaggiare oggetti XML,
111 ed offre semplici metodi per (de-)serializzare. Non è richiesto che la classe
112 sia serializzabile per essere processata da xstream
113 http://xstream.codehaus.org/javadoc/index.html
114

```

```

115 (04)
116 In generale XStream serializza i dati in forma di elementi (tag) annidati,
117 tramite il metodo useAttributeFor si puo' dire di serializzare come attributo
118
119 (05)
120 Tramite il metodo alias si puo' dire di non serializzare una data classe,
121 ma di inserire una semplice label
122
123 (06)
124 Serializza da Java in formato XML.
125
126 (07)
127 Deserializza da formato XML a Java
128
129 (08) dataOutputStream e' uno stream per primitive Java data types
130 http://docs.oracle.com/javase/7/docs/api/java/io/DataOutputStream.html
131
132 (09)
133 SimpleDateFormat e' una classe per formattare (data -> testo) o fare il
134 parsing (testo -> data) le date e le ore
135 https://docs.oracle.com/javase/8/docs/api/java/text/SimpleDateFormat.html
136
137 ---
138 Output applicativo
139 <Cantina categoria="Rosso">
140   <vini>
141     <Vino>
142       <denominazione>Chianti Classico</denominazione>
143       <data>1998-11-23</data>
144       <Produttore>
145         <nome>Giorgio</nome>
146         <cognome>Rossi</cognome>
147       </Produttore>
148       <Prezzo valuta="EUR">
149         <importo>14.5</importo>
150       </Prezzo>
151     </Vino>
152   <Vino>
153     <denominazione>Lambrusco Frizzante</denominazione>
154     <data>2000-04-15</data>
155     <Produttore>
156       <nome>Gianni</nome>
157       <cognome>Frizzi</cognome>
158     </Produttore>
159     <Prezzo valuta="EUR">
160       <importo>20.11</importo>
161     </Prezzo>
162   </Vino>
163 </vini>
164 </Cantina>
165 */
166

```

Programmazione

LAB 07

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

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

Validazione XML: introduzione

2 of 19

1. Nello scambio di documenti XML, è necessario avere struttura e tipi di dato ben definiti: i documenti delle istanze devono essere conformi ad un unico documento della classe (detto **schema**).
2. Esempi di schemi: editoria web (XHTML 1.0 strict, transitional, frameset), grafica vettoriale (SVG³), musica (MusicXML⁴), chimica, (CML⁵), Interfacce Grafiche (FXML⁶), interrogazioni di documenti semi-strutturati (XQuery⁷), e molti altri settori della conoscenza.
3. I linguaggi schema permettono di creare nuove classi XML: sono DTD (Document Type Definition) e XSD (XML Schema Definition). Essi permettono di definire tag e attributi che possono essere presenti, in che numero, ordine, ecc. per una data classe di documenti XML da adoperare per specifiche applicazioni.

Riferimenti:

- (1) <http://it.wikipedia.org/wiki/XML>
- (2) http://it.wikipedia.org/wiki/XML_Schema
- (3) http://en.wikipedia.org/wiki/Scalable_Vector_Graphics
- (4) <http://en.wikipedia.org/wiki/MusicXML>
- (5) http://en.wikipedia.org/wiki/Chemical_Markup_Language
- (6) http://docs.oracle.com/javafx/2/fxml_get_started/jfxpub-fxml_get_started.htm
- (7) http://www.w3schools.com/xsl/xquery_intro.asp

Nota: il corso tratta solo Java-XML relativamente all'I/O File e Socket. Non viene trattato Java-XML per GUI (FXML) e DB (XQuery e database semi strutturati)

1. Lo XML schema è a sua volta una grammatica XML: si tratta in sostanza di ‘tag universali per creare nuovi tag particolari’.
2. Tali tag universali appartengono al seguente ‘package’, o più correttamente ‘xml namespace’ (xml ns):
`xmlns:xs="http://www.w3.org/2001/XMLSchema"`
3. Nota: questo indirizzo non è una URL ma uno *Universal Resource Identifier (URI)*, una stringa che identifica simbolicamente uno spazio per tag nel web, e permette ai tag di non essere confusi (collisione) con altri tag omonimi.
4. Il meccanismo è l’analogo dei package per i nomi delle classi Java.
5. Per evitare URI duplicate vengono adoperati i domini web (come per i package java), ma a differenza di una URL (Uniform Resource Locator), digitando una URI nel browser può non esserci alcuna risorsa.

Validazione XML: esercizi

- a) Strutturare il testo seguente come un documento XML. Quindi definire uno schema XML.

Nota per Pietro da Sara. Promemoria: chiamami questo weekend

nota.xml

```
1 <?xml version="1.0"?>
2 <nota xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
3   xsi:noNamespaceSchemaLocation="nota.xsd">
4   <per>Pietro</per>
5   <da>Sara</da>
6   <titolo>Promemoria</titolo>
7   <corpo>Chiamami questo weekend</corpo>
8 </nota>
```

nota.xsd

```
1 <?xml version="1.0"?>
2 <xss:schema xmlns:xss="http://www.w3.org/2001/XMLSchema">
3 <xss:element name="nota">
4   <xss:complexType>
5     <xss:sequence>
6       <xss:element name="per" type="xs:string"/>
7       <xss:element name="da" type="xs:string"/>
8       <xss:element name="titolo" type="xs:string"/>
9       <xss:element name="corpo" type="xs:string"/>
10      </xss:sequence>
11    </xss:complexType>
12  </xss:element>
13 </xss:schema>
```

Si può **validare** un documento xml su un xsd

Primo metodo di validazione:
tramite browser,
caricando i file
xml e xsd su
www.xmlvalidation.com

Secondo metodo di validazione:
programmatico,
tramite API Java del
package javax.xml

The screenshot shows a web page titled "Validate an XML file". At the top, there is a note: "Read here how to validate your XML files (including referenced DTDs) online with just a few mouse clicks." Below this, a text area says "Please copy your XML document in here:" followed by an XML snippet. A "Choose File" button and a "No file chosen" message are present. A note below states: "The validation check is performed against any XML schema or DTD declared inside the XML document. If neither an XML schema nor a DTD is declared, only a syntax check is performed. To validate the XML document against an external XML schema, click below." A checked checkbox labeled "Validate against external XML schema" is shown. A "validate" button is at the bottom. A note at the bottom says: "The following files have been uploaded so far: XML document: Ø XML schema: Ø Click on any file name if you want to edit the file."

Validazione XML: esercizi

- b) Scambiare gli elementi <da> e <per> nel file XML e validare.
Sostituire *sequence* con *all* nel file XSD e validare nuovamente.
- c) Strutturare il testo seguente come un documento XML. Quindi definire uno schema XML.
Nella mia famiglia ci sono tre persone, con i seguenti nomi (e soprannomi): Luigi Rossi ("Gigi"), Pietro Rossi ("Pietrino", "Salsedine" o "Ciuffo"), e Sara Rossi. Non sono ammessi più di tre soprannomi.

famiglia.xml

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <persone xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
3           xsi:noNamespaceSchemaLocation="famiglia.xsd">
4   <persona>
5     <nome>Luigi Rossi</nome>
6     <soprannome>Gigi</soprannome>
7   </persona>
8   <persona>
9     <nome>Pietro Rossi</nome>
10    <soprannome>Pietrino</soprannome>
11    <soprannome>Salsedine</soprannome>
12    <soprannome>Ciuffo</soprannome>
13   </persona>
14   <persona>
15     <nome>Sara Verdi</nome>
16   </persona>
17 </persone>
```

famiglia.xsd

```

1 <?xml version="1.0" encoding="UTF-8"?>
2 <xsschema xmlns:xs="http://www.w3.org/2001/XMLSchema">
3 <xselement name="persone">
4   <xsccomplexType>
5     <xsssequence>
6       <xselement name="persona" maxOccurs="unbounded">
7         <xsccomplexType>
8           <xsssequence>
9             <xselement name="nome" type="xs:string"/>
10            <xselement name="soprannome" type="xs:string"
11              minOccurs="0" maxOccurs="3"/>
12           </xsssequence>
13         </xsccomplexType>
14       </xselement>
15     </xsssequence>
16   </xsccomplexType>
17 </xselement>
18 </xsschema>
```

- d) Aggiungere l'informazione (obbligatoria) sul formato del nome della persona.

famiglia-bis.xml

```

<persona>
  <nome formato="nome cognome">Luigi Rossi</nome>
  <soprannome>Gigi</soprannome>
</persona>
```

famiglia-bis.xsd

```

1 <?xml version="1.0" encoding="UTF-8"?>
2 <xsschema xmlns:xs="http://www.w3.org/2001/XMLSchema">
3 <xselement name="persone">
4   <xsccomplexType>
5     <xsssequence>
6       <xselement name="persona" maxOccurs="unbounded">
7         <xsccomplexType>
8           <xsssequence>
9             <xselement name="nome">
10            <xsccomplexType>
11              <xssimpleContent>
12                <xsextension base="xs:string">
13                  <xssattribute name="formato"
14                    type="xs:string" use="required"/>
15                </xsextension>
16              </xssimpleContent>
17            </xsccomplexType>
18          </xselement>
19        <xselement name="soprannome" type="xs:string"
20          minOccurs="0" maxOccurs="3"/>
21      </xsssequence>
22    </xsccomplexType>
23  </xselement>
24 </xsccomplexType>
25 </xselement>
26 </xsschema>
```

- ✓ <xs:schema xmlns:xs=...> definisce uno XML namespace, un prefisso del tag per evitare conflitti di nomi quando si collegano documenti XML di diverse applicazioni (è come i Java package). Spesso si usano le URI come spazio di nomi per avere un prefisso globalmente unico basato sul DNS. L'elemento <xs:schema> è la radice di ogni schema XML.
- ✓ **URI** (Universal Resource Identifier): è una stringa di caratteri usata per identificare una risorsa web. Non necessariamente punta ad un reale documento o pagina web, come fa la URL fornendo una locazione nella rete.
- ✓ <xs:element name=...> definisce il nome di un elemento. Il numero di occorrenze permesse all'interno dell'elemento contenitore è 1 di default.
- ✓ <xs:attribute name=...> definisce il nome di un attributo. L'attributo è opzionale di default. Per renderlo obbligatorio adoperare l'attributo "use".
- ✓ **type="xs:string", "xs:date", "xs:time", "xs:decimal", "xs:integer", "xs:boolean"**: per dichiarare il tipo di un contenuto semplice (tipo semplice).
- ✓ <xs:complexType><xs:sequence>, <xs:all>, <xs:choice>: definisce un elemento complesso fatto da una sequenza di elementi ordinata, non ordinata, alterantivi. Un elemento semplice contiene un tipo semplice, un elemento complesso contiene altri elementi e/o attributi
- ✓ Rif: www.w3.org/TR/xmlschema11-1/ www.w3schools.com/xml/schema_intro.asp

Ottava applicazione: Validazione XML

10 of 19

Schema del documento

Cantina: un elemento

"cantina" è un tipo complesso, fatto da un elemento complesso "vini" e un attributo "categoria".
Un elemento "vini" è fatto da una sequenza di elementi "vino".

Un elemento "vino" è fatto da una sequenza di elementi

"denominazione",
"data",...ecc. ecc.

Cardinalità

Riferimento

Tipo

Tipo complesso

Sequenza di

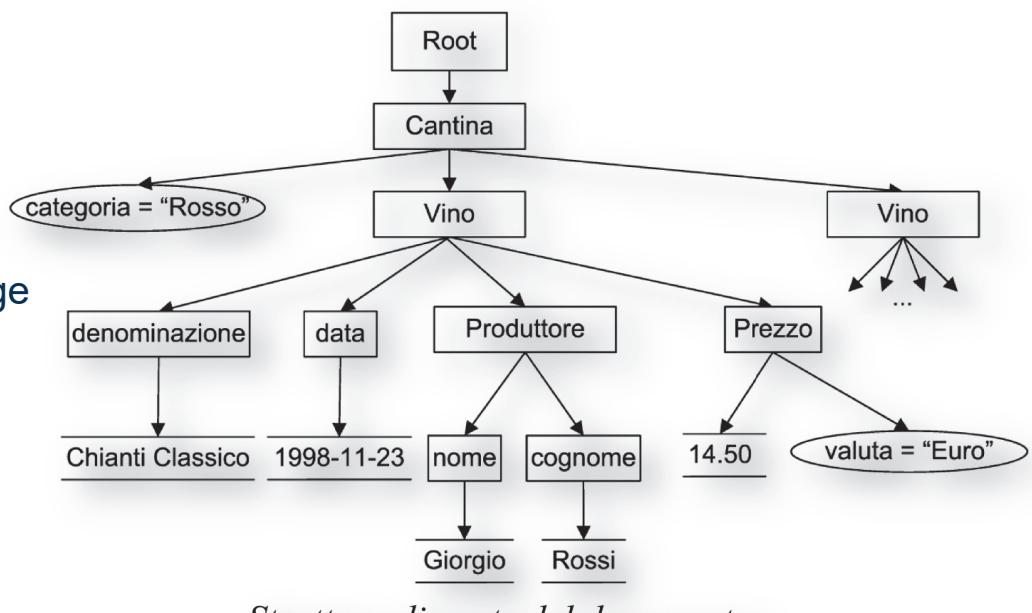
Elemento

Attributo

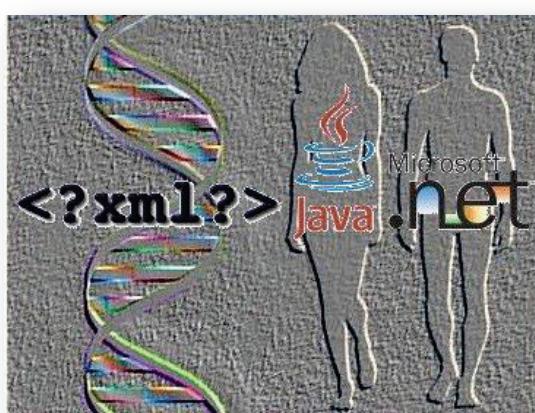
```
<?xml version="1.0" encoding="UTF-8"?>
<!-- cantina.xsd -->
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" >
  <xs:element name="Cantina">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="vini">
          <xs:complexType>
            <xs:sequence>
              <xs:element ref="Vino" minOccurs="0" maxOccurs="unbounded"/>
            </xs:sequence>
          </xs:complexType>
        </xs:element>
        <xs:attribute name="categoria" type="xs:string" use="required"/>
      </xs:complexType>
    </xs:element>
    <xs:element name="Vino">
      <xs:complexType>
        <xs:sequence>
          <xs:element name="denominazione" type="xs:string"/>
          <xs:element name="data" type="xs:date"/>
          <xs:element ref="Produttore"/>
          <xs:element ref="Prezzo"/>
        </xs:sequence>
      </xs:complexType>
    </xs:element>
    <xs:element name="Produttore">
      <xs:complexType>
        <xs:sequence>
          <xs:element name="nome" type="xs:string"/>
          <xs:element name="cognome" type="xs:string"/>
        </xs:sequence>
      </xs:complexType>
    </xs:element>
    <xs:element name="Prezzo">
      <xs:complexType>
        <xs:sequence>
          <xs:element name="importo" type="xs:decimal"/>
        </xs:sequence>
        <xs:attribute name="valuta" type="xs:string" use="required"/>
      </xs:complexType>
    </xs:element>
  </xs:schema>
```

1. Invece di costruirsi oggetti Java specifici per ogni elemento, si possono usare gli oggetti generici del DOM per manipolare un documento XML.
2. Document Object Model (DOM) è un modello ad oggetti, definito dal W3C, che consente di manipolare un documento XML costruendo una struttura ad albero i cui nodi sono gli elementi, mediante una Interfaccia di Programmazione delle Applicazioni (API) uguale per tutti i linguaggi di programmazione.

3. In Java il package org.w3c.dom, fornisce le API generiche per il DOM.



Serializzazione XML tra .NET e Java



1. Esempio di comunicazione tra un client .NET con un server Java, tramite XML over HTTP. Il client è una .NET console application che invia tramite HTTP-POST un flusso di dati XML al server HTTP Java.
2. Per il lato .NET, se necessario, installare: Windows Installer 3.1, Microsoft .NET Framework 2.0 Service Pack 1 (Runtime Environment), .NET Framework 2.0 (Software Development Kit).

Serializzazione XML tra .NET e Java

4 .NET è l'architettura Microsoft per lo sviluppo di applicazioni. Il codice sorgente (.cs) scritto in C# ("C sharp") viene compilato in un linguaggio intermedio (IL, corrisponde al bytecode di Java) ed eseguito da una opportuna macchina virtuale. L'applicazione ha estensione .exe come una applicazione per macchina fisica (in Java è .class).

Approfondimenti:

- <http://msdn.microsoft.com/it-it/library/z1zx9t92.aspx>
- http://en.wikipedia.org/wiki/.NET_Framework

5. Adoperiamo una versione base, la 2.0, scaricabile da:

Windows Installer 3.1 (solo se viene richiesto), .NET Framework 2.0 (RE)

http://www.iet.unipi.it/m.cimino/tiga/dotnet2.0_a.zip

.NET Framework 2.0 SDK (zip, 360MB)

http://www.iet.unipi.it/m.cimino/tiga/dotnet2.0_b.zip

6. Importare il progetto su NetBeans. Per compilare ed eseguire cliccare su compilaNET.bat ed eseguiNET.bat della cartella src. Il file c# è ClientNet.cs

Per compilare da riga di comando:

C:\WINDOWS\Microsoft.NET\Framework\v2.0.50727\csc /t:exe ClientNet.cs

```
C:\prg\myapps\Validazione\src\ValidazioneXML.java
 1 import javax.xml.*;
 2 import javax.xml.parsers.*;
 3 import org.w3c.dom.*;
 4 import java.io.*;
 5 import org.xml.sax.*;
 6 import javax.xml.validation.*;
 7 import javax.xml.transform.stream.*;
 8 import javax.xml.transform.dom.*;
 9 import com.sun.net.httpserver.*;
10 import java.net.*;
11
12 public class ValidazioneXML {
13
14     public static void valida() {
15         try {
16             DocumentBuilder db = DocumentBuilderFactory.newInstance().newDocumentBuilder(); // (01)
17             SchemaFactory sf = SchemaFactory.newInstance(XMLConstants.W3C_XML_SCHEMA_NS_URI); // (02)
18             Document d = db.parse(new File("cantina.xml")); // (03)
19             Schema s = sf.newSchema(new StreamSource(new File("cantina.xsd"))); // (04)
20             s.newValidator().validate(new DOMSource(d)); // (05)
21         } catch (Exception e) {
22             if (e instanceof SAXException)
23                 System.out.println("Errore di validazione: " + e.getMessage());
24             else
25                 System.out.println(e.getMessage());
26         }
27     }
28
29     public static void riceviXMLsuHTTP() {
30         try {
31             HttpServer s = HttpServer.create(new InetSocketAddress(80), 0); // (06)
32             s.createContext("/www", (HttpExchange t) -> { // (07)
33                 String string = "";
34                 if (t.getRequestMethod().equals("GET")) string = RISPOSTA_GET; // (08)
35                 else {
36                     BufferedReader br = new BufferedReader(new InputStreamReader(t.getRequestBody(), "utf-8")); // (09)
37                     string = br.readLine().substring(10); // (10)
38                     System.out.println("ricevo: " + string);
39                 }
40                 DocumentBuilder db = DocumentBuilderFactory.newInstance().newDocumentBuilder();
```

```

41     InputSource is = new InputSource(); // (11)
42     is.setCharacterStream(new StringReader(string)); // (11)
43     Document d = db.parse(is); // (11)
44     string = d.getElementsByTagName("importo").item(0).getTextContent(); // (12)
45     } catch (Exception e) {System.out.println(e);}
46   }
47   System.out.println("invio: " + string);
48   t.sendResponseHeaders(200, string.length()); // (13)
49   try (OutputStream os = t.getResponseBody())
50   { os.write(string.getBytes()); }
51 };
52 s.start(); // (14)
53 }catch (Exception e) {e.printStackTrace();}
54 }

55 public static void main(String[] args) {
56   valida();
57   riceviXMLsuHTTP();
58 }
59 }

60 private static final String RISPOSTA_GET =
61   "<!DOCTYPE html><html><head><meta charset=\"utf-8\"><title>Risposta a richieste GET</title></head>" +
62   "<body><p>Errore: devi invocarmi con metodo POST</p></body></html>";
63 }
64 }
65 /*
66 Note
67 (01)
68 Per validare un documento statico come cantina.xml su cantina.xsd si può usare
69 http://www.xmlvalidation.com/
70 Ma se il documento è modificabile da un agente esterno (es. e' un file di configurazione
71 Oppure un file proveniente da socket) si deve validare dinamicamente, adoperando
72 le API Java.xml.validation
73 https://docs.oracle.com/javase/8/docs/api/javax/xml/validation/package-summary.html
74
75 DocumentBuilderFactory istanzia dei parser che producono oggetti DOM da documenti XML
76 (02)
77 SchemaFactory legge rappresentazioni esterne di schemi, per la validazione
78 (03)
79 Document e' l'oggetto documento DOM vero e proprio caricato dal file XML
80 (04)

82 Schema e' l'oggetto schema vero e proprio caricato da file XSD
83 (05)
84 Crea un oggetto validatore che valida un documento xml sullo schema caricato
85 In caso di errore vi validazione viene generata una SAXException.
86 Il parser di validazione e' di tipo SAX, ossia ad eventi.
87 In generale ci sono due tipi di parser: quello DOM (che costruisce in memoria
88 una struttura ad albero del documento) e quello SAX (che non crea strutture
89 interne ma lo scorre e quando legge qualcosa di non corretto genera un evento.
90 Il parser SAX sono migliori per query semplici su grossi documenti,
91 es. cercare una entry in una rubrica XML, mentre il parser DOM e' migliore
92 per query complesse su documenti contenuti.
93 http://howtodoinjava.com/2014/07/30/dom-vs-sax-parser-in-java/
94 (06)
95 Crea un httpserver embeded nell'applicazione, per ricevere XML via http.
96 https://docs.oracle.com/javase/8/docs/jre/api/net/httpserver/spec/com/sun/net/httpserver/package-summary.html
97 Per http server scalabili si usa invece apache tomcat, un servlet container,
98 e le Java servlet.
99 (07)
100 Quando arriva una richiesta http per un certo percorso interno del server
101 (context) viene generato un evento, che tramite la consueta lamda expression
102 viene gestito da opportuno codice.
103 Quindi ciascun percorso interno rappresenta la location di un servizio distinto,
104 L'oggetto HTTPExchange incapsula le informazioni relative allo scambio
105 richiesta http e risposta http da generare.
106 https://docs.oracle.com/javase/8/docs/api/javax/xml/ws/spi/http/HttpExchange.html
107 (08)
108 in caso di accesso con metodo GET, restituisce una pagina HTML di risposta
109 Provare con un browser ad accedere su http://localhost/www
110 (09)
111 restituisce un inputstream per leggere il corpo della richiesta http.
112 (10)
113 il corpo della richiesta contiene un unico parametro di nome 'messaggio',
114 quindi inizia con 'messaggio='... where al posto dei puntini c'e' la parte XML.
115 poiche' la stringa 'messaggio=' è lunga 10 caratteri, viene eliminata tramite la substring
116 (11)
117 L'oggetto Document viene costruito a partire da uno stream di caratteri.
118 In generale InputSource permette di incapsulare in un unico oggetto tutte le
119 informazioni che servono al parser su come leggere l'input XML (es. la
120 codifica di caratteri)
121 https://docs.oracle.com/javase/8/docs/api/org/xml/sax/InputSource.html
122 (12)

```

```

123 Uso del DOM per prelevare delle informazioni
124 Package org.w3c.dom
125 https://docs.oracle.com/javase/7/docs/api/org/w3c/dom/package-summary.html
126 Interfaccia Node
127 https://docs.oracle.com/javase/7/docs/api/org/w3c/dom/Node.html
128 (13)
129 fornisce un outpustream per inviare il corpo della richiesta
130 il primo parametro è un codice risposta
131 (14)
132 Avvia il server HTTP */

```

```

using System; // (01)
using System.Xml;
using System.Net;
using System.Text;
using System.IO;

namespace Invia {
    class ClientNET {
        static void Main(string[] args) {
            XmlDocument d = new XmlDocument(); // (03)
            d.Load("cantina.xml"); // (03)
            Console.WriteLine(d.GetElementsByTagName("importo")[0].InnerText); // (04)
            string mess = d.DocumentElement.FirstChild.ChildNodes[0].OuterXml;
            Console.WriteLine("invio: " + mess);
            Console.WriteLine("ricevo: " + ClientNET.HttpPost("http://localhost/www", mess));
        }

        static string HttpPost(string uri, string messaggio) { // (07)
            string parameters = "messaggio=" + messaggio;
            byte[] bytes = Encoding.ASCII.GetBytes(parameters); // (08)
            WebRequest webRequest = WebRequest.Create(uri);
            webRequest.ContentType = "application/x-www-form-urlencoded";
            webRequest.Method = "POST";
            webRequest.ContentLength = bytes.Length;
            Stream os = null;
            String response = null;
            StreamReader sr = null;
           WebResponse webResponse = null;
            try { os = webRequest.GetRequestStream();
                os.Write(bytes, 0, bytes.Length);
                os.Flush();
                if (os != null) os.Close();
            } catch (Exception ex) {
                Console.WriteLine(ex.ToString());
                if (os != null) os.Close();
                return null;
            }
            try { webResponse = webRequest.GetResponse();
                if (webResponse == null) { return null; }
                sr = new StreamReader(webResponse.GetResponseStream());
                response = sr.ReadToEnd().Trim();
                if (sr != null) { sr.Close(); webResponse.Close(); }
                return response;
            } catch (Exception ex) {
                Console.WriteLine(ex.ToString());
                if (sr != null) { sr.Close(); webResponse.Close(); }
                return null;
            }
        }
    } //HttpPost
} //Program

```

```
}

/*
Note
(00)
.NET e' l'architettura Microsoft per lo sviluppo di applicazioni.
Il codice sorgente scritto in C# viene compilato in un linguaggio intermedio (IL) ed eseguito
da una opportuna macchina virtuale.
L'applicazione ha estensione .exe, come una qualsiasi applicazione per macchina fisica.
http://msdn.microsoft.com/it-it/library/z1zx9t92.aspx
http://msdn.microsoft.com/en-us/library/ff361664\(v=vs.110\).aspx

.NET 2.0 Framework Class Library (equivalenti alle API di Java)
http://msdn.microsoft.com/en-us/library/ms229335\(v=vs.80\).aspx
http://msdn.microsoft.com/en-us/library/aa139623.aspx

Adoperiamo una versione base, la 2.0, scaricabile da:
Windows Installer 3.1 (solo se viene richiesto), .NET Framework 2.0 (RE)
http://www.ipt.unipi.it/m.cimino/tiga/dotnet2.0\_a.zip
.NET Framework 2.0 SDK (zip, 360MB)
http://www.ipt.unipi.it/m.cimino/tiga/dotnet2.0\_b.zip
(01)
using in C# equivale ad import per Java
(03)
carica un file xml come un oggetto DOM
System.xml.XmlDocument
http://msdn.microsoft.com/en-us/library/ms229335\(v=vs.80\).aspx
http://msdn.microsoft.com/en-US/library/azsy1tw2\(v=vs.80\).aspx?cs-save-lang=1&cs-lang=csharp#code-snippet-1
(04)
Console.WriteLine() in C# equivale a System.out.println() in Java
GetElementsByTagName fornisce una lista di tutti i nodi con un certo nome di tag
http://msdn.microsoft.com/en-US/library/system.xml.xmlelement\_methods\(v=vs.80\).aspx
GetElementsByTagName("importo")[1] prende il secondo elemento
InnerText prende il valore interno (20.11)
(05)
d.DocumentElement restituisce il nodo XmlElement radice (<Cantina>)
http://msdn.microsoft.com/en-US/library/system.xml.xmldocument.documentElement\(v=vs.80\).aspx

http://msdn.microsoft.com/en-US/library/system.xml.xmlelement\_members\(v=vs.80\).aspx
FirstChild restituisce il primo figlio (<vini>)
ChildNodes[0] restituisce il primo nodo figlio (<Vino>)
InnerText restituisce tutta la parte XML interna a quel nodo (<denominazione>...)
(06)
OuterXml restituisce tutta la parte XML inclusa il nodo stesso (<Cantina>)
(07)
Fa la richiesta httpPost (l'implementazione non e' importante, essendo di livello sistema)
(08)
parameters: name1=value1&name2=value2 */
```

Programmazione

LAB 08

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

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

Idee di progetto

2 of 28

- prendere spunto dalle app disponibili per smartphone realmente adoperate
- semplici moduli con calcoli in linea (es. libretto esami)
- semplici giochi da tavolo senza intelligenza (es. battaglia navale)
- messaggistica istantanea semplificata (ispirandosi a whatsapp, skype)
- raccolta foto (album fotografico), dati utili (rubrica), lista delle cose da fare
- rassegna stampa
- prenotazione posti a teatro, cinema, in aereo, treno, ecc.
- spesa online
- dizionario, vocabolario, rimario
- test, quiz (anche visivi es. far apparire rapidamente una figura o una sagoma)
- scadenziario
- giochi spaziali (space invaders), giochi di logica (es. cruciverba)
- ...

Si raccomanda di partire con una unica finestra dalla struttura statica, focalizzandosi sulle funzionalità applicative (es. nessun login/logout), svolgere tutte le fasi del progetto fino in fondo, e poi eventualmente ingrandire a seconda delle ore rimaste.

Coprire tutto il Java I/O studiato, non esagerare su JavaFX anche se è accattivante, rispetto agli altri Java I/O; dopo l'esame si può eventualmente approfondire come tesi da 3CFU (75 ore).

Come adoperare il registro in modo *online*

- Quando inizi a lavorare al progetto:
 - a) inserisci la matricola
 - b) seleziona *inizio*
 - c) clicca su *submit*
(data-ora corrente è inviata in automatico)
- Quando finisci di lavorare al progetto:
 - a) inserisci la matricola
 - b) seleziona *fine*
 - c) seleziona il tipo di attività
 - c) descrivi l'attività
 - d) clicca su *submit*
(data-ora corrente è inviata in automatico)

ISO 9001: DAILY ACTIVITIES REPORTING

Programmazione

MATRICOLA *

INIZIO/FINE ATTIVITA *

- INIZIO
 FINE

TIPO DI ATTIVITA

- 1) ANALISI
 2) PROGETTO
 3) PROTOTIPAZIONE
 4) SVILUPPO-INTEGRAZIONE
 5) COLLAUDO
 DOCUMENTAZIONE

DESCRIZIONE ATTIVITA

DATA-ORA DI RIFERIMENTO

SubmitCome adoperare il registro in modo *offline*

- Lo studente offline prende nota su carta delle informazioni di cui sopra a inizio e fine attività. Viene anche annotata la data-ora corrente (che nel form sarà la data-ora di riferimento)
- Non appena è connesso, inserisce i dati suddetti. (la data-ora corrente viene inviata in automatico con la data ora di riferimento)

Come descrivere bene il registro?

- a) Usare il modo offline solo con specifiche motivazioni, messe nella descrizione;
- b) Non inserire offline ore artificiali (es. inizio 10.00 fine 15.00), in generale prendere nota sul momento delle informazioni da inserire;
- c) Non inserire record multipli ma immettere ciascun record appena possibile;
- d) La descrizione deve essere dettagliata adoperando tutti i 500 caratteri;
- e) In caso di errori di inserimento inviare subito una email con copia e incolla della riga in versione attuale e della medesima nella versione desiderata;
- f) Il registro è un diario professionale: descrivere sinteticamente task svolti (task di approfondimento, problemi e tentativi per risolverli) e i progressi/risultati ottenuti;
- g) Snellire le frasi per non sprecare caratteri in giri di parole; evidenziare come si è speso il tempo; non scrivere secondo schemi automatici: es. non inserire sistematicamente i nomi di tutte le classi realizzate e i loro metodi (informazione che tra l'altro si trova nel diagramma di classe) ma citare le classi dove si è speso più tempo e introdurle in termini di responsabilità e non di metodi;
- h) Evitare frasi universali, ossia applicabili a qualsiasi progetto (es. "completato il documento di analisi"), o prevedibili, ossia che si potevano dire anche prima di iniziare quelle ore (es. "iniziato il caso d'uso").

Quando si corregge il registro?

- a) Al superamento delle prime 15 ore (ossia alla conclusione del record che supera le 15 ore), si invia una email per verificare la qualità del registro. Il registro viene commentato (via skype o di persona) ed inviato in formato excel, per essere poi rieditato (nei campi descrizione e tipo di attivita) come se si ritornasse indietro nel tempo e si ripartisse. Questo esperimento mentale di ritornare indietro nel tempo e rimuovere gli errori è importante per non sbagliare sucessivamente, in quanto non sarà più possibile correggere.
- b) Il registro è pubblicamente accessibile in tutti i campi tranne la descrizione, quest'ultima può essere conservata autonomamente dallo studente all'atto dell'invio; per semplificare la gestione lato docente non si possono inviare copie del registro al di là delle prime 15 ore;

Come si fa a sapere se il registro va bene dopo la correzione?

- a) Se vi sono dubbi su record specifici del registro si possono espressamente fare domande, senza aspettarsi che debba essere il docente a ricontrollare il file ricevuto. Se non sono chiare le regole da applicare allora si fanno domande sulle regole prima di fare le modifiche.
- b) L'errore iniziale piu' frequente è di scrivere frasi generiche come "*Revisione diagramma UML*"; scrivere invece frasi sull'argomento concreto del progetto e informazioni che è possibile sapere solo alla fine delle ore.

Come formulare domande che puntano alla formazione e non alla dipendenza?

- a) L'attività di svolgimento del progetto, se condotta sempre con domande generiche come "va bene così?" porta alla progressiva dipendenza dello studente dal docente, e quindi alla non maturazione per il lavoro aziendale produttivo.
- b) Comunque lo studente non deve sentirsi giudicato prima dell'esame, e può fare domande di qualsiasi tipo, poiché la valutazione del progetto avviene solo nella versione finale sottomessa all'esame. Per spendere tempo in modo formativo e consapevole dei progressi, le domande dovrebbero essere sempre più specifiche.
- c) Dopo la fase di analisi, lo studente può avere dubbi sull'applicazione di alcuni specifici criteri di qualità (es. architettura leggibile e modulare) a specifiche parti del progetto (es. ad una classe). È responsabilità del docente spiegare come applicare quel criterio a quelle parti del progetto. Però l'applicazione del medesimo criterio a tutte le parti del progetto è responsabilità dello studente.

Come si fa a proporre un progetto da realizzare in 45 ore?

- Partire in piccolo, con una sola finstra dalla struttura statica, con una sola tabella, un grafico, e dei campi di input/output
- Allo scadere delle 30 ore contattare il docente per verificare il rischio di finire prima o dopo le 45 ore, e quindi per concordare una possibile strategia per ridurre o ampliare il progetto al momento opportuno

In che formato vanno inviati i documenti nelle fasi intermedie e finale?

Inviare un unico documento in formato pdf che cresce incrementalmente. Per creare immagini di alta qualità dentro pdf installare la stampante pdf995 (www.pdf995.com). In MS Word (1) selezionare i menu File > Opzioni > Avanzate > Dimensioni e qualità immagine > Non comprimere immagini nel file (2) includere l'immagine png nel testo word e stampare il documento su stampante virtuale pdf995, settando una risoluzione di 1200 dpi

Criteri di qualità del progetto

CRITERIO	DESCRIZIONE
(1) REGISTRO CHIARO E IN FASE	<ul style="list-style-type: none"> - Il registro è adoperato prevalentemente online, e le attività si susseguono nell'ordine: - (1) analisi: si descrive cosa fa l'applicazione con degli scenari tipici e ispirati ai casi d'uso - (2) progetto: si disegna un diagramma delle classi del sistema, tramite le conoscenze derivanti dai laboratori e/o prototipazioni - (3) prototipazione: si provano frammenti di codice dei tutorial per ampliare le conoscenze di alcuni componenti, prendendo estratti ed eseguendoli in locale, ma non si modificano - (4) sviluppo-integrazione: si realizzano e integrano le componenti dell'applicazione - (5) collaudo: si testa l'applicazione sugli scenari stabiliti nell'analisi - (*) documentazione può essere abbinata alle fasi precedenti - condurre la fase (2) fino al massimo possibile con le conoscenze dei laboratori, poi passare alla fase (3) per formarsi su nuovi componenti, e infine completare la fase (2); è ammesso un solo passaggio indietro da 3→2 all'interno di un macro ciclo; - si può ripartire con un nuovo macrociclo, da (5) a (1) per la 2^a o 3^a iterazione, per ampliare/correggere il progetto, motivandolo nel registro, ma non si può tornare indietro tra le attività a meno del caso 3→2.
(2) ARCHI- TETTURA LEGGIBILE E MODULARE	<ul style="list-style-type: none"> - classi distinte tra front-end, middleware e back-end - classi e attributi nominati con nomi, metodi nominati con verbi (da dizionario italiano) - i nomi e i verbi corrispondono alle funzionalità realizzate - indentare il codice, commentare il codice come note in fondo al file - metodi non più lunghi di una videata - separare lunghe strutture dati (es. stringhe, numeri) da codice
(3) COPERTURA PROGRAMM A DEL CORSO	<p>l'applicazione contiene le seguenti 5 componenti di I/O:</p> <ul style="list-style-type: none"> (i) interfaccia grafica in JavaFX; (ii) file di configurazione locale in XML/XSD; (iii) cache locale degli input su file binario; (iv) base di dati in JDBC/MySQL; (v) server/file di log remoto in XML/XSD

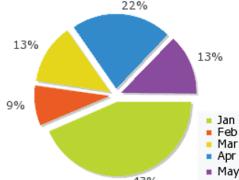
CRITERIO	DESCRIZIONE
(4) COESIONE DELLE FUNZIONALITÀ	- le classi appartengono al medesimo servizio primario e interagiscono nel caso d'uso. evitare di aggregare servizi indipendenti come 'autenticazione', 'inserisci lista della spesa' e 'cerca negozio', ma specializzarsi su uno di essi.
(5) PORTABILITÀ	- l'applicazione deve essere eseguibile sui PC Windows dell'aula con il pacchetto all-in-one. Fornire script sql, zip di progetto netbeans, eventuali librerie o script .bat
(6) DOCUMENTAZIONE	- composta dal documento di analisi, documento di progetto (con un diagramma di classe), e dal documento di collaudo (manuale utente)
(7) TEST FUNZIONALE	- esecuzione dello scenario descritto nel documento di collaudo; assumere che l'utente sia disciplinato e non considerare scenari di robustezza diversi dal caso d'uso
(8) CODICE INEDITO	- rilevanti parti di codice molto simile tra due progetti, anche di appelli diversi, rendono il codice edito per il progetto che viene presentato dopo
(9) UTILITÀ	- l'applicazione deve essere di utilità per qualche utente del mondo reale, lo scenario pensato è realistico. l'utente per il quale si sviluppa l'applicazione non deve essere né lo sviluppatore né il docente.
(10) NON RIDONDANZA DEL CODICE	- il codice è ridondante se vi sono parti di codice di controllo o di dati che sono molto simili. ridurre al minimo il popolamento dei dati nel database.
(11) CAPACITÀ DI MANUTENZIONE	- lo studente è in grado di manutenere l'applicazione: localizzare il codice corrispondente alle richieste del docente ed effettuare modifiche minime in sede di esame, accedendo alle specifiche e/o al web se necessario

Documento di analisi: vista statica dell'interfaccia

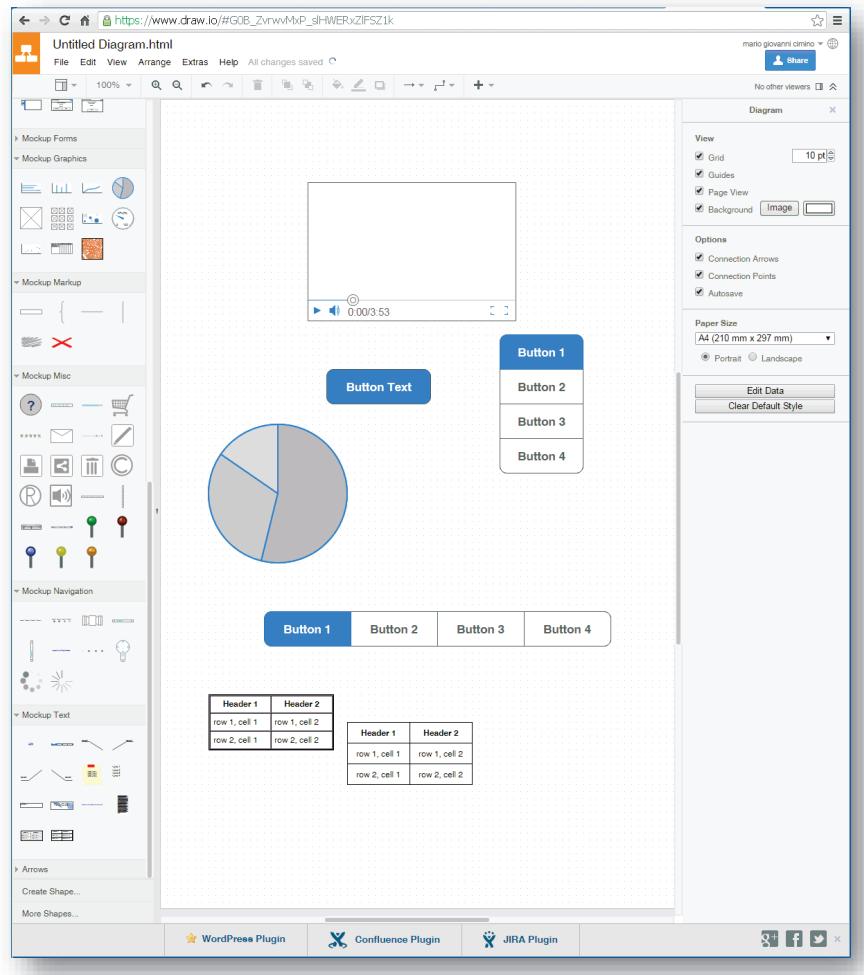
10 of 28

- Si rappresenta uno schizzo dell'interfaccia e si descrive uno scenario di utilizzo, sottolineando i termini che appaiono nell'interfaccia (label).
 - Porre attenzione ai termini usati perchè poi ci si aspetta che termini uguali o molto simili siano usati per i nomi delle classi, dei metodi, degli attributi, dello schema del database, dei tag xml, ecc. (terminologia del dominio applicativo).
 - Inserire anche un aspetto grafico nell'interfaccia e non solo controlli per data entry
 - Lo schizzo dell'interfaccia è detto wireframe o mockup e include solo struttura statica senza stile, non è un prototipo.
- Adoperare draw.io (<https://www.draw.io/>) o tool simili per creare ed esportare in formato immagine da inserire poi nel pdf

Inserimento Spedizione

Nome di spedizione	<input type="text"/>																									
Indirizzo di spedizione	<input type="text"/>																									
CAP, Città di spedizione	<input type="text"/>																									
Prodotti da spedire <table border="1" style="margin-top: 5px; border-collapse: collapse; width: 100%;"> <thead> <tr> <th style="width: 15%;">Codice Prodotto</th> <th style="width: 15%;">Descrizione</th> <th style="width: 15%;">Quantità</th> <th style="width: 15%;">Prezzo Unitario</th> <th style="width: 15%;">Prezzo Multiplo</th> </tr> </thead> <tbody> <tr><td> </td><td> </td><td> </td><td> </td><td> </td></tr> </tbody> </table>		Codice Prodotto	Descrizione	Quantità	Prezzo Unitario	Prezzo Multiplo																				
Codice Prodotto	Descrizione	Quantità	Prezzo Unitario	Prezzo Multiplo																						
Valore mensile spedizioni	 Subtotale Tasse Totale																									
Identificativo Ordine	<input type="text"/> ANNULLA CONFERMA																									

- Su draw.io ci sono tavolozze per mockup di vario tipo



Documento di analisi: vista dinamica (scenario)

1. l'Utente inserisce il Nome di spedizione
2. l'Utente inserisce l'Indirizzo di spedizione
3. l'Utente inserisce CAP, Città di spedizione
4. FOR EACH Codice Prodotto inserito in Prodotti da spedire
 - 4.1 Il Sistema visualizza la Descrizione e il Prezzo Unitario
 - 4.2 l'Utente inserisce la Quantità
 - 4.3 il Sistema visualizza il Prezzo multiplo
 - 4.4 il Sistema visualizza il Subtotale
 - 4.5 il Sistema visualizza le Tasse
 - 4.6 il Sistema visualizza il Totale
 - 4.7 il Sistema aggiorna il Valore Mensile Spedizioni
5. IF l'Utente preme Conferma
 - 7.1 il Sistema archivia i dati e visualizza l'Identificativo Ordine
8. IF l'Utente preme Annulla
 - 8.1 il Sistema svuota tutti i campi inseriti

FILE DI CONFIGURAZIONE LOCALE IN XML

All'avvio il Sistema legge dal file di configurazione i seguenti dati:

- la percentuale di tasse da applicare al subtotale per ottenere il totale
- la quantità massima di prodotti che si possono spedire
- numero di righe della tabella dei prodotti da inserire
- nome, indirizzo, cap, città del mittente
- font, dimensioni, colore del background
- l'indirizzo IP del client, l'indirizzo IP e la porta del server di log

CACHE LOCALE DEGLI INPUT

Alla chiusura il Sistema, se identificativo ordine è vuoto, salva su file binario: nome di spedizione, indirizzo di spedizione, prodotti da spedire.

All'avvio il Sistema carica dal file binario i suddetti dati.

ARCHIVIO

Il Sistema archivia i dati:

- nome, indirizzo, cap, città del mittente
- nome, indirizzo, cap, città di spedizione
- codice e quantità per ogni prodotto inserito
- percentuale di tasse da applicare
- identificativo ordine
- data ed ora di inserimento della spedizione

FILE DI LOG REMOTO IN XML

Il sistema invia una riga di log ad ogni evento di seguito

- Avvio dell'applicazione ("AVVIO")
- Pressione dei pulsanti "CONFERMA" o "ANNULLA"
- Termine dell'applicazione ("TERMINA")

La riga di log contiene: nome dell'applicazione, indirizzo IP del client, data-ora corrente, l'etichetta associata all'evento

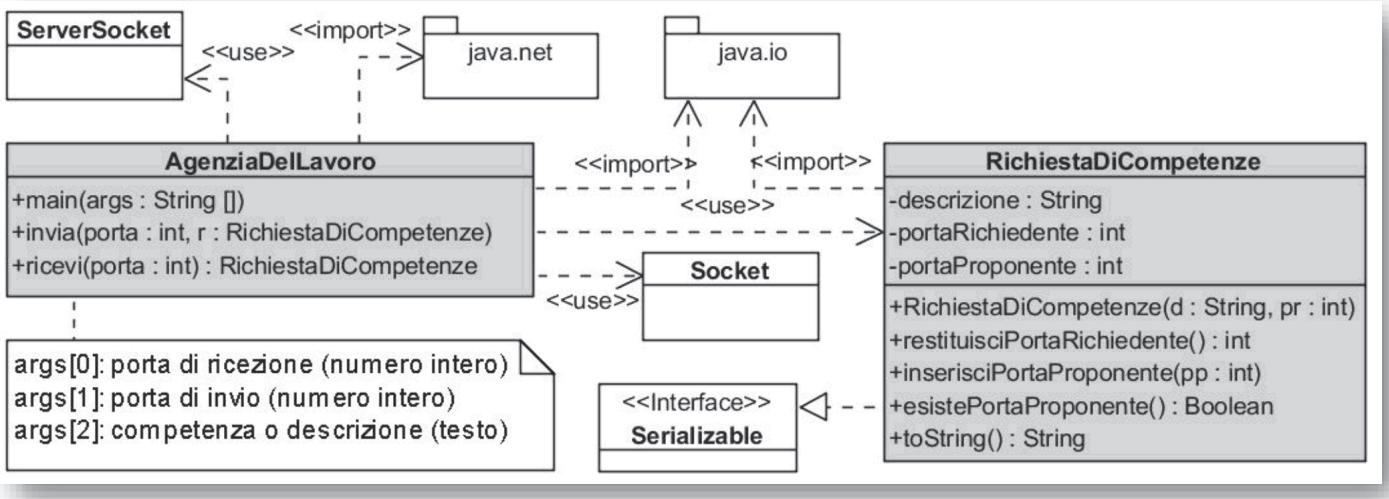
N.B. IL DOCUMENTO DI ANALISI CON LE PARTI SINORA EVIDENZIATE VA INVIATO PER EMAIL PER UN PRIMO RISCONTRO DEL DOCENTE PRIMA DI PASSARE ALLA FASE DI PROGETTO

a) Si può includere nel server di log del contenuto applicativo?

No, il server di log è pensato come un sistema di raccolta di statistiche di navigazione dell'interfaccia grafica, ossia dati che permetterebbero di fare il redesign del layout (es. due pulsanti vengono premuti sempre in sequenza per cui si metteranno vicini; oppure un menu non è adoperato mai e quindi si rimuove)

Per cui il server di log contiene solo le label dei controlli ma non i relativi contenuti, che andranno su un archivio

- Nella fase di progetto occorre disegnare un diagramma di classe con elementi del tipo quelli raffigurati di seguito
- Si può usare draw.io (<https://www.draw.io/>)



Nel documento di progetto inoltre si scrivono le responsabilità di ogni classe, come di seguito.

Documento di progetto: responsabilità di alcune classi

Classe **CacheUltimaSpedizione**: contiene su file binario gli ultimi dati inseriti; preleva i dati dal form di inserimento; rimette i dati nel form di inserimento; invoca il ricalcolo di subtotale e totale.

Classe **EventoDiNavigazioneGUI**: contiene le informazioni di un evento del log; si serializza in XML; invia l'evento di log in XML alla classe **LogDiNavigazioneGUI**.

Classe **LogDiNavigazioneGUI**: (server) riceve un evento di log XML; invoca la validazione dell'evento di log tramite XML Schema, aggiunge la riga XML al file di log in modo incrementale

Classe **ParametriDiConfigurazione**: legge il file di configurazione XML; invoca la validazione del file di configurazione XML, deserializza il contenuto XML come come oggetto Java; mette a disposizione di qualsiasi classe i vari parametri.

Classe **ArchivioSpedizioni**: restituisce descrizione e prezzo unitario di un dato codice prodotto; restituisce il valore mensile delle spedizioni dell'ultimo anno; inserisce una nuova spedizione in archivio; restituisce l'ultimo identificativo ordine inserito

Classe **InserimentoSpedizione**: inizializza la GUI; è il controller applicativo che risponde agli eventi invocando le opportune classi.

Il documento di progettazione deve essere controllato dal docente?

No, e non deve essere definitivo; se non si hanno domande specifiche basta inviare il documento di progetto via email come traccia dell'avvenuta progettazione, dopo aver ultimato la medesima a seguito della prototipazione.

Mentre il documento di analisi deve essere controllato dal docente (le dimensioni medie di un progetto sono note al docente), per il documento di progetto è responsabilità dello studente applicare le regole di buona progettazione (es. chiarezza dei nomi) e verificare che tutte le funzionalità che servono sono implementate da queste classi.

Si potranno comunque fare altre 2-3 iterazioni per migliorarlo, quindi non occorre soffermarsi troppo perché sia perfetto, meglio fare sucessivamente le rifiniture.

Comunque il documento di progettazione che vale è quello finale sottomesso all'esame.

Consigli su come tracciare l'architettura di progetto

Trovare le classi

Cosa dovrebbe essere una classe?

1. una classe rappresenta un'astrazione ben definita nel dominio del problema;
2. una classe descrive concetti del dominio di applicazione nel mondo reale. Per esempio, Cliente, Prodotto, ContoCorrente, etc..

Le operazioni nelle classi specificano i servizi fondamentali che la classe deve offrire.

La figura seguente è un esempio di una classe:



Cosa rende una classe una buona classe?

- Il suo nome riflette il suo intento. Consideriamo un sistema per il commercio elettronico. *Cliente* sembrerebbe riferire qualcosa di molto preciso del mondo reale e quindi è un buon candidato per una classe. Stessa cosa per *CarrelloDellaSpesa* (*ShoppingBasket*).

Una classe *VisitatoreSitoWeb* invece sembra avere una semantica vaga. Infatti, visitatore di un sito web sembra più un ruolo interpretato dal *Cliente* che una classe con la sua semantica.

- È un'astrazione ben definita che modella uno specifico elemento del dominio del problema ed ha una semantica chiara ed ovvia.
- Ha un insieme di responsabilità piccolo e ben definito. Ad esempio, la classe *CarrelloDellaSpesa* ci si aspetta che abbia le responsabilità “aggiungi articolo al carrello”, “rimuovi articolo dal carrello”, “visualizza gli articoli nel carrello”. Questo è un insieme coesivo di responsabilità, perché tutte le responsabilità lavorano verso lo stesso goal – gestire il carrello della spesa.
- Ha un'alta coesione. Nell'esempio della classe *CarrelloDellaSpesa* se aggiungessimo responsabilità come “valida la carta di credito” o “accetta pagamento” alla classe, perderemmo sicuramente la coesione, perché le responsabilità a questo punto si riferirebbero ad obiettivi diversi. Queste responsabilità sembrerebbero appartenere più a classi come *CompagniaCartaDiCredito* o *ControlloInUscita*.
- Ha un basso accoppiamento. L'accoppiamento è misurato come il numero di altre classi con cui una data classe ha relazioni. Una buona distribuzione delle responsabilità tra classi porterà ad un basso accoppiamento. La localizzazione del controllo o di molte responsabilità in una singola classe tende ad incrementare l'accoppiamento di quella classe.

Regole empiriche per le classi

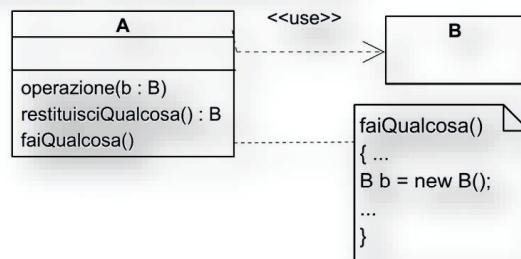
- Mantieni le classi semplici: attribuisci ad ogni classe un numero limitato di responsabilità (da 3 a 5, normalmente)
- Diffida delle classi onnipotenti: classi come *Sistema* o *Controllore* tenderanno ad essere sovraccaricate di responsabilità. Controlla se le responsabilità attribuite a queste classi possano essere organizzate in sottoinsiemi coesivi. È probabile che questi sottoinsiemi possano essere trasformati in classi separate.
- Evita classi *solitarie*: in una buona analisi object-oriented le classi collaborano per fornire servizi agli utenti.
- Evita molte classi con poche responsabilità: se il modello ha molte classi con una o due responsabilità diventa difficile da capire e seguire. Cerca di compattare classi affini.
- Evita “funziodi”. Una funzioide è una normale procedura mascherata da classe.

Dipendenza

dipendenza: "una relazione tra due elementi in cui una modifica ad un elemento (il fornitore) può influenzare l'altro elemento (il cliente) o fornire ad esso informazione necessaria". Una dipendenza è modellata da una linea tratteggiata con una freccia all'estremo. Le dipendenze possono avvenire tra classi, tra oggetti e classi, tra package e package e tra un'operazione ed una classe.

- **Dipendenza di uso:** il cliente usa alcuni servizi messi a disposizione dal fornitore. Ci sono cinque dipendenze d'uso, individuate da altrettanti stereotipi:

- «use» - il cliente usa il fornitore in qualche modo. In genere se lo stereotipo è omesso, la dipendenza è una dipendenza d'uso.



La classe A utilizza la classe B perché:

1. un'operazione della classe A ha bisogno di un parametro della classe B
2. un'operazione della classe A restituisce un oggetto della classe B
3. un'operazione della classe A usa un oggetto della classe B, ma non come attributo.

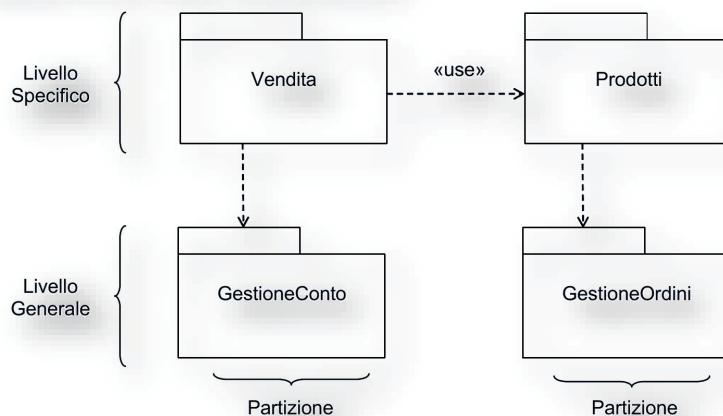
Consigli su come tracciare l'architettura di progetto

I casi 1. e 2. possono essere modellati più accuratamente da una dipendenza d'uso stereotipata «parameter» ed il caso 3. da una dipendenza d'uso stereotipata «call».

- «call» - un'operazione nel cliente invoca un'operazione nel fornitore.
- «parameter» - il fornitore è un parametro di un'operazione del cliente;
- «send» - il cliente è un'operazione che invia il fornitore (che deve essere un segnale) a qualche specificato target.

Analisi architetturale

Nell'analisi architettonica, tutte le classi di analisi coesivi e questi sono ulteriormente organizzati in partizioni e livelli. Ogni package di analisi all'interno di un livello è una partizione.



L'analisi architettonica dovrebbe minimizzare l'accoppiamento tra package, andando a minimizzare le dipendenze tra package ed il numero di elementi pubblici in ogni package, e massimizzando il numero di elementi privati.

In Analisi, i package generalmente vengono organizzati in due livelli: livello specifico e livello generale. Il livello specifico contiene funzionalità che sono specifiche della applicazione particolare. Il livello generale contiene funzionalità che sono più generalmente utili.

- Nel progetto non occorre implementare i package, ma occorre pensarli a livello concettuale per ottimizzare le dipendenze (es. nessuna dipendenza tra front-end e back-end)
- Nella fase di sviluppo e integrazione, documentare commentando semplicemente il codice aggiungendo eventuali riferimenti su web alle classi adoperate, se diverse da quelle usate nei laboratori
- CONSEGNARE unico file *matricola_cognome_titoloprogetto.zip* con:
 - un file pdf con documento di analisi, documento di progetto, documento di collaudo
 - progetto/i netBeans esportato/i come format/i zip e inclusivi di file SQL, eventuali script bat di avvio e immagini o file con dati/parametri.

Cosa significa classi distinte tra front-end, middleware e back-end

Significa che, ad esempio una stessa classe non può implementare sia interfaccia grafica che archiviazione su database.

In generale (a) *front-end*, letteralmente "frontale", "all'estremità più vicina a chi usa l'applicazione" e quindi "che fornisce un'interfaccia" o "che esegue funzioni di comunicazione con dispositivi esterni", es. interfaccia grafica o gestione parametri utente;

(b) *back-end*, letteralmente "posteriore", "all'estremità più lontana da chi usa l'applicazione", es. database o file di log;

(c) *middleware*, letteralmente “parte intermedia”, è lo strato di software tra i primi due, es. la logica applicativa e la gestione del flusso dati e di controllo.

Nel progetto di Programmazione la logica applicativa è secondaria perché gli argomenti vertono su Java I/O e non su algoritmi o workflow complessi.

Cosa inserire nella documentazione di collaudo? Che differenza c'è tra manuale utente e documento di analisi.

Il documento di analisi è astratto, presenta un (o più) *casi d'uso* su uno schizzo di interfaccia e su brevi descrizioni dei vari Java I/O. Mentre il manuale utente è concreto, descrive step by step come si usa l'applicazione in uno (o più) *casi di test*.

I casi di test sono basati sul caso d'uso stabilito in fase di analisi, ma mostrano nel dettaglio attraverso le schermate delle applicazioni client i dati caricati nel db esportato (schermata di MySQL Client), gli input dell'utente (schermata dell'applicazione), i dati di configurazione (schermata di notepad++), gli output risultanti (schermate applicative, schermata MySQL client, schermata della console di log remoto e del relativo file XML), in modo da rendere completamente riproducibile il collaudo.

Dovendo evitare metodi più grandi di una videata, come si fa per il metodo start che inizializza l'interfaccia?

Es. 1) estendere componenti grafici complessi, es. la tabella, come delle classi inclusive di stile, struttura, contenuto, e poi dal metodo principale si istanziano semplicemente;

Es. 2) segmentare il metodo principale in più macro-funzioni (es. più funzioni inerenti la struttura, lo stile, il comportamento) ognuna delle quali diventa un metodo, e poi chiamare i metodi in successione.

Attenzione: l'estensione es. di *TableView* come *GestisciTableView* non va bene senza il giusto nome per la classe estesa. Il termine "gestisci" è onnipotente, non fa capire cosa ha in più la classe estesa rispetto a quella Java (la quale ha già dei metodi che la gestiscono). Es. una classe *TableView* che visualizza la lista della spesa si potrebbe chiamare *ListaDellaSpesaVisuale*. Quindi l'estensione deve essere caratterizzata dall'applicazione e non da funzionalità generiche come "gestisci" perché stiamo facendo classi applicative e non librerie.

FILE DI CONFIGURAZIONE LOCALE IN XML (Definire XSD e validare dinamicamente)

```
<?xml version="1.0" encoding="UTF-8"?>
<Parametri>
  <economici>
    <tasse unita "%">21</tasse>
    <massimaQuantitaPerTipologia>3</massimaQuantitaPerTipologia>
    <massimoNumeroTipologieProdotti>5</massimoNumeroTipologieProdotti>
  </economici>
  <stilistici>
    <font>Times New Roman</font>
    <dimensioneFont unita="pt">12</dimensioneFont>
    <coloreSfondo>green</coloreSfondo>
  </stilistici>
  <tecnologici>
    <indirizzolPClient>131.114.80.255</indirizzolPClient>
    <indirizzolPServerLog>131.114.90.255</indirizzolPServerLog>
    <portaServerLog>80</portaServerLog>
  </tecnologici>
</Parametri>
```

Definire anche degli attributi nel proprio schema, non limitarsi agli elementi.

FILE DI LOG REMOTO IN XML (Occorre definire anche XSD e validare dinamicamente)

```
<?xml version="1.0" encoding="UTF-8"?>
<Evento>
  <nomeApplicazione>Inserimento Spedizione</nomeApplicazione>
  <indirizzolP>131.114.50.255</indirizzolP>
  <data formato= "aaaa-mm-gg" >2011-12-20</data>
  <ora formato= "hh-mm-ss" >23:30:11</ora>
  <nomeEvento>avvio</nomeEvento>
</Evento>

<?xml version="1.0" encoding="UTF-8"?>
<Evento>
  <nomeApplicazione>Inserimento Spedizione</nomeApplicazione>
  <indirizzolP>131.114.50.255</indirizzolP>
  <data formato= "aaaa-mm-gg" >2011-12-20</data>
  <ora formato= "hh-mm-ss" >23:40:01</ora>
  <nomeEvento>conferma</nomeEvento>
</Evento>
```

Il server di log valida il singolo XML in arrivo e lo aggiunge ad un file di testo. Tale file di testo non deve essere validato perché non ha i tag di chiusura, essendo un log aperto.

Programmazione

LAB 09

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

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

Illustrazione dei compiti d'esame e FAQ

2 of 2

- Download ed illustrazione dei testi d'esame degli appelli precedenti
- Illustrazione del formato della traccia dell'esercizio 2, con particolare riferimento ai diagrammi UML
- Illustrazione con maggiore dettaglio di un testo con Socket I/O e un testo con XML I/O

DOMANDE FREQUENTI (sezione aggiornata man mano)

01) gg/mm/aaaa: Domanda?

Risposta.