Il Linguaggio Java

Utilizzo di librerie alternative, caso di studio :

XStream e ***JDOM***, serializzazione basata sullo standard XML

Mario G. Cimino, 2005

Terminologia

- ✓ **specifica** = come la tecnologia deve funzionare: cosa dovrebbe fare (metodi) e non fare (vincoli)

- ✓ **standard** = specifica approvata dalla maggioranza di una comunità del settore. Es. VHS (standard **aperto** per il formato dei dati video sui nastri magnetici per i videoregistratori), MPEG, SOAP, XML,…

  … WINDOWS (standard **proprietario** "de facto" per la gestione delle risorse HW dei personal computer, con la maggioranza del mercato mondiale).

Concetti introduttivi

- ✓ Lo sviluppo di un' applicazione *Java* distribuita comporta l'utilizzo di diverse componenti API (*Application Programming Interface*) e l'integrazione con soluzioni di terze parti (*Database Management System, File System, Web Server, Firewall,…*).

- ✓ La tecnologia ***Java***, i cui componenti sono eseguiti sulla *Java Virtual Machine*, permette di realizzare applicazioni ***platform independent***.

- ✓ *Java* fornisce anche un valido supporto alla *serializzazione* degli oggetti, ossia alla loro codifica in un formato lineare che possa viaggiare in uno stream. Uno stream può essere a sua volta collegato ad un file per rendere persistente l'oggetto, oppure ad un socket per trasmettere dati ad un processo residente su un host remoto.

- ✓ Tuttavia, questa codifica si basa su un formato binario legato a *Java*. Ciò significa la necessità di N interfacce per N applicazioni "non Java" e – dualmente – altrettante interfacce per ogni formato legato ad altri framework di sviluppo o applicativi.

- ✓ Il linguaggio ***XML (Extensible Markup Language)***, derivato da una famiglia di linguaggi di markup nati per codificare i documenti web, consente di strutturare i dati in documenti *plain text* e trasportarli su *HTTP*.

- ✓ 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***.

- ✓ 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 intelleggibili per i progettisti.

- ✓ Ciò consente di realizzare la ***integrazione fra le organizzazioni***: tecnicamente parlando, due organizzazioni che intendono cooperare hanno bisogno di poter scambiarsi protocolli, formati, contenuti dei messaggi, descrizioni dei rispettivi processi.

XStream

- ✓ 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`).

- ✓ Non richiede alcun mapping tra gli oggetti Java ed elementi XML, il cui schema è semplice da interpretare.

- ✓ Tipico utilizzo:

  a) Persistenza (archiviare la struttura XML in un file o in un database)

  b) Trasporto (trasmettere l'oggetto tra due piattaforme applicative)

  c) Configurazione (inizializzazione degli oggetti da parte dell'utente)

- ✓ Download: **`http://xstream.codehaus.org/`**
  `http://xstream.codehaus.org/javadoc/index.html`

```java
// XStream_test.java
// http://xstream.codehaus.org/download.html

import com.thoughtworks.xstream.XStream;
import java.io.*;
import java.util.*;

    class Computer {

        private String type;
        public  Computer(String type) { this.type = type; }
    }

    class Person {
        private String  name;
        private List    toys = new ArrayList();
        public  Person(String name, List books) { this.name = name; this.toys = books; }
    }

public class XStream_test {

    public static void main(String[] args) {

        XStream xstream = new XStream();
        xstream.alias("person", Person.class);
        xstream.alias("computer", Computer.class);

        List toys = new ArrayList();
        toys.add(new Computer("apple"));
        Person person = new Person("Joe", toys);

        String xml = xstream.toXML(person);          // serialize to XML
        System.out.println(xml);
        Object person2 = xstream.fromXML(xml);       // deserialize from XML
    }
}


// make.bat
// set CLASSPATH=
// javac -classpath xstream-1.1.jar;xpp3_min-1.1.3.4.I.jar  XStream_test.java
// java  -classpath xstream-1.1.jar;xpp3_min-1.1.3.4.I.jar; XStream_test

// output
//  <person>
//    <name>Joe</name>
//    <toys>
//      <computer>
//        <type>apple</type>
//      </computer>
//    </toys>
//  </person>
```

Modello dei dati e documenti XML

✓ Supponiamo di avere il semplice modello dei dati raffigurato di seguito in UML.

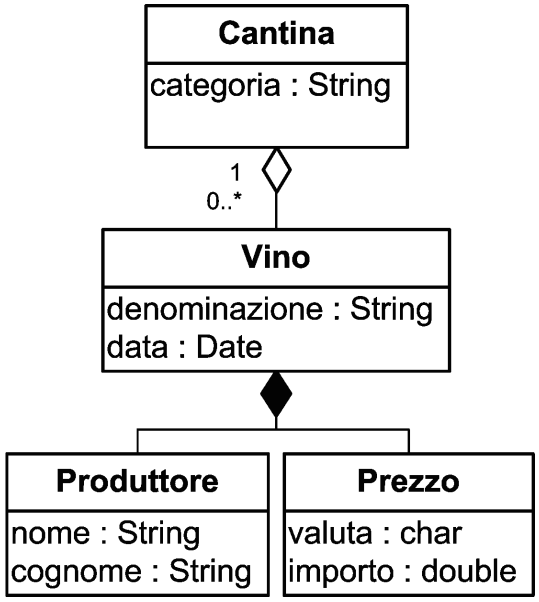


Fig.1 Diagramma delle classi

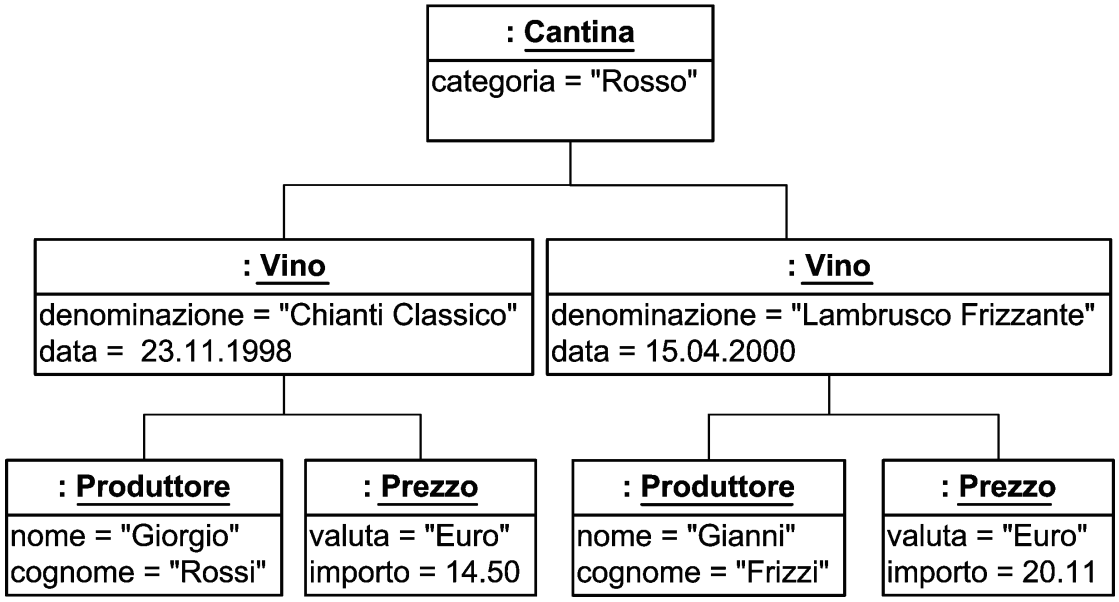


Fig.2 Diagramma degli oggetti (istanze)

✓ Una *Cantina* contiene diverse tipologie di *Vino* di una medesima categoria. Ciascun tipo di *Vino*, caratterizzato da una *denominazione* ed una *data* di produzione, si compone di un *Produttore* ed un *Prezzo*.

✓ Un *Produttore* è caratterizzato da *nome* e *cognome*, mentre il *Prezzo* dalla *valuta* e dall' *importo*.

- ✓ Vogliamo esprimere le entità di Fig.2 in un documento XML che ne mantenga la struttura gerarchica.
- ✓ Ci sono diversi modi di rappresentare gli oggetti di Fig.2 in *documenti delle istanze XML*.

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- Formato n.1 -->
<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**

Fig. 3 – Un possibile documento XML delle istanze

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- file cantina.xml, Formato n.2 -->
<Cantina categoria = "Rosso"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:noNamespaceSchemaLocation='cantina.xsd'>
  <Vino>
    <denominazione>Chianti Classico</denominazione>
    <data>1998-11-23</data>
    <Produttore>
     <nome>Giorgio</nome>
     <cognome>Rossi</cognome>
    </Produttore>
    <Prezzo valuta = "Euro" >14.50</Prezzo>
  </Vino>
  <Vino>
    <denominazione>Lambrusco Frizzante</denominazione>
    <data>2000-04-15</data>
    <Produttore>
     <nome>Gianni</nome>
     <cognome>Frizzi</cognome>
    </Produttore>
    <Prezzo valuta = "Euro" >20.11</Prezzo>
  </Vino>
</Cantina>
```

<!--commento-->

Progettazione di una struttura dati XML: attributo o elemento ?

a) nuovo elemento, se il dato è strutturato su linee multiple, o cambia spesso

b) nuovo attributo se il dato è una stringa semplice o numero (e non cambiano spesso), o appartiene a un set di possibilità predefinite.

Fig. 4 – Un altro possibile documento XML delle istanze

✓ Nello scambio di documenti XML, è necessario avere struttura e tipi di dato ben definiti, ossia che i documenti delle istanze siano conformi al medesimo schema. Lo schema è un documento XML, che definisce una classe di documenti XML, i cui

elementi ed attributi appartengono al namespace (ambito di visibilità) identificato da una URI predefinita.

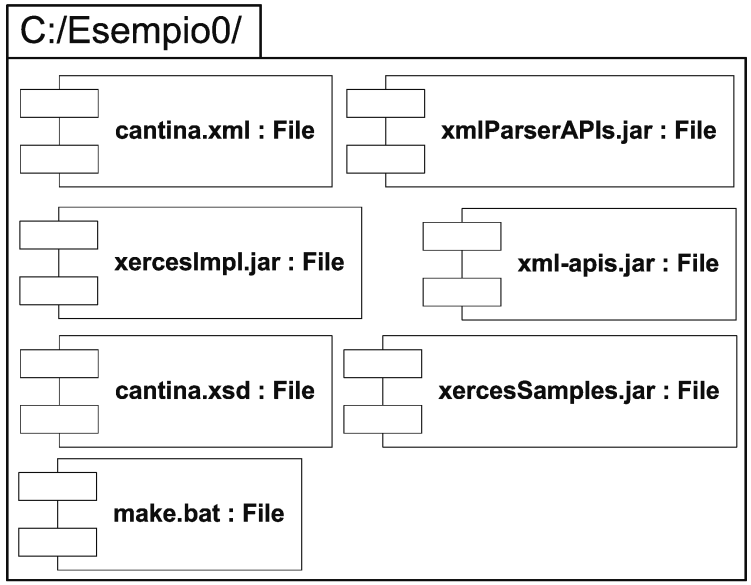
```
<?xml version="1.0" encoding="UTF-8"?>
<!-- file cantina.xsd, Schema per il Formato n.2,  -->
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" >
 <xs:element name="Cantina">
  <xs:complexType>
   <xs:sequence>
    <xs:element ref="Vino" minOccurs="0" maxOccurs="unbounded"/>
   </xs:sequence>
   <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:simpleContent>
    <xs:extension base="xs:decimal">
     <xs:attribute name="valuta" type="xs:string" use="required"/>
    </xs:extension>
   </xs:simpleContent>
  </xs:complexType>
 </xs:element>
</xs:schema>
```

Universal Resource Identifier (URI) è una stringa che identifica simbolicamente una risorsa nel web.
A differenza di una URL (Uniform Resource Locator), digitando una URI nel browser può non esserci alcuna risorsa.

Fig. 5 Un documento della classe (XML Schema) per il Formato n.2

Elaborazione di documenti XML

- ✓ Per validare il documento `cantina.xml` sullo schema `cantina.xsd` occorre un validating XML parser che supporti gli Schema, come il parser open source Xerces (progetto Apache XML). Scritto in Java, il package (http://xml.apache.org/dist/xerces-j/Xerces-J-bin.2.5.0.zip) include un programma a linea di comando chiamato `dom.Writer` (http://xml.apache.org/ xerces-j/domwriter.html).




```
rem make.bat
set CLASSPATH=
java -classpath xmlParserAPIs.jar;
     xercesImpl.jar;xercesSamples.jar;
     dom.Writer -v -s cantina.xml
```

Fig.6 File necessari e relativa composizione

- ✓ Se il documento è valido, DOMWriter farà semplicemente echo sullo schermo, altrimenti segnalerà gli errori (**`dom.Writer -h`** per l'help)

- ✓ *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.

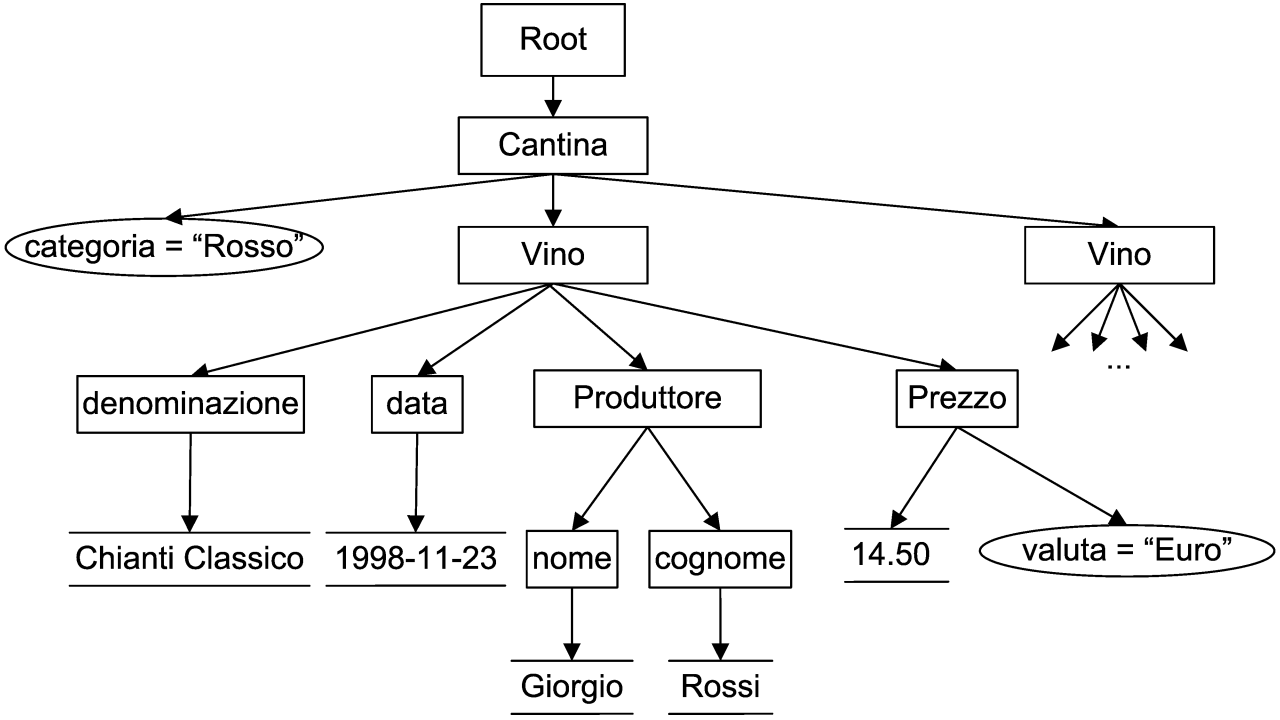


Fig.7 Struttura di parte del documento di Fig.4

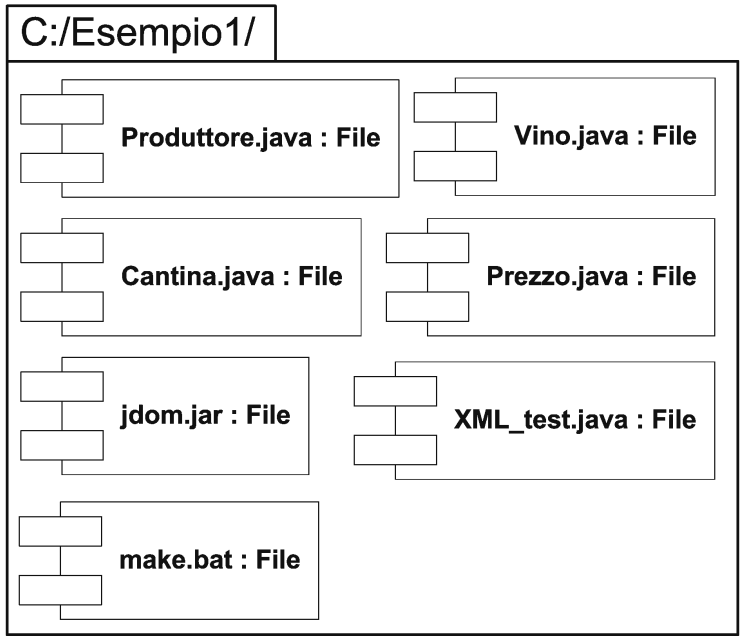
- ✓ *Simple API for XML* (SAX) è un altro modello, basato su un paradigma ad eventi, che non costruisce un modello dell’intero documento ma consente un accesso in sola lettura e la definizione di un handler di eventi per segnalare le categorie di interesse da riscontrare nel documento.

- ✓ Es. un documento tipo Word Processor (molto strutturato e dimensioni contenute) si presta ad essere manipolato in DOM, mentre per dati di un'agenda (notevoli dimensioni e struttura poco annidata) si predilige un approccio SAX.

- ✓ Nella piattaforma *Java* i package *javax.xml*, *org.w3c.dom*, *org.xml.sax* forniscono le API generiche per entrambi i modelli.

- ✓ Un'altra interfaccia API, chiamata JDOM, rappresenta un livello più alto del modello DOM del W3C, sviluppato appositamente per gestire documenti XML sfruttando tutte le funzionalità di java che semplificano la programmazione, interfacciandosi anche alle principali implementazioni SAX e DOM (Sun, IBM, Oracle, Xerces, Crimson,...).

- ✓ Una semplice applicazione Java che adopera JDOM si compone come in Fig.8 e si esegue con il comando `make` da shell di Windows. Il package **`jdom.jar`** è disponibile su *www.jdom.org*.




```
rem make.bat
set CLASSPATH=
javac -classpath jdom.jar *.java
java -classpath jdom.jar; XML_test
```

Fig.8 File necessari e relativa composizione

```java
// Produttore.java
import org.jdom.Element;
public class Produttore {
    private String nome;
    private String cognome;

    public Produttore(String nome, String cognome) {
        this.nome = nome;
        this.cognome = cognome;
    }

    public Element getElement() {
        Element e = new Element("Produttore");
        e.addContent(new Element("nome").addContent(nome));
        e.addContent(new Element("cognome").addContent(cognome));
        return e;
    }
}
```

```java
// Vino.java
import java.util.Date;
import org.jdom.Element;
import java.text.SimpleDateFormat;

public class Vino {

    private String       denominazione;
    private Date         data;
    private Produttore  produttore;
    private Prezzo       prezzo;

    public Vino(String denominazione, Date data, Produttore produttore, Prezzo prezzo) {
        this.denominazione    = denominazione;
        this.data             = data;
        this.produttore  = produttore;
        this.prezzo           = prezzo;
    }

    public Element getElement() {
        Element e = new Element("Vino");
        e.addContent(new Element("denominazione").addContent(denominazione));
        e.addContent(new Element("data").addContent(new SimpleDateFormat(
                                                              "yyyy-MM-dd").format(data)));
        e.addContent(produttore.getElement());
        e.addContent(prezzo.getElement());
        return e;
    }
}

// Cantina.java
import org.jdom.Element;

public class Cantina {

    private String  categoria;
    private Vino[]  vini;
```

```
    public Cantina(String categoria, Vino[] vini) {
        this.categoria = categoria;
        this.vini = vini;
    }

    public Element getElement() {
        Element e = new Element("Cantina");
        e.setAttribute("categoria",categoria);
        for (int i=0; i<vini.length; i++)
            e.addContent(vini[i].getElement());
        return e;
    }
}

// XML_test.java;
import java.util.*;
import java.io.*;
import org.jdom.*;
import org.jdom.input.*;
import org.jdom.output.*;
public class XML_test {
    public static void main(String[] args) throws Exception {
        Vino[] vini = {  new Vino(      "Chianti Classico",
                                  new GregorianCalendar(1998,10,23).getTime(),
                                  new Produttore("Giorgio","Rossi"),
                                  new Prezzo(14.50)
                                  ),
                         new Vino( "Lambrusco Frizzante",
                              new GregorianCalendar(2000,03,15).getTime(),
                                  new Produttore("Gianni","Frizzi"),
                                  new Prezzo(20.11)
                                )
                      };
        Cantina cantina = new Cantina("Rosso", vini);
        Document doc1 = new Document(cantina.getElement());

        XMLOutputter outputter = new XMLOutputter(Format.getPrettyFormat());
```

```
        outputter.output(doc1, new FileOutputStream("./cantina.xml"));

        SAXBuilder builder = new SAXBuilder();
        Document doc2 = builder.build("./cantina.xml");
        System.out.println(outputter.outputString(doc2));

        Element root = doc2.getRootElement();
        Attribute attribute = (Attribute) root.getAttributes().get(0);
        System.out.println("La " + attribute.getName() + " e` " + attribute.getValue());

        List children = root.getChild("Vino").getChildren();
        Element child = (Element) children.get(3);
        System.out.println("Il " + child.getName() + " e` " + child.getText());
        attribute = (Attribute) child.getAttributes().get(0);
        System.out.println("La " + attribute.getName() + " e` in " + attribute.getValue());
    }
}
```

✓ Come risultato della esecuzione, viene generato e visualizzato il file `cantina.xml` dopodiché vengono visualizzate alcune informazioni sulla struttura.

```
D:\Esempio1> make

<?xml version="1.0" encoding="UTF-8"?>
<Cantina categoria="Rosso">
  <Vino>
    <denominazione>Chianti Classico</denominazione>
    <data>1998-11-23</data>
    <Produttore>
      <nome>Giorgio</nome>
      <cognome>Rossi</cognome>
    </Produttore>
    <prezzo valuta="Euro">14.5</prezzo>
  </Vino>
```

```
  <Vino>
    <denominazione>Lambrusco Frizzante</denominazione>
    <data>2000-04-15</data>
    <Produttore>
      <nome>Gianni</nome>
      <cognome>Frizzi</cognome>
    </Produttore>
    <prezzo valuta="Euro">20.11</prezzo>
  </Vino>
</Cantina>


La categoria e` Rosso
Il prezzo e` 14.5
La valuta e` in Euro
```