XML Parsing

Marco Gradassi, Marco Scavarda, Marco Vallini

31 gennaio 2006

Indice

Capitolo 1

Introduzione al problema

1.1 Generalità

Il problema analizzato in questo documento riguarda la gestione di alcune informazioni aziendali registrate in un file XML. Il formato XML è un metalinguaggio in cui è possibile definire in modo efficace ed efficiente le informazioni specificandole attraverso etichette (tag) ed attributi. Le caratteristiche salienti del linguaggio sono:

- consente di strutturare i dati per modellare entità
- rappresenta una famiglia di tecnologie
- è modulare perchè permette di definire un formato per un determinato documento riutilizzando altri formati (anche standard)
- è indipendente dalla piattaforma quindi flessibile

1.2 Informazioni analizzate

Le informazioni analizzate riguardano la gestione di aziende, dipendenti e progetti. In particolare si focalizza l'attenzione sulle relazioni che intercorrono tra un'azienda, i suoi dipendenti ed i progetti svolti da questi. Le informazioni, registrate nel file di ingresso possono essere elaborate seguendo il seguente schema logico:

1. analisi delle informazioni presenti nel documento di ingresso
2. costruzione in memoria di una struttura dati adeguata
3. analisi delle relazioni, eventuale processing e visualizzazione
4. costruzione del documento di report che rappresenta le informazioni in modalità ipertestuale (documento HTML)

1.3 File d'ingresso

Il file d'ingresso, in cui sono registrate le informazioni da analizzare è il seguente:

```
<?xml version="1.0" ?>
<ElencoAziende>
    <Azienda>
        <Sigla></Sigla>
        <Nome></Nome>
        <Sede></Sede>
        <Impiegato>
            <CodiceFiscale></CodiceFiscale>
            <Nome></Nome>
            <DataNascita></DataNascita>
            <Professione></Professione>
            <Progetto>
                <Codice></Codice>
                <Nome></Nome>
                <Descrizione></Descrizione>
                <NumOre></NumOre>
            </Progetto>
        </Impiegato>
    </Azienda>
</ElencoAziende>
```

ElencoAziende è l'etichetta che racchiude l'elenco delle aziende presenti nel documento

Azienda rappresenta le informazioni registrate per l'azienda: *Sigla*, *Nome*, *Sede*, *Impiegato*

Impiegato rappresenta le informazioni specifiche degli impiegati: *CodiceFiscale*, *Nome*, *DataNascita*, *Professione*, *Progetto*

Progetto rappresenta le informazioni specifiche del progetto: *Codice*, *Nome*, *Descrizione*, *NumOre* Il seguente file è *ben formato* secondo il linguaggio XML. Tuttavia per essere *valido* è necessario associarlo ad una grammatica (e rispettarla) che ne definisca le tipologie dei dati presenti (stringhe, numeri, ...), le etichette e gli attributi. Gli strumenti utilizzati per definire una grammatica per un file XML sono: **XML Schema** e **DTD** (*Document Type Definition*). In questo documento si utilizzerà l'*XML Schema*.

1.4 Definizione della grammatica

Come detto nel paragrafo precedente, un *XML Schema* permette di definire una grammatica associata ad un documento XML e di *validarlo*. Di seguito si riporta la grammatica associata al file d'ingresso. Il file d'ingresso, in cui sono registrate le informazioni da analizzare è il seguente:

```
<?xml version="1.0" ?>
```

```
<xs:schema id="ElencoAziende" targetNamespace="draft_dati.xsd"
xmlns:mstns="draft_dati.xsd" xmlns="draft_dati.xsd"
xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns:msdata="urn:schemas-microsoft-com:xml-msdata"
attributeFormDefault="qualified" elementFormDefault="qualified">
<xs:element name="ElencoAziende" msdata:IsDataSet="true" msdata:Locale="it-IT"
msdata:EnforceConstraints="False">
<xs:complexType>
<xs:choice maxOccurs="unbounded">
<xs:element name="Azienda">
<xs:complexType>
<xs:sequence>
<xs:element name="Sigla" type="xs:string" minOccurs="0" />
<xs:element name="Nome" type="xs:string" minOccurs="0" />
<xs:element name="Sede" minOccurs="0">
<xs:simpleType>
<xs:restriction base="xs:string">
<xs:pattern value="[a-zA-Z\s]+" />
</xs:restriction>
</xs:simpleType>
</xs:element>
<xs:element name="Impiegato" minOccurs="0" maxOccurs="unbounded">
<xs:complexType>
<xs:sequence>
<xs:element name="CodiceFiscale" minOccurs="0">
<xs:simpleType>
<xs:restriction base="xs:string">
<xs:pattern value="[a-zA-Z0-9]+" />
</xs:restriction>
</xs:simpleType>
</xs:element>
<xs:element name="Nome" minOccurs="0">
<xs:simpleType>
<xs:restriction base="xs:string">
<xs:pattern value="[a-zA-Z\s]+" />
</xs:restriction>
</xs:simpleType>
</xs:element>
<xs:element name="DataNascita" type="xs:date" minOccurs="0" />
<xs:element name="Professione" minOccurs="0">
<xs:simpleType>
<xs:restriction base="xs:string">
<xs:pattern value="[a-zA-Z\s]+" />
</xs:restriction>
</xs:simpleType>
</xs:element>
<xs:element name="Progetto" minOccurs="0" maxOccurs="unbounded">
<xs:complexType>
```

```
<xs:sequence>
<xs:element name="Codice" minOccurs="0">
<xs:simpleType>
<xs:restriction base="xs:string">
<xs:pattern value="[a-zA-Z0-9]+" />
</xs:restriction>
</xs:simpleType>
</xs:element>
<xs:element name="Nome" type="xs:string" minOccurs="0" />
<xs:element name="Descrizione" type="xs:string" minOccurs="0
    " />
<xs:element name="NumOre" type="xs:integer" minOccurs="0" />
</xs:sequence>
</xs:complexType>
</xs:element>
</xs:sequence>
</xs:complexType>
</xs:element>
</xs:sequence>
</xs:complexType>
</xs:element>
</xs:choice>
</xs:complexType>
</xs:element>
</xs:schema>
```

1.5 La classe di inizializzazione

Scanner e parser sono inizializzati dalla classe *Main*, presente nel file `Main.java`. Il seguente codice istanzia e avvia lo scanner passando come primo argomento il file d'ingresso, successivamente si istanzia e si avvia il parser, catturando eventuali eccezioni riscontrate.

```
import java.io.*;

public class Main {
 static public void main(String argv[]) {
   try {

     XMLScanner l = new XMLScanner(new FileReader(argv[0]));

     parser p = new parser(l);

     Object result = p.parse();

   } catch (Exception e) {
     e.printStackTrace();
   }
 }
}
```

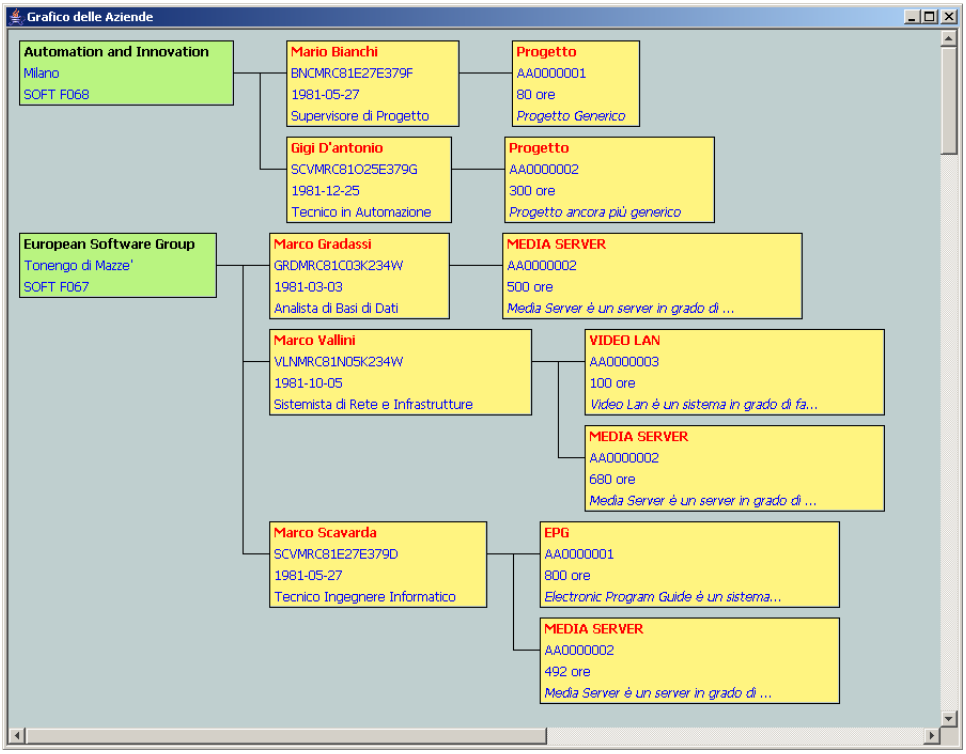

Figura 1.1: Grafico delle Aziende

1.6 Visualizzazione grafica delle informazioni

Partendo da una struttura di dati già istanziata con il parser in precedenza, è stato possibile realizzare un grafico che esplica in modo molto semplice le relazioni e gli attributi di ogni singola entità presente nel file xml.
In particolare la rappresentazione è del tipo ad albero con segmenti di unione tra i vari box che rappresentano le relazioni uno a molti.
E' stato interamente realizzato in Java, appoggiandosi alle librerie java.awt e javax.swing che permettono una semplice gestione delle primitive più comuni per la grafica.
Il disegno dei rettangoli e linee, il posizionamento di label contenenti testo e la gestione dei colori sono lo stretto necessario per la costruzione di un output grafico.

1.7 Report di uscita

Il documento di report permette di visualizzare le informazioni utilizzando il linguaggio HTML e sfuttando le caratteristiche dei documenti ipertestuali, rendendo quindi più semplice e flessibile la lettura. Il file HTML è organizzato in 2 sezioni principali:

- elenco aziende

- infomazioni di dettaglio per ogni azienda

La prima sezione contiene i riferimenti alle informazioni dettagliate per ogni azienda, agli impiegati e ai progetti. La seconda sezione contiene informazioni quali *Sigla*, *Sede*, *Impiegati*, *Progetti*. Le informazioni di ogni impiegato sono contenute in una tabella e dall'elenco dei progetti è possibile raggiungere ulteriori informazioni attraverso un *link ipertestuale*. Sia gli impiegati che le aziende dispongono di una tabella *Progetti*. Questa distinzione permette di rappresentare sia informazioni specifiche che di riepilogo.

Capitolo 2

Scanner

2.1 Definizione

Uno scanner è un'applicazione che serve ad eseguire un'analisi *lessicale*. Attraverso la definizione di regole (*espressioni regolari*) è possibile analizzare un file di testo e verificarne la correttezza (dal punto di vista lessicale). Lo Scanner è un elemento necessario per analizzare il file XML di ingresso e riconoscere le parole chiave caratterizzate dai tag (etichette, per es. *ElencoAziende*), i tipi dati contenuti ed il contenuto stesso (per es. l'identificatore **Data**, caratterizzata dal formato *yyyy-mm-dd*). A livello tecnico, lo scanner è l'implementazione di un *automa a stati finiti deterministico*. La trasformazione di espressioni regolari in automi a stati finiti deterministici e la relativa implementazione sono processi meccanici, quindi generalmente si utilizzano applicativi preposti a tale scopo. Lo strumento scelto come generatore di analizzatori lessicali è **JFlex**, che riceve in input un insieme di espressioni regolari e produce in output un'applicazione *Java*.

2.2 Struttura del programma sorgente (`scanner.jflex`)

Analizzando il codice sorgente dello scanner (incluso in appendice) si possono osservare tre sezioni:

- *sezione del codice*: contiene il codice utente che si vuole riportare nell'applicazione
- *sezione delle dichiarazioni*: contiene le dichiarazioni e le opzioni
- *sezione delle regole lessicali*: contiene le regole lessicali secondo il formalismo *espressioneRegolare azione*

2.2.1 Sezione del codice

La sezione del codice specifica l'inclusione di due package utili per avere la compatibilità con l'analizzatore grammaticale **Cup** e per l'utilizzo della libreria di I/O di Java:

```
import java_cup.runtime.*;
import java.io.*;
```

Oltre a queste informazioni, è presente il codice da includere nell'applicativo Java, utile alla memorizzazione e successivamente, al passaggio dei simboli al parser:

```
%{
  private Symbol symbol(int type) {
    return new Symbol(type, yyline, yycolumn);
  }
  private Symbol symbol(int type, Object value) {
    return new Symbol(type, yyline, yycolumn, value);
  }
%}
```

2.2.2 Sezione delle dichiarazioni

La sezione delle opzioni contiene: il nome della classe dello scanner da generare, la specifica di utilizzo dei caratteri *unicode*, l'opzione di utilizzo di *Cup*, e le opzioni che specificano di memorizzare le informazioni circa la riga e la colonna:

```
%class XMLScanner
%unicode
%cup
%line
%column
```

La sezione delle dichiarazioni contiene i seguenti identificatori, utili per semplificare la costruzione delle regole:

- new line, spazi e tabulazioni:

  ```
  nl = \r|\n|\r\n
  ws = [\t ]
  ```

- definizione dell'identificatore **Data** con il formato *yyyy-mm-dd*:

  ```
  Data = (19[0-9]{2}|(2[0-9]{3}))"-"(0[1-9]|1[0-2])
  "-"(0[1-9]|1[0-9]|2[0-9]|3[0-1])
  ```

- definizione dell'identificatore **Numero** per esprimere un numero intero senza segno:

  ```
  Numero = [0-9]+
  ```

- definizione degli identificatori **Parola**, **ParolaAlfaNum**, **ParolaApostrofo**, **ParolaNumApostrofo** e **Desc**. La differenza tra questi identificatori è nei caratteri riconosciuti, adatti quindi a situazioni diverse a seconda che sia un nome di persona, di azienda, una sigla, un codice fiscale:

```
Parola = [a-zA-Z]+
ParolaAlfaNum = [a-zA-z0-9]+
ParolaApostrofo = [a-zA-Z\']+
ParolaNumApostrofo = [a-zA-Z0-9\']+
```

- definizione degli stati esclusivi per raggruppare un insieme di regole valide solo all'interno di questi stati. Quando l'analizzatore si trova in uno di questi stati, le altre regole risultano disabilitate e sono attive solo le regole esplicitamente attivate nello stato:

  ```
  %xstate PREAMBOLO
  %xstate AZIENDA
  %xstate IMPIEGATO
  %xstate PROGETTO
  %xstate DESCRIZIONE
  ```

  PREAMBOLO: consente di valutare il tag *xml*.
  AZIENDA: consente di valutare in modo corretto il contenuto di *Sigla*, *Nome* e *Sede*.
  IMPIEGATO: consente di valutare in modo corretto il contenuto di *CodiceFiscale*, *Nome*, *Data* e *Professione*.
  PROGETTO: consente di valutare in modo corretto il contenuto di *Codice*, *Nome*, *Descrizione* e *NumOre*.
  DESCRIZIONE: consente di valutare in modo corretto il suo contenuto.

2.2.3 Sezione delle regole e stati esclusivi

La sezione delle regole contiene le espressioni regolari e le azioni associate a queste nel formato *espressioneRegolare azione/i*. In questa sezione sono presenti anche gli stati esclusivi elencati in precedenza. Le regole, le azioni e gli stati sono:

- *PREAMBOLO*: se trova la prima espressione regolare entra nello stato, quando trova la seconda torna allo stato iniziale. Nello stato preambolo tutti i caratteri salvo quelli della seconda espressione vengono tralasciati.

  ```
  "<?xml"  {yybegin(PREAMBOLO);}
  <PREAMBOLO>{
  "?>" {yybegin(YYINITIAL);}
  . {;}
  }
  ```

- *YYINITIAL*: quando si trovano i tag `<ElencoAziende>` e `</ElencoAziende>` si restituiscono i rispettivi simboli. Quando si incontra il tag `<Azienda>` si entra nello stato relativo. Gli altri caratteri vengono scartati.

  ```
  "<ElencoAziende>"
  {return symbol(sym.ElencoAziende_START_TAG);}
  "</ElencoAziende>"
  ```

```
{return symbol(sym.ElencoAziende_END_TAG);}

"<Azienda>"
{yybegin(AZIENDA); return symbol(sym.Azienda_START_TAG);}

{ws} {;}
{nl}  {;}
. {;}
```

- *AZIENDA*: quando si trova il tag `</Azienda>` si torna allo stato *YYINITIAL* e si restituisce il simbolo di chiusura del tag, quando si trovano i tag `<Sigla></Sigla>`, `<Nome></Nome>`, `<Sede></Sede>` si restituiscono i relativi simboli. Quando si incontra il tag `<Impiegato>` si entra nello stato *IMPIEGATO*. Le ulteriori regole specificano le tipologie di contenute riconosciute nello stato.

```
<AZIENDA>{
"</Azienda>"
{yybegin(YYINITIAL); return symbol(sym.Azienda_END_TAG);}

"<Sigla>"
{return symbol(sym.Sigla_START_TAG);}
"</Sigla>"
{return symbol(sym.Sigla_END_TAG);}

"<Nome>"
{return symbol(sym.Nome_START_TAG);}
"</Nome>"
{return symbol(sym.Nome_END_TAG);}

"<Sede>"
{return symbol(sym.Sede_START_TAG);}
"</Sede>"
{return symbol(sym.Sede_END_TAG);}

"<Impiegato>"
{yybegin(IMPIEGATO); return symbol(sym.Impiegato_START_TAG);}

{Numero}
{return symbol(sym.NUMERO, new Integer(yytext()));}
{Parola}
{return symbol(sym.PAROLA, new String(yytext()));}
{ParolaAlfaNum}
{return symbol(sym.PAROLANUM, new String(yytext()));}
{ParolaApostrofo}
{return symbol(sym.PAROLA_APOSTROFO, new String(yytext()));}
{ParolaNumApostrofo}
{return symbol(sym.PAROLA_NUM_APOSTROFO, new String(yytext()));}

{ws} {;}
```

```
{nl}  {;}
. {;}
}
```

- *IMPIEGATO*: quando si trova il tag `</Impiegato>` si torna allo stato *AZIENDA* e si restituisce il simbolo di chiusura del tag, quando si trovano i tag `<CodiceFiscale></CodiceFiscale>`, `<Nome></Nome>`, `<DataNascita></DataNascita>` e `<Professione></Professione>` si restituiscono i relativi simboli. Quando si trova il tag `<Progetto>` si entra nello stato *PROGETTO*. Le ulteriori regole specificano le tipologie di contenute riconosciute nello stato.

```
<IMPIEGATO>{

"</Impiegato>"
{yybegin(AZIENDA); return symbol(sym.Impiegato_END_TAG);}

"<CodiceFiscale>"
{return symbol(sym.CodiceFiscale_START_TAG);}
"</CodiceFiscale>"
{return symbol(sym.CodiceFiscale_END_TAG);}

"<Nome>"
{return symbol(sym.Nome_START_TAG);}
"</Nome>"
{return symbol(sym.Nome_END_TAG);}

"<DataNascita>"
{return symbol(sym.DataNascita_START_TAG);}
"</DataNascita>"
{return symbol(sym.DataNascita_END_TAG);}

"<Professione>"
{return symbol(sym.Professione_START_TAG);}
"</Professione>"
{return symbol(sym.Professione_END_TAG);}

"<Progetto>"
{yybegin(PROGETTO); return symbol(sym.Progetto_START_TAG);}

{Data}
{return symbol(sym.DATA, new String(yytext()));}
{Numero}
{return symbol(sym.NUMERO, new Integer(yytext()));}
{Parola}
{return symbol(sym.PAROLA, new String(yytext()));}
{ParolaAlfaNum}
{return symbol(sym.PAROLANUM, new String(yytext()));}
{ParolaApostrofo}
{return symbol(sym.PAROLA_APOSTROFO, new String(yytext()));}
```

```
{ws} {;}
{nl}  {;}
. {;}
}
```

- *PROGETTO*: quando si trova il tag `</PROGETTO>` si torna allo stato *IMPIEGATO* e si restituisce il simbolo di chiusura del tag, quando si trovano i tag `<Codice></Codice>`, `<Nome></Nome>`, `<NumOre></NumOre>` si restituiscono i relativi simboli. Quando si incontra il tag `<DESCRIZIONE>` si entra nello stato *DESCRIZIONE*. Le ulteriori regole specificano le tipologie di contenute riconosciute nello stato.

```
<PROGETTO>{

"</Progetto>"
{yybegin(IMPIEGATO); return symbol(sym.Progetto_END_TAG);}

"<Codice>"
    {return symbol(sym.Codice_START_TAG);}
"</Codice>"
{return symbol(sym.Codice_END_TAG);}

"<Nome>"
{return symbol(sym.Nome_START_TAG);}
"</Nome>"
{return symbol(sym.Nome_END_TAG);}

"<Descrizione>"
{yybegin(DESCRIZIONE); return symbol(sym.Descrizione_START_TAG);}

"<NumOre>"
{return symbol(sym.NumOre_START_TAG);}
"</NumOre>"
{return symbol(sym.NumOre_END_TAG);}

{Numero}
{return symbol(sym.NUMERO, new Integer(yytext()));}
{Parola}
{return symbol(sym.PAROLA, new String(yytext()));}
{ParolaAlfaNum}
{return symbol(sym.PAROLANUM, new String(yytext()));}
{ParolaApostrofo}
{return symbol(sym.PAROLA_APOSTROFO, new String(yytext()));}
{ParolaNumApostrofo}
{return symbol(sym.PAROLA_NUM_APOSTROFO, new String(yytext()));}

{ws} {;}
{nl}  {;}
. {;}
}
```

- *DESCRIZIONE*: quando si trova il tag `</Descrizione>` si torna allo stato *PROGETTO* e si restituisce il simbolo di chiusura del tag. Le ulteriori regole specificano le tipologie di contenuto riconosciute nello stato.

  ```
  <DESCRIZIONE>{

  "</Descrizione>"
  {yybegin(PROGETTO); return symbol(sym.Descrizione_END_TAG);}
  [^< \t]+
  {return symbol(sym.ALTRO, new String(yytext()));}
  {ws} {;}
  }
  ```

In fine le seguenti regole che specificano gli spazi, il ritorno a capo, le tabulazioni e qualsiasi altro carattere non specificato diversamente permettono di ignorare i caratteri:

```
{ws} {;}
{nl}  {;}
. {;}
```

Capitolo 3

Parser

3.1 Definizione

Data una grammatica non ambigua ed una sequenza di simboli in ingresso, un riconoscitore è un programma che verifica se la sequenza appartine al linguaggio definito. Un *analizzatore sintattico*, detto *parser* è un'applicazione in grado di associare ad una sequenza di simboli in input il relativo albero di derivazione. Gli algoritmi di analisi possono essere di due tipi: top-down e bottom-up. Lo strumento software utilizzato nel progetto per generare il parser è **Cup** che utilizza un algoritmo di tipo bottom-up. *Cup*, è un generatore di *analizzatori sintattici* che trasforma la descrizione di una grammatica context-free in un programma *Java* che riconosce ed analizza. La definizione delle regole sintattiche si integra a quella delle azioni da intraprendere. L'integrazione tra *JFlex* e *Cup* permette di ottenere un *analizzatore lessicale* ed un *analizzatore sintattico* integrati.

3.2 Struttura del programma sorgente (`parser.cup`)

Analizzando il codice sorgente del parser (incluso in appendice) si possono osservare tre sezioni:

- *sezione di Setup*: contiene le direttive ed il codice utente da includere
- *sezione dei simboli Terminali e Non Terminali*: contiene i simboli della grammatica
- *sezione delle regole*: contiene le regole della grammatica secondo il formalismo *simboloNonTerminale ::= corpoDellaRegola;*

3.2.1 Sezione di Setup

La sezione di setup specifica l'inclusione di cinque package utili per avere:

- impiego di analizzatore grammaticale a runtime
- utilizzo delle util di Java
- utilizzo dell'I/O di Java

- utilizzo delle interfacce grafiche *awt* e *Swing*

```
import java_cup.runtime.*;
import java.util.*;
import java.io.*;
import javax.swing.*;
import java.awt.*;
```

Si definisce anche la parte relativa all'inclusione di codice composta di due sezioni:

- personalizzazione del comportamento del parser (`init with {: :}`)
- aggiungere il codice all'interno della classe che realizza il parser (`paser code {: :}`)

3.2.2 Sezione Setup/Codice: `init with {: :}`

La seguente sezione definisce le operazioni preliminari che devono essere eseguite prima di ogni chiamata allo scanner e quindi prima che qualsiasi simbolo terminale venga fornito al parser. In particolare si osserva:

- la costruzione della *HashMap* per contenere di dati di *ElencoAziende*
- la costruzione degli oggetti *azienda*, *impiegato*, *progetto*

```
init with {:
ElencoAziende = new HashMap();
azienda = new Azienda();
impiegato = new Impiegato();
progetto = new Progetto();
:};
```

3.2.3 Sezione Setup/Codice: `paser code {: :}`

In questa sezione sono incluse tutte le funzioni per la gestione e la visualizzazione dei dati, in particolare:

- definizione delle classi e struttura dati per la registrazione dei dati in memoria
- definizione delle classi per la visualizzazione grafica
- definizione delle classi per la visualizzazione in formato HTML

Classe *Azienda*

La classe registra le seguenti informazioni: *NomeAzienda*, *SedeAzienda*, *SiglaAzienda* e *Impiegati*. Quest'ultima è una hashmap che contiene come chiave il codice fiscale dell'impiegato e come valore l'oggetto *impiegato*. Per semplicità l'implementazione dei metodi è riportata solo in appendice.

```
public class Azienda {
String NomeAzienda;
String SedeAzienda;
String SiglaAzienda;
HashMap Impiegati;

public Azienda(){
this.NomeAzienda = "";
this.SedeAzienda = "";
this.SiglaAzienda = "";
this.Impiegati = new HashMap();
}

public Azienda(Azienda a){
this.NomeAzienda = a.NomeAzienda;
this.SedeAzienda = a.SedeAzienda;
this.SiglaAzienda = a.SiglaAzienda;
this.Impiegati = (HashMap)a.Impiegati.clone();
}

public void SetNomeAzienda(String nomeAzienda){
}

public void SetSedeAzienda(String sedeAzienda){
}

public void SetSiglaAzienda(String siglaAzienda){
}

public void ResetAzienda(){
}

public void InsertImpiegato(Impiegato impiegato){
}

public void StampaAzienda(){
}
}
```

Classe *Impiegato*

La classe registra le seguenti informazioni: *NomeImpiegato*, *CodiceFiscale*, *DataNascita*, *Professione* e *Progetti*. Quest'ultima è una hashmap che contiene come chiave il codice del progetto e come valore l'oggetto *progetto*. Per semplicità l'implementazione dei metodi è riportata solo in appendice.

```
public class Impiegato {
String NomeImpiegato;
String CodiceFiscale;
String DataNascita;
```

```
String Professione;
HashMap Progetti;

public Impiegato(){
this.NomeImpiegato = "";
this.CodiceFiscale = "";
this.DataNascita = "";
this.Professione = "";
this.Progetti = new HashMap();
}

public Impiegato(Impiegato i){
this.NomeImpiegato = i.NomeImpiegato;
this.CodiceFiscale = i.CodiceFiscale;
this.DataNascita = i.DataNascita;
this.Professione = i.Professione;
this.Progetti = (HashMap)i.Progetti.clone();;
}

public void SetNomeImpiegato(String nomeImpiegato){
}

public void SetCodiceFiscale(String codiceFiscale){
}

public void SetDataNascita(String dataNascita){
}

public void SetProfessione(String professione){
}

public void ResetImpiegato(){
}

public void InsertProgetto(Progetto progetto){
}

public void StampaImpiegato(){
}
}
```

Classe *Progetto*

La classe registra le seguenti informazioni: *CodiceProgetto*, *NomeProgetto*, *Descrizione* e *NumOre*. Per semplicità l'implementazione dei metodi è riportata solo in appendice.

```
public class Progetto {
String CodiceProgetto;
String NomeProgetto;
```

```
String Descrizione;
int NumOre;

public Progetto(){
this.CodiceProgetto = "";
this.NomeProgetto = "";
this.Descrizione = "";
this.NumOre = 0;
}

public Progetto(Progetto p){
this.CodiceProgetto = p.CodiceProgetto;
this.NomeProgetto = p.NomeProgetto;
this.Descrizione = p.Descrizione;
this.NumOre = p.NumOre;
}

public void SetCodiceProgetto(String codiceProgetto){
}

public void SetNomeProgetto(String nomeProgetto){
}

public void SetDescrizione(String descrizione){
}

public void SetNumOre(int numOre){
}

public void ResetProgetto(){
}

public void StampaProgetto(){
}
}
```

Altre classi

Per le altre classi, quelle relative alla visualizzazione grafica e in formato HTML si rimanda ai capitoli 4 e 5.

3.2.4 Sezione dei simboli Terminali e Non Terminali

Lo scanner, come già detto in precedenza, è incaricato di eseguire l'analisi lessicale e ritornare i simboli che soddisfano le regole dei vari pattern.
Il parser è organizzato in regole gerarchiche che permettono la realizzazione più che intuitiva dell'analisi grammaticale.
La prima parte è la definizione dei simboli terminali, che può essere fatta a priori, subito dopo la scrittura del parser (infatti i simboli sono quelli riconosciuti

dal parser).

La seconda parte è la definizione dei simboli non terminali, che viene fatta man mano che si creano delle regole, poichè definiscono passaggi intermedi di riconoscimento grammaticale.

```
terminal ElencoAziende_START_TAG, ElencoAziende_END_TAG;
terminal Azienda_START_TAG, Azienda_END_TAG;
terminal Sigla_START_TAG, Sigla_END_TAG;
terminal Nome_START_TAG, Nome_END_TAG;
terminal Sede_START_TAG, Sede_END_TAG;
terminal Impiegato_START_TAG, Impiegato_END_TAG;
terminal CodiceFiscale_START_TAG, CodiceFiscale_END_TAG;
terminal DataNascita_START_TAG, DataNascita_END_TAG;
terminal Professione_START_TAG, Professione_END_TAG;
terminal Progetto_START_TAG, Progetto_END_TAG;
terminal Codice_START_TAG, Codice_END_TAG;
terminal Descrizione_START_TAG, Descrizione_END_TAG;
terminal NumOre_START_TAG, NumOre_END_TAG;
terminal NUMERO, PAROLANUM, DATA, PAROLA, ALTRO;
terminal PAROLA_APOSTROFO, PAROLA_NUM_APOSTROFO;

non terminal XML;
non terminal elenco_aziende, azienda, elenco_attributi_azienda,
non terminal attributi_azienda, elenco_impiegati, impiegato;
non terminal elenco_attributi_impiegato, attributi_impiegato;
non terminal elenco_progetti, progetto, elenco_attributi_progetto;
non terminal attributi_progetto;

non terminal frase_parola_e_num, frase_parola, frase_parola_e_parolanum;
non terminal frase_tutto, frase_parola_e_parola_apo, altro;
```

3.2.5 Sezione delle regole

- Per iniziare è obbligatorio indicare al parser il punto di ingresso, un simbolo non terminale da cui parte il processo di parsing.

  ```
  start with XML;
  ```

- La prima regola è cruciale, infatti è associata alle principali funzioni di reporting scritte in Java. Quando il documento soddisfa la prima regola significa che rispetta tutte le regole successive dichiarate nel parser. L'azione associata è la chiamata di 3 funzioni:

  - `public void StampaTutto()`, che stampa su standard output (shell dei comandi) un riassunto grezzo dei dati memorizzati nella struttura ElencoAziende, iterandola in tutti i sui attributi.
  - `public void StampaHTML()`, che scrive su un file, passato come argomento, il codice HTML relativo al reporting con collegamenti ipertestuali e tabelle strutturate con dati.

– `public void StampaGrafico()`, che visualizza un grafico di tipo box chart in una finestra, mostrando in modo significativo le dipendenze tra i vari elementi.

La regola è soddisfatta se:
inizio con un simbolo terminale `ElencoAziende_START_TAG` (`<ElencoAziende>`),
termino con un simbolo terminale `ElencoAziende_END_TAG` (`</ElencoAziende>`)
e al loro interno ho un simbolo non terminale che rappresenta un elenco di aziende

```
XML ::= ElencoAziende_START_TAG elenco_aziende ElencoAziende_END_TAG
       {: parser.StampaTutto(); parser.StampaHTML();
  parser.StampaGrafico(); :}
;
```

- Un elenco aziende è composto da una o più aziende:

```
elenco_aziende ::= elenco_aziende azienda.
       | azienda
;
```

- Un'azienda è definita da un elenco di attributi dell'azienda (`elenco_attributi_azienda`) compreso tra i simboli terminali `Azienda_START_TAG` (`<Azienda>`) e `Azienda_END_TAG` (`</Azienda>`).
L'azione corrispondente è l'inserimento di un'azienda nella struttura globale *ElencoAziende* e la successiva cancellazione dei dati temporanei presenti nell'oggetto *azienda*:

```
azienda ::= Azienda_START_TAG elenco_attributi_azienda Azienda_END_TAG
       {: parser.InsertAzienda(parser.azienda);
  parser.azienda.ResetAzienda(); :}
;
```

- Un elenco di attributi di un'azienda è composto da uno o più attributi di un'azienda:

```
elenco_attributi_azienda ::=  elenco_attributi_azienda attributi_azienda
       | attributi_azienda
;
```

- Gli attributi di un'azienda possono essere molteplici:
  – iniziano per `Sigla_START_TAG` e terminano per `Sigla_END_TAG`
  – iniziano per `Nome_START_TAG` e terminano per `Nome_END_TAG`
  – iniziano per `Sede_START_TAG` e terminano per `Sede_END_TAG`

  Le azioni associate richiamano i metodi della classe *Azienda*, che settano i valori dell'oggetto temporaneo *azienda* con i valori corrispondenti.
  Come ultima possibilità può esserci un elenco di impiegati.

```
attributi_azienda ::= Sigla_START_TAG frase_parola_e_parolanum:sigla
Sigla_END_TAG
       {: parser.azienda.SetSiglaAzienda((String)sigla); :}
       | Nome_START_TAG frase_tutto:nome_azienda Nome_END_TAG
       {: parser.azienda.SetNomeAzienda((String)nome_azienda); :}
       | Sede_START_TAG frase_parola_e_parola_apo:sede Sede_END_TAG
       {: parser.azienda.SetSedeAzienda((String)sede); :}
       | elenco_impiegati
;
```

`frase_parola_e_parolanum`, `frase_tutto`, `frase_parola_e_parola_apo` e altri simboli non terminali che si incontreranno in seguito sono composizioni precise di parole, numeri, apostrofi che identificano il tipo di contenuto permesso. Verranno esposti alla fine.

- Un elenco impiegati è composto da uno o più impiegati:

```
elenco_impiegati ::= elenco_impiegati impiegato
       | impiegato
;
```

- Un impiegato è definito da un elenco di attributi dell'impiegato (`elenco_attributi_impiegato`) compreso tra i simboli terminali `Impiegato_START_TAG` (`<Impiegato>`) e `Impiegato_END_TAG` (`</Impiegato>`).
  L'azione corrispondente è l'inserimento di un oggetto *Impiegato* nella HashMap di un oggetto *Azienda* e la successiva cancellazione dei dati temporanei presenti nell'oggetto *impiegato*.

```
impiegato ::= Impiegato_START_TAG elenco_attributi_impiegato
Impiegato_END_TAG
{: parser.azienda.InsertImpiegato(parser.impiegato);
   parser.impiegato.ResetImpiegato(); :}
;
```

- Un elenco di attributi di un impiegato è composto da uno o più attributi di un impiegato:

```
elenco_attributi_impiegato ::=  elenco_attributi_impiegato
attributi_impiegato
       | attributi_impiegato
;
```

- Gli attributi di un impiegato:
  - iniziano per `CodiceFiscale_START_TAG` e terminano per `CodiceFiscale_END_TAG`
  - iniziano per `Nome_START_TAG` e terminano per `Nome_END_TAG`
  - iniziano per `DataNascita_START_TAG` e terminano per `DataNascita_END_TAG`
  - iniziano per `Professione_START_TAG` e terminano per `Professione_END_TAG`

Le azioni associate richiamano i metodi della classe *Impiegato*, che settano i valori dell'oggetto temporaneo *impiegato* con i valori corrispondenti. Come ultima possibilità può esserci un elenco di progetti.

```
attributi_impiegato ::= CodiceFiscale_START_TAG PAROLANUM:codice_fis
CodiceFiscale_END_TAG
       {: parser.impiegato.SetCodiceFiscale((String)codice_fis); :}
       | Nome_START_TAG frase_parola_e_parola_apo:nome Nome_END_TAG
       {: parser.impiegato.SetNomeImpiegato((String)nome); :}
       | DataNascita_START_TAG DATA:data_nasc DataNascita_END_TAG
       {: parser.impiegato.SetDataNascita((String)data_nasc); :}
       | Professione_START_TAG frase_parola:prof Professione_END_TAG
       {: parser.impiegato.SetProfessione((String)prof); :}
       | elenco_progetti
;
```

- Un elenco progetti è composto da uno o più progetti:

```
elenco_progetti ::= elenco_progetti progetto
       | progetto
;
```

- Un progetto è definito da un elenco di attributi del progetto (`elenco_attributi_progetto`) compreso tra i simboli terminali `Progetto_START_TAG` (`<Progetto>`) e `Progetto_END_TAG` (`</Progetto>`).
  L'azione corrispondente è l'inserimento di un oggetto *Progetto* nella HashMap di un oggetto *Impiegato* e la successiva cancellazione dei dati temporanei presenti nell'oggetto *progetto*.

```
progetto ::= Progetto_START_TAG elenco_attributi_progetto Progetto_END_TAG
{: parser.impiegato.InsertProgetto(parser.progetto);
parser.progetto.ResetProgetto(); :}
;
```

- Un elenco di attributi di un progetto è composto da uno o più attributi di un progetto:

```
elenco_attributi_progetto ::= elenco_attributi_progetto attributi_progetto
       | attributi_progetto
;
```

- Gli attributi di un progetto:
  - iniziano per `Codice_START_TAG` e terminano per `Codice_END_TAG`
  - iniziano per `Nome_START_TAG` e terminano per `Nome_END_TAG`
  - iniziano per `Descrizione_START_TAG` e terminano per `Descrizione_END_TAG`
  - iniziano per `NumOre_START_TAG` e terminano per `NumOre_END_TAG`

  Le azioni associate richiamano i metodi della classe *Progetto*, che settano i valori dell'oggetto temporaneo *progetto* con i valori corrispondenti.

```
attributi_progetto ::= Codice_START_TAG PAROLANUM:cod_prog Codice_END_TAG
       {: parser.progetto.SetCodiceProgetto((String)cod_prog); :}
       | Nome_START_TAG frase_parola_e_num:nome_prog Nome_END_TAG
       {: parser.progetto.SetNomeProgetto((String)nome_prog); :}
       | Descrizione_START_TAG altro:desc_prog Descrizione_END_TAG
       {: parser.progetto.SetDescrizione((String)desc_prog); :}
       | NumOre_START_TAG NUMERO:no_prog NumOre_END_TAG
       {: parser.progetto.SetNumOre(((Integer)no_prog).intValue()); :}
;
```

- Definizioni di frasi:
  - frase composta da parole e numeri:

    ```
    frase_parola_e_num ::= frase_parola_e_num:a NUMERO:b
    {: RESULT = a + " " + b; :}
           | frase_parola_e_num:a PAROLA:b
    {: RESULT = a + " " + b; :}
           | PAROLA:a
    {: RESULT = a; :}
           | NUMERO:a
    {: RESULT = a; :}
    ;
    ```

  - frase composta da sole parole:

    ```
    frase_parola ::= frase_parola:a PAROLA:b
    {: RESULT = a + " " + b; :}
           | PAROLA:a
    {: RESULT = a; :}
    ;
    ```

  - frasecomposta da parole e parole+numeri:

    ```
    frase_parola_e_parolanum ::= frase_parola_e_parolanum:a PAROLANUM:b
           {: RESULT = a + " " + b; :}
           | frase_parola_e_parolanum:a PAROLA:b
           {: RESULT = a + " " + b; :}
           | PAROLA:a
           {: RESULT = a; :}
           | PAROLANUM:a
           {: RESULT = a; :}
    ;
    ```

  - frase composta da parole, numeri, parole+numeri e parole+numeri+apostrofi:

    ```
    frase_tutto ::= frase_tutto:a PAROLANUM:b
    {: RESULT = a + " " + b; :}
           | frase_tutto:a PAROLA:b
    {: RESULT = a + " " + b; :}
           | frase_tutto:a PAROLA_APOSTROFO:b
    {: RESULT = a + " " + b; :}
           | frase_tutto:a PAROLA_NUM_APOSTROFO:b
    {: RESULT = a + " " + b; :}
    ```

```
        | frase_tutto:a NUMERO:b
  {: RESULT = a + " " + b; :}
        | PAROLA:a
  {: RESULT = a; :}
        | PAROLANUM:a
  {: RESULT = a; :}
        | PAROLA_APOSTROFO:a
  {: RESULT = a; :}
        | PAROLA_NUM_APOSTROFO:a
  {: RESULT = a; :}
        | NUMERO:a
  {: RESULT = a; :}
  ;
```

– frase composta da parole e parole+apostrofi:

```
frase_parola_e_parola_apo::= frase_parola_e_parola_apo:a PAROLA:b
        {: RESULT = a + " " + b; :}
        | frase_parola_e_parola_apo:a PAROLA_APOSTROFO:b
        {: RESULT = a + " " + b; :}
        | PAROLA:a
        {: RESULT = a; :}
        | PAROLA_APOSTROFO:a
        {: RESULT = a; :}
;
```

– Frase composta da qualsiasi parola che non contenga spazi, tabulazioni e <. Inispensabile per rappresentare una descrizione generica dove son presenti caratteri molto inusuali:

```
altro ::= altro:a ALTRO:b {: RESULT = a + " " + b; :}
        | ALTRO:a          {: RESULT = a; :}
;
```

Capitolo 4

Funzionalità di visualizzazione grafica

La funzionalità di visualizzazione grafica parte dal momento in cui viene riconosciuto il testo in ingresso in tutta la sua integrità: quindi viene lanciata la funzione `public void StampaGrafico()`.

4.1 La funzione: `StampaGrafico`

```
public void StampaGrafico(void)
```

Si occupa di creare un oggetto di tipo `Grafico`, lo colloca in un oggetto di tipo `JFrame` con barre di scorrimento verticali e orizzontali. Questo è necessario per una corretta visualizzazione nel caso di molte aziende, impiegati o progetti presenti nel file di ingresso.

4.2 La classe: `Grafico`

```
public class Grafico extends JPanel
```

Questa classe itera tutta la struttura di oggetti *ElencoAziende* - *Azienda* - *Impiegato* - *Progetto* ricavando i valori dei singoli attributi (come nome, codice, ecc.). Richiama alcune funzioni scritte ad hoc per la visualizzazione a rettangoli delle *Aziende*, degli *Impiegati* e dei *Progetti*.

4.3 La funzione: `DisegnaBoxProgetto`

```
public void DisegnaBoxProgetto(Graphics g, Progetto p, int x,
int y, int x_line, int y_line, boolean first)
```

Si occupa di disegnare un singolo box di un *Progetto*, gli argomenti sono:

- `g` è il contesto grafico in cui viene disegnato il box
- `p` è l'oggetto di tipo *Progetto* che contiene i dati da stampare a video nei box

- `x` e `y` sono le coordinate del vertice superiore-sinistro del box *Progetto*
- `x_line` e `y_line` sono le coordinate del vertice che rappresenta il punto di giunzione tra un box *Impiegato* ed un box *Progetto*
- first indica se è il primo *Progetto* per l'*Impiegato* corrente (infatti necessita di una stampa differente)

4.4 La funzione: `DisegnaBoxImpiegato`

```
public void DisegnaBoxImpiegato(Graphics g, Impiegato i, int x,
int y, int x_line, int y_line, boolean first, int nprog)
```

Si occupa di disegnare un singolo box di un *Impiegato*, gli argomenti sono:

- `g` è il contesto grafico in cui viene disegnato il box
- `i` è l'oggetto di tipo *Impiegato* che contiene i dati da stampare a video nei box
- `x` e `y` sono le coordinate del vertice superiore-sinistro del box *Impiegato*
- `x_line` e `y_line` sono le coordinate del vertice che rappresenta il punto di giunzione tra un box *Azienda* ed un box *Impiegato*
- `first` indica se è il primo Impiegato per l'*Azienda* corrente (infatti necessita di una stampa differente)
- `nprog` è il numero di progetti che appartengono all'*Impiegato* corrente

4.5 La funzione: `DisegnaBoxAzienda`

```
public int DisegnaBoxAzienda(Graphics g, Azienda a, int x, int y)
```

Si occupa di disegnare un singolo box di un'*Azienda*, gli argomenti sono:

- `g` è il contesto grafico in cui viene disegnato il box
- `i` è l'oggetto di tipo *Azienda* che contiene i dati da stampare a video nei box
- `x` e `y` sono le coordinate del vertice superiore-sinistro del box *Azienda*

Capitolo 5

Funzionalità di reportistica

Questa funzionalità permette di generare un report in formato HTML per organizzare i dati in elenchi, tabelle e link costruendo un documento ipertestuale. La funzione che si occupa di questa funzionalità, denominata `StampaHTML()` è richiamata nella prima regola.

5.1 La funzione: `StampaHTML`

```
public void StampaHTML()
```

Si occupa di creare il file di uscita `output.html` e richiama al suo interno le seguenti funzioni:

- `StampaTAGI`: inizializza il documento
- `StampaListaAziende`: scrive la lista delle aziende
- `StampaAziende`: scrive le informazioni relative alle aziende
- `StampaTAGF`: chiude il documento

successivamente chiude il file di report.

5.2 La funzione: `StampaTAGI`

```
public void StampaTAGI(BufferedWriter out)
```

Si occupa di definire un nuovo documento HTML, scrivere il titolo e definire lo stile. L'unico argomento costituisce il riferimento al file di uscita.

5.3 La funzione: `StampaListaAziende`

```
public void StampaListaAziende(BufferedWriter out)
```

Si occupa di stampare l'elenco delle aziende presenti, iterando sulla struttura dati definita nella HashMap *ElencoAziende*. L'unico argomento costituisce il riferimento al file di uscita.

5.4 La funzione: `StampaAziende`

```
public void StampaAziende(BufferedWriter out)
```

Si occupa di stampare le singole informazioni delle aziende, degli impiegati, dei progetti, costruendo dinamicamente i collegamenti ipertestuali. L'unico argomento costituisce il riferimento al file di uscita.

5.5 La funzione: `StampaTAGF`

```
StampaTAGF(BufferedWriter out)
```

Si occupa di chiudere il documento ipertestuale, inserendo i tag di fine. L'unico argomento costituisce il riferimento al file di uscita.

Capitolo 6

Conclusioni

Appendice A

Codice Sorgente

A.1 Scanner

```
1   import java_cup.runtime.*;
2   import java.io.*;
3
4   %%
5
6   %class XMLScanner
7   %unicode
8   %cup
9   %line
10  %column
11
12
13  %{
14    private Symbol symbol(int type) {
15      return new Symbol(type, yyline, yycolumn);
16    }
17    private Symbol symbol(int type, Object value) {
18      return new Symbol(type, yyline, yycolumn, value);
19    }
20  %}
21
22  nl = \r|\n|\r\n
23  ws = [\t ]
24  Data = (19[0-9]{2}|(2[0-9]{3}))"-"(0[1-9]|1[0-2])"-"(0[1-9]|1[0-9]|2[0-9]|3[0-1])
25  Numero = [0-9]+
26  Parola = [a-zA-Z]+
27  ParolaAlfaNum = [a-zA-z0-9]+
28  ParolaApostrofo = [a-zA-Z\']+
29  ParolaNumApostrofo␣=␣[a-zA-Z0-9\']+
30  Desc = "</Descrizione>"
31  %xstate PREAMBOLO
32  %xstate AZIENDA
33  %xstate IMPIEGATO
34  %xstate PROGETTO
35  %xstate DESCRIZIONE
36
37  %%
38
39  "<?xml"                 {yybegin(PREAMBOLO);}
40  <PREAMBOLO>{
41  "?>"                    {yybegin(YYINITIAL);}
42  .                       {;}
43  }
44
45  "<ElencoAziende>"       {return symbol(sym.ElencoAziende_START_TAG);}
46  "</ElencoAziende>"      {return symbol(sym.ElencoAziende_END_TAG);}
47
48  "<Azienda>"             {yybegin(AZIENDA); return symbol(sym.Azienda_START_TAG);}
49
50  {ws}                    {;}
51  {nl}                    {;}
52  .                       {;}
53
54  <AZIENDA>{
55
56          "</Azienda>"            {yybegin(YYINITIAL); return symbol(sym.Azienda_END_TAG);}
57
58          "<Sigla>"               {return symbol(sym.Sigla_START_TAG);}
59          "</Sigla>"              {return symbol(sym.Sigla_END_TAG);}
60
61          "<Nome>"                {return symbol(sym.Nome_START_TAG);}
62          "</Nome>"               {return symbol(sym.Nome_END_TAG);}
63
64          "<Sede>"                {return symbol(sym.Sede_START_TAG);}
65          "</Sede>"               {return symbol(sym.Sede_END_TAG);}
```

```
66
67          "<Impiegato>"           {yybegin(IMPIEGATO); return symbol(sym.Impiegato_START_TAG);}
68
69          {Numero}                {System.out.println("\t\t\t\tSCANNER␣-␣Numero:␣" + yytext()); return symbol(
                sym.NUMERO, new Integer(yytext()));}
70          {Parola}                {System.out.println("\t\t\t\tSCANNER␣-␣Parola:␣" + yytext()); return symbol(
                sym.PAROLA, new String(yytext()));}
71          {ParolaAlfaNum}         {System.out.println("\t\t\t\tSCANNER␣-␣ParolaAlfaNum:␣" + yytext()); return
                symbol(sym.PAROLANUM, new String(yytext()));}
72          {ParolaApostrofo}       {System.out.println("\t\t\t\tSCANNER␣-␣ParolaApostrofo:␣" + yytext());
                return symbol(sym.PAROLA_APOSTROFO, new String(yytext()));}
73          {ParolaNumApostrofo}    {System.out.println("\t\t\t\tSCANNER␣-␣ParolaNumApostrofo:␣" + yytext());
                return symbol(sym.PAROLA_NUM_APOSTROFO, new String(yytext()));}
74
75          {ws}                    {;}
76          {nl}                    {;}
77          .                       {;}
78  }
79
80  <IMPIEGATO>{
81
82          "</Impiegato>"          {yybegin(AZIENDA); return symbol(sym.Impiegato_END_TAG);}
83
84          "<CodiceFiscale>"       {return symbol(sym.CodiceFiscale_START_TAG);}
85          "</CodiceFiscale>"      {return symbol(sym.CodiceFiscale_END_TAG);}
86
87          "<Nome>"                {return symbol(sym.Nome_START_TAG);}
88          "</Nome>"               {return symbol(sym.Nome_END_TAG);}
89
90          "<DataNascita>"         {return symbol(sym.DataNascita_START_TAG);}
91          "</DataNascita>"        {return symbol(sym.DataNascita_END_TAG);}
92
93          "<Professione>"         {return symbol(sym.Professione_START_TAG);}
94          "</Professione>"        {return symbol(sym.Professione_END_TAG);}
95
96          "<Progetto>"            {yybegin(PROGETTO); return symbol(sym.Progetto_START_TAG);}
97
98          {Data}                  {System.out.println("\t\t\t\tSCANNER␣-␣Data:␣" + yytext()); return symbol(
                sym.DATA, new String(yytext()));}
99          {Numero}                {System.out.println("\t\t\t\tSCANNER␣-␣Numero:␣" + yytext()); return symbol(
                sym.NUMERO, new Integer(yytext()));}
100         {Parola}                {System.out.println("\t\t\t\tSCANNER␣-␣Parola:␣" + yytext()); return symbol(
                sym.PAROLA, new String(yytext()));}
101         {ParolaAlfaNum}         {System.out.println("\t\t\t\tSCANNER␣-␣ParolaAlfaNum:␣" + yytext()); return
                symbol(sym.PAROLANUM, new String(yytext()));}
102         {ParolaApostrofo}       {System.out.println("\t\t\t\tSCANNER␣-␣ParolaApostrofo:␣" + yytext());
                return symbol(sym.PAROLA_APOSTROFO, new String(yytext()));}
103
104         {ws}                    {;}
105         {nl}                    {;}
106         .                       {;}
107 }
108
109 <PROGETTO>{
110
111         "</Progetto>"           {yybegin(IMPIEGATO); return symbol(sym.Progetto_END_TAG);}
112
113         "<Codice>"              {return symbol(sym.Codice_START_TAG);}
114         "</Codice>"             {return symbol(sym.Codice_END_TAG);}
115
116         "<Nome>"                {return symbol(sym.Nome_START_TAG);}
117         "</Nome>"               {return symbol(sym.Nome_END_TAG);}
118
119         "<Descrizione>"         {yybegin(DESCRIZIONE); return symbol(sym.Descrizione_START_TAG);}
120
121         "<NumOre>"              {return symbol(sym.NumOre_START_TAG);}
122         "</NumOre>"             {return symbol(sym.NumOre_END_TAG);}
123
124         {Numero}                {System.out.println("\t\t\t\tSCANNER␣-␣Numero:␣" + yytext()); return symbol(
                sym.NUMERO, new Integer(yytext()));}
125         {Parola}                {System.out.println("\t\t\t\tSCANNER␣-␣Parola:␣" + yytext()); return symbol(
                sym.PAROLA, new String(yytext()));}
126         {ParolaAlfaNum}         {System.out.println("\t\t\t\tSCANNER␣-␣ParolaAlfaNum:␣" + yytext()); return
                symbol(sym.PAROLANUM, new String(yytext()));}
127         {ParolaApostrofo}       {System.out.println("\t\t\t\tSCANNER␣-␣ParolaApostrofo:␣" + yytext());
                return symbol(sym.PAROLA_APOSTROFO, new String(yytext()));}
128         {ParolaNumApostrofo}    {System.out.println("\t\t\t\tSCANNER␣-␣ParolaNumApostrofo:␣" + yytext());
                return symbol(sym.PAROLA_NUM_APOSTROFO, new String(yytext()));}
129
130         {ws}                    {;}
131         {nl}                    {;}
132         .                       {;}
133 }
134
135 <DESCRIZIONE>{
136
137         "</Descrizione>"        {yybegin(PROGETTO); return symbol(sym.Descrizione_END_TAG);}
138         [^< \t]+                {System.out.println("\t\t\t\tSCANNER␣-␣Altro:␣" + yytext()); return symbol(
                sym.ALTRO, new String(yytext()));}
139         {ws}                    {;}
140 }
```

A.2 Parser

```
1   import java_cup.runtime.*;
```

```
2   import java.util.*;
3   import java.io.*;
4   import javax.swing.*;
5   import java.awt.*;
6
7
8   init with {:
9           ElencoAziende = new HashMap();
10          azienda = new Azienda();
11          impiegato = new Impiegato();
12          progetto = new Progetto();
13  :};
14
15
16  parser code {:
17
18  /*
19          ******************************
20          ** CLASSI PER LA GRAMMATICA **
21          ******************************
22  */
23
24  public class Azienda {
25          String NomeAzienda;
26          String SedeAzienda;
27          String SiglaAzienda;
28          HashMap Impiegati;
29
30          public Azienda(){
31                  this.NomeAzienda = "";
32                  this.SedeAzienda = "";
33                  this.SiglaAzienda = "";
34                  this.Impiegati = new HashMap();
35          }
36
37          public Azienda(Azienda a){
38                  this.NomeAzienda = a.NomeAzienda;
39                  this.SedeAzienda = a.SedeAzienda;
40                  this.SiglaAzienda = a.SiglaAzienda;
41                  this.Impiegati = (HashMap)a.Impiegati.clone();
42          }
43
44          public void SetNomeAzienda(String nomeAzienda){
45                  this.NomeAzienda = nomeAzienda;
46          }
47
48          public void SetSedeAzienda(String sedeAzienda){
49                  this.SedeAzienda = sedeAzienda;
50          }
51
52          public void SetSiglaAzienda(String siglaAzienda){
53                  this.SiglaAzienda = siglaAzienda;
54          }
55
56          public void ResetAzienda(){
57                  this.NomeAzienda = "";
58                  this.SedeAzienda = "";
59                  this.SiglaAzienda = "";
60                  this.Impiegati.clear();
61          }
62
63          public void InsertImpiegato(Impiegato impiegato){
64                  Impiegato temp = new Impiegato(impiegato);
65                  this.Impiegati.put(temp.CodiceFiscale,temp);
66          }
67
68          public void StampaAzienda(){
69                  System.out.println("-----------------------------------");
70                  System.out.println("Nome␣Azienda:␣" + this.NomeAzienda);
71                  System.out.println("␣Sede␣Azienda:␣" + this.SedeAzienda);
72                  System.out.println("␣Sigla␣Azienda:␣" + this.SiglaAzienda);
73                  System.out.println("-----------------------------------");
74
75                  Collection key = this.Impiegati.keySet();
76                  Iterator i = key.iterator();
77
78                  while(i.hasNext())
79                  {
80                          String indice = (String)i.next();
81                          Impiegato temp = (Impiegato)this.Impiegati.get(indice);
82                          temp.StampaImpiegato();
83                  }
84
85          }
86  }
87
88
89  public class Impiegato {
90          String NomeImpiegato;
91          String CodiceFiscale;
92          String DataNascita;
93          String Professione;
94          HashMap Progetti;
95
96          public Impiegato(){
97                  this.NomeImpiegato = "";
98                  this.CodiceFiscale = "";
99                  this.DataNascita = "";
100                 this.Professione = "";
```

```
101                 this.Progetti = new HashMap();
102         }
103
104         public Impiegato(Impiegato i){
105                 this.NomeImpiegato = i.NomeImpiegato;
106                 this.CodiceFiscale = i.CodiceFiscale;
107                 this.DataNascita = i.DataNascita;
108                 this.Professione = i.Professione;
109                 this.Progetti = (HashMap)i.Progetti.clone();;
110         }
111
112         public void SetNomeImpiegato(String nomeImpiegato){
113                 this.NomeImpiegato = nomeImpiegato;
114         }
115
116         public void SetCodiceFiscale(String codiceFiscale){
117                 this.CodiceFiscale = codiceFiscale;
118         }
119
120         public void SetDataNascita(String dataNascita){
121                 this.DataNascita = dataNascita;
122         }
123
124         public void SetProfessione(String professione){
125                 this.Professione = professione;
126         }
127
128         public void ResetImpiegato(){
129                 this.NomeImpiegato = "";
130                 this.CodiceFiscale = "";
131                 this.DataNascita = "";
132                 this.Professione = "";
133                 this.Progetti.clear();
134         }
135
136         public void InsertProgetto(Progetto progetto){
137                 Progetto temp = new Progetto(progetto);
138                 this.Progetti.put(temp.CodiceProgetto,temp);
139         }
140
141         public void StampaImpiegato(){
142                 System.out.println("-------------------------------------");
143                 System.out.println("Nome␣Impiegato:␣" + this.NomeImpiegato);
144                 System.out.println("Codice␣Fiscale:␣" + this.CodiceFiscale);
145                 System.out.println("Data␣di␣Nascita:␣" + this.DataNascita);
146                 System.out.println("Professione:␣" + this.Professione);
147                 System.out.println("-------------------------------------");
148
149                 Collection key = this.Progetti.keySet();
150                 Iterator i = key.iterator();
151
152                 while(i.hasNext())
153                 {
154                         String indice = (String)i.next();
155                         Progetto temp = (Progetto)this.Progetti.get(indice);
156                         temp.StampaProgetto();
157                 }
158
159         }
160
161 }
162
163 public class Progetto {
164         String CodiceProgetto;
165         String NomeProgetto;
166         String Descrizione;
167         int NumOre;
168
169         public Progetto(){
170                 this.CodiceProgetto = "";
171                 this.NomeProgetto = "";
172                 this.Descrizione = "";
173                 this.NumOre = 0;
174         }
175
176         public Progetto(Progetto p){
177                 this.CodiceProgetto = p.CodiceProgetto;
178                 this.NomeProgetto = p.NomeProgetto;
179                 this.Descrizione = p.Descrizione;
180                 this.NumOre = p.NumOre;
181         }
182
183         public void SetCodiceProgetto(String codiceProgetto){
184                 this.CodiceProgetto = codiceProgetto;
185         }
186
187         public void SetNomeProgetto(String nomeProgetto){
188                 this.NomeProgetto = nomeProgetto;
189         }
190
191         public void SetDescrizione(String descrizione){
192                 this.Descrizione = descrizione;
193         }
194
195         public void SetNumOre(int numOre){
196                 this.NumOre = numOre;
197         }
198
199         public void ResetProgetto(){
```

```
200                 this.CodiceProgetto = "";
201                 this.NomeProgetto = "";
202                 this.Descrizione = "";
203                 this.NumOre = 0;
204         }
205
206         public void StampaProgetto(){
207                 System.out.println("---------------------------------");
208                 System.out.println("Codice:␣" + this.CodiceProgetto);
209                 System.out.println("Nome:␣" + this.NomeProgetto);
210                 System.out.println("Descrizione:␣" + this.Descrizione);
211                 System.out.println("Numero␣Ore:␣" + this.NumOre);
212                 System.out.println("---------------------------------");
213         }
214 }
215
216
217 /*
218         ******************************
219         **           FINE           **
220         ******************************
221 */
222
223 public HashMap ElencoAziende;
224 public Azienda azienda;
225 public Impiegato impiegato;
226 public Progetto progetto;
227
228 public void syntax_error(Symbol simbolo_attuale){
229         StringBuffer m = new StringBuffer("Errore");
230         if (cur_token.left != -1){
231                 m.append("␣in␣linea␣" + (simbolo_attuale.left+1));
232                 m.append(",␣colonna␣" + (simbolo_attuale.right+1));
233         }
234         m.append(",␣simbolo:␣" + (simbolo_attuale));
235         m.append("␣:␣Errore␣di␣Sintassi");
236         System.err.println(m);
237 }
238
239 public void InsertAzienda(Azienda a){
240         Azienda temp = new Azienda(a);
241         this.ElencoAziende.put(temp.NomeAzienda,temp); //SiglaAzienda
242 }
243
244
245 /*
246         ******************************
247         **     FUNZIONI PER HTML    **
248         ******************************
249 */
250
251 public void StampaHTML()
252 {
253         try{
254                 BufferedWriter out = new BufferedWriter(new FileWriter("output.html"));
255                 this.StampaTAGI(out);
256                 this.StampaListaAziende(out);
257                 this.StampaAziende(out);
258                 this.StampaTAGF(out);
259                 out.close();
260         } catch (IOException e){System.out.println("Errore␣nella␣scrittura␣su␣file");}
261         }
262
263 public void StampaTAGI(BufferedWriter out) throws IOException
264 {
265         out.write("<html><head><title>Report␣Aziende</title>");
266         out.write("<style␣type=\"text/css\">td{padding-left:␣5px;}</style>");
267         out.write("</head><body>\n");
268         out.write("<h1␣align=\"center\"><font␣face=\"Verdana\">Report␣Aziende␣e␣Progetti
269 </font></h1><h2␣align=\"justify\"><a␣name=\"top\"><font␣face=\"Verdana\">
270 Aziende</font></a></h2><ul>\n");
271 }
272
273 public void StampaTAGF(BufferedWriter out) throws IOException
274 {
275         out.write("</body></html>\n");
276 }
277
278 public void StampaListaAziende(BufferedWriter out) throws IOException
279 {
280         //le aziende non sono ordinate
281         Collection key = ElencoAziende.keySet();
282         for(Iterator i = key.iterator(); i.hasNext();)
283         {
284                 String nomeAzienda = (String)i.next();
285
286                 out.write("<li>\n");
287                 out.write("␣␣<p␣align=\"justify\"><font␣face=\"Verdana\">"+nomeAzienda+"
288 <font␣size=\"2\">\n");
289                 out.write("[<a␣href=\"#"+nomeAzienda+"\">informazioni</a>]
290 [<a␣href=\"#"+nomeAzienda+"_imp\">impiegati</a>]␣[<a␣href=\"#"+nomeAzienda+"_pro\">
291 progetti</a>]</font></font></li>\n");
292                 out.write("</li>\n");
293         }
294         out.write("</ul><p><font␣size=\"2\">[<a␣href=\"#top\">lista␣aziende</a>]</p>\n");
295
296 }
297
298
```

```
299 public void StampaAziende(BufferedWriter out) throws IOException
300 {
301         if(ElencoAziende != null)
302         {
303                 Collection keyaz = ElencoAziende.keySet();
304                 for(Iterator iaz = keyaz.iterator(); iaz.hasNext();)
305                 {
306                         HashMap progettiAziendali = new HashMap();
307                         String nomeAzienda = (String)iaz.next();
308                         Azienda aziendaObj = (Azienda)ElencoAziende.get(nomeAzienda);
309
310                         out.write("<h3><a␣name=\""+nomeAzienda+"\"><font␣face=\"Verdana\">
311 "+nomeAzienda+"</font></a></h3>\n");
312                         out.write("<p><font␣face=\"Verdana\"><b>Sigla:␣</b>"
313 +aziendaObj.SiglaAzienda+"</font></p>\n");
314                         out.write("<p><font␣face=\"Verdana\"><b>Sede:␣</b>"
315 +aziendaObj.SedeAzienda+"</font></p>\n");
316                         out.write("<p><b><a␣name=\""+nomeAzienda+"_imp\">
317 <font␣face=\"Verdana\">Impiegati</font></a></b></p>\n");
318                         out.write("<table␣border=\"1\"␣cellpadding=\"0\"␣cellspacing=\"0\"
319 style=\"border-collapse: collapse\"␣bordercolor=\"#111111\"␣width=\"100%\">\n");
320                         out.write("<tr>\n");
321                         out.write("<td␣width=\"22%\"␣bgcolor=\"#FFFFCC\">
322 <font␣face=\"Verdana\">Nome</font></td>\n");
323                         out.write("<td␣width=\"24%\"␣bgcolor=\"#FFFFCC\">
324 <font␣face=\"Verdana\">Professione</font></td>\n");
325                         out.write("<td␣width=\"54%\"␣bgcolor=\"#FFFFCC\">
326 <font␣face=\"Verdana\">Progetto/i</font></td>\n");
327                         out.write("</tr>\n");
328
329                         if(aziendaObj.Impiegati != null)
330                         {
331                                 Collection keyimp = aziendaObj.Impiegati.keySet();
332                                 for(Iterator iimp = keyimp.iterator(); iimp.hasNext();)
333                                 {
334                                         String codiceFis = (String)iimp.next();
335                                         Impiegato impiegatoObj = (Impiegato)aziendaObj.Impiegati.
336 get(codiceFis);
337
338                                         out.write("<tr>\n");
339                                         out.write("<td␣width=\"22%\"><font␣face=\"Verdana\"
340 size=\"2\"><b>"+impiegatoObj.NomeImpiegato+"</b></font><br>\n");
341                                         out.write("<font␣face=\"Verdana\"␣size=\"2\">"+codiceFis
342 +"</font><br>\n");
343                                         out.write("<font␣face=\"Verdana\"␣size=\"2\">"+impiegatoObj.
344 DataNascita+"</font></td>\n");
345                                         out.write("<td␣width=\"24%\"><font␣face=\"Verdana\"
346 size=\"2\">"+impiegatoObj.Professione+"</font></td>\n");
347                                         out.write("<td␣width=\"54%\">\n");
348                                         out.write("<ul>\n");
349
350                                         if(impiegatoObj.Progetti != null)
351                                         {
352                                                 Collection keyprog = impiegatoObj.Progetti.keySet();
353                                                 for(Iterator iprog = keyprog.iterator(); iprog.hasNext();)
354                                                 {
355                                                         String codiceProg = (String)iprog.next();
356                                                         Progetto progettoObj = (Progetto)impiegatoObj.
                                                                 Progetti.
357 get(codiceProg);
358                                                         if(!progettiAziendali.containsKey(codiceProg))
359                                                         {
360                                                                 Progetto progtmp = new Progetto();
361                                                                 progtmp.CodiceProgetto = progettoObj.
                                                                        CodiceProgetto;
362                                                                 progtmp.Descrizione = progettoObj.
                                                                        Descrizione;
363                                                                 progtmp.NomeProgetto = progettoObj.
                                                                        NomeProgetto;
364                                                                 progtmp.NumOre = progettoObj.NumOre;
365                                                                 progettiAziendali.put(codiceProg, progtmp);
366                                                         }
367                                                         else
368                                                         {
369                                                                 Progetto progtmp = new Progetto();
370                                                                 progtmp = (Progetto)progettiAziendali.get(
                                                                        codiceProg);
371                                                                 progtmp.NumOre = progtmp.NumOre+progettoObj.
                                                                        NumOre;
372                                                                 progettiAziendali.remove(codiceProg);
373                                                                 progettiAziendali.put(codiceProg, progtmp);
374                                                         }
375
376                                                         out.write("<li><font␣face=\"Verdana\"␣size=\"2\">"+
377 progettoObj.NomeProgetto+"␣[<a␣href=\"#"+aziendaObj.NomeAzienda+"_"+impiegatoObj.
378 CodiceFiscale+"_"+progettoObj.CodiceProgetto+"\">informazioni</a>]</font></li>\n");
379                                                 }
380                                         }
381
382                                         out.write("</ul>\n");
383                                         out.write("</td>\n");
384                                         out.write("</tr>\n");
385                                 }
386                         }
387                         out.write("</table>\n");
388                         out.write("<p><font␣size=\"2\">[<a␣href=\"#top\">lista␣aziende</a>]</font></p>\n");
389
390                         //progetti singolo impiegato
391                         if(aziendaObj.Impiegati != null)
```

```
                        {
                                Collection keyimp = aziendaObj.Impiegati.keySet();
                                for(Iterator iimp = keyimp.iterator(); iimp.hasNext();)
                                {
                                        String codiceFis = (String)iimp.next();
                                        Impiegato impiegatoObj = (Impiegato)aziendaObj.Impiegati.
get(codiceFis);

                                        out.write("<p><b><font␣face=\"Verdana\"␣color=\"#000080\">
Progetti␣impiegato␣"+impiegatoObj.NomeImpiegato+"</font></b></p>\n");
                                        out.write("<table␣border=\"1\"␣cellpadding=\"0\"␣cellspacing=\"0\"
style=\"border-collapse: collapse\"␣bordercolor=\"#111111\"␣width=\"100%\">\n");
                                        out.write("<tr>\n");
                                        out.write("<td␣width=\"12%\"␣bgcolor=\"#FFFFCC\">
<font␣face=\"Verdana\">Codice</font></td>\n");
                                        out.write("<td␣width=\"25%\"␣bgcolor=\"#FFFFCC\">
<font␣face=\"Verdana\">Nome</font></td>\n");
                                        out.write("<td␣width=\"52%\"␣bgcolor=\"#FFFFCC\">
<font␣face=\"Verdana\">Descrizione</font></td>\n");
                                        out.write("<td␣width=\"11%\"␣bgcolor=\"#FFFFCC\">
<font␣face=\"Verdana\">N.␣ore</font></td>\n");
                                        out.write("</tr>\n");

                                        if(impiegatoObj.Progetti != null)
                                        {
                                                Collection keyprog = impiegatoObj.Progetti.keySet();
                                                for(Iterator iprog = keyprog.iterator(); iprog.hasNext();)
                                                {
                                                        String codiceProg = (String)iprog.next();
                                                        Progetto progettoObj = (Progetto)impiegatoObj.
Progetti.get(codiceProg);

                                                        out.write("<tr>\n");
                                                        out.write("<td␣width=\"12%\"><font␣face=\"Verdana\"
size=\"2\">"+codiceProg+"</font></td>\n");
                                                        out.write("<td␣width=\"25%\"><font␣face=\"Verdana\"␣
                                                                size=\"2\">
<a␣name='"+nomeAzienda+"_"+impiegatoObj.CodiceFiscale+"_"+progettoObj.CodiceProgetto+"'>
"+progettoObj.NomeProgetto+"</a></font></td>\n");
                                                        out.write("<td␣width=\"52%\"><font␣face=\"Verdana\"
size=\"2\">"+progettoObj.Descrizione+"</font></td>\n");
                                                        out.write("<td␣width=\"11%\"><font␣face=\"Verdana\"
size=\"2\">"+progettoObj.NumOre+"</font></td>\n");
                                                        out.write("</tr>\n");
                                                }
                                                out.write("</table>\n");
                                        }
                                }
                        }
                        out.write("</table>\n");
                        out.write("<p><font␣size=\"2\">[<a␣href=\"#top\">lista␣aziende</a>]
</font></p>\n");


                        //progetti totali azienda
                        out.write("<p><b><a␣name=\""+nomeAzienda+"_pro\">
<font␣face=\"Verdana\">Progetti</a></font></b></p>\n");
                        out.write("<table␣border=\"1\"␣cellpadding=\"0\"␣cellspacing=\"0\"
style=\"border-collapse: collapse\"␣bordercolor=\"#111111\"␣width=\"100%\">\n");
                        out.write("<tr>\n");
                        out.write("<td␣width=\"25%\"␣bgcolor=\"#FFFFCC\">
<font␣face=\"Verdana\">Codice</font></td>\n");
                        out.write("<td␣width=\"50%\"␣bgcolor=\"#FFFFCC\">
<font␣face=\"Verdana\">Nome</font></td>\n");
                        out.write("<td␣width=\"25%\"␣bgcolor=\"#FFFFCC\">
<font␣face=\"Verdana\">N.␣ore</font></td>\n");
                        out.write("</tr>\n");


                        if(progettiAziendali != null)
                        {
                                Collection keyprogaz = progettiAziendali.keySet();
                                for(Iterator iprogaz = keyprogaz.iterator(); iprogaz.hasNext();)
                                {
                                        String codiceProg = (String)iprogaz.next();
                                        Progetto progettoObj = (Progetto)progettiAziendali.get(codiceProg);

                                        out.write("<tr>\n");
                                        out.write("<td␣width=\"25%\"><font␣face=\"Verdana\"
size=\"2\">"+codiceProg+"</font></td>\n");
                                        out.write("<td␣width=\"50%\"><font␣face=\"Verdana\"
size=\"2\"><a␣name='"+nomeAzienda+"_"+progettoObj.NomeProgetto+"'>"+progettoObj.
NomeProgetto+"</a></font></td>\n");
                                        out.write("<td␣width=\"25%\"><font␣face=\"Verdana\"␣size=\"2\">"
+progettoObj.NumOre+"</font></td>\n");
                                        out.write("</tr>\n");
                                }
                        }

                        out.write("</table>\n");
                        out.write("<p><font␣size=\"2\">[<a␣href=\"#top\">lista␣aziende</a>]
</font><p>\n");
                        out.write("<hr>");
                }
        }
}
```

```
490   /*
491           ******************************
492           **   FUNZIONI PER GRAFICO   **
493           ******************************
494   */
495
496
497   public int DisegnaBoxAzienda(Graphics g, Azienda a, int x, int y){
498           String nomeA = a.NomeAzienda;
499           String sedeA = a.SedeAzienda;
500           String siglaA = a.SiglaAzienda;
501           int lunghezza = 0;
502
503           if(nomeA.length()>sedeA.length())
504           {
505                   if(nomeA.length()>siglaA.length())
506                   {
507                           lunghezza = nomeA.length();
508                   }
509                   else lunghezza = siglaA.length();
510           }
511           else if(sedeA.length()>siglaA.length())
512           {
513                   lunghezza = sedeA.length();
514           }
515           else lunghezza = siglaA.length();
516
517
518           int x_width = lunghezza * 8;
519           int y_height = 60;
520
521           Font Tahoma_Plain = new Font("Tahoma", Font.PLAIN, 12);
522           Font Tahoma_Bold = new Font("Tahoma", Font.BOLD, 12);
523
524
525           g.setColor(new Color(185,244,128));
526           g.fillRect(x+1, y+1, x_width-2, y_height-2);
527           g.setColor(Color.BLACK);
528           g.drawRect(x, y, x_width, y_height);
529
530           g.setFont(Tahoma_Bold);
531           //g.setColor(Color.RED);
532           g.drawString(nomeA, x+5, y+15);
533
534           g.setFont(Tahoma_Plain);
535           g.setColor(Color.BLUE);
536           g.drawString(sedeA, x+5, y+35);
537
538           g.setColor(Color.BLUE);
539           g.drawString(siglaA, x+5, y+55);
540
541           return x_width;
542   }
543
544   public int DisegnaBoxImpiegato(Graphics g, Impiegato i,
545   int x, int y, int x_line, int y_line, boolean first, int nprog){
546           String nomeI = i.NomeImpiegato;
547           String codfisI = i.CodiceFiscale;
548           String dataI = i.DataNascita;
549           String profI = i.Professione;
550
551           int lunghezza = 0;
552
553           if(nomeI.length()>codfisI.length())
554           {
555                   if(nomeI.length()>profI.length())
556                   {
557                           lunghezza = nomeI.length();
558                   }
559                   else lunghezza = profI.length();
560           }
561           else if(codfisI.length()>profI.length())
562           {
563                   lunghezza = codfisI.length();
564           }
565           else lunghezza = profI.length();
566
567
568           int x_width = lunghezza * 7;
569           int y_height = 80;
570
571           Font Tahoma_Plain = new Font("Tahoma", Font.PLAIN, 12);
572           Font Tahoma_Bold = new Font("Tahoma", Font.BOLD, 12);
573
574
575           g.setColor(new Color(255,244,128));
576           g.fillRect(x+1, y+1, x_width-2, y_height-2);
577           g.setColor(Color.BLACK);
578           g.drawRect(x, y, x_width, y_height);
579
580           g.setFont(Tahoma_Bold);
581           g.setColor(Color.RED);
582           g.drawString(nomeI, x+5, y+15);
583
584           g.setFont(Tahoma_Plain);
585           g.setColor(Color.BLUE);
586           g.drawString(codfisI, x+5, y+35);
587
588           g.setColor(Color.BLUE);
```

```
589           g.drawString(dataI, x+5, y+55);
590
591           g.setColor(Color.BLUE);
592           g.drawString(profI, x+5, y+75);
593
594
595           if(first)
596           {
597                   g.setColor(Color.BLACK);
598                   g.drawLine(x_line, y_line, x_line+50, y_line);
599           }
600           else
601           {
602                   g.setColor(Color.BLACK);
603                   g.drawLine(x_line+25, y_line, x_line+50, y_line);
604                   g.drawLine(x_line+25, y_line, x_line+25, y_line-90*nprog);
605           }
606
607           return x_width;
608   }
609
610   public void DisegnaBoxProgetto(Graphics g, Progetto p, int x,
611   int y, int x_line, int y_line, boolean first){
612           String codP = p.CodiceProgetto;
613           String nomeP = p.NomeProgetto;
614           String descP = p.Descrizione;
615           String oreP = Integer.toString(p.NumOre) + "␣ore";
616
617           int lunghezza = 0;
618
619           if(codP.length()>nomeP.length())
620           {
621                   if(codP.length()>descP.length())
622                   {
623                           lunghezza = codP.length();
624                   }
625                   else lunghezza = descP.length();
626           }
627           else if(nomeP.length()>descP.length())
628           {
629                   lunghezza = nomeP.length();
630           }
631           else lunghezza = descP.length();
632
633           if(descP.length()>37)
634           {
635                   lunghezza = 40;
636                   descP = descP.substring(0,37) + "...";
637           }
638
639           int x_width = lunghezza * 7;
640           int y_height = 80;
641
642           Font Tahoma_Plain = new Font("Tahoma", Font.PLAIN, 12);
643           Font Tahoma_Bold = new Font("Tahoma", Font.BOLD, 12);
644           Font Tahoma_Italic = new Font("Tahoma", Font.ITALIC, 12);
645
646
647           g.setColor(new Color(255,244,128));
648           g.fillRect(x+1, y+1, x_width-2, y_height-2);
649           g.setColor(Color.BLACK);
650           g.drawRect(x, y, x_width, y_height);
651
652           g.setFont(Tahoma_Bold);
653           g.setColor(Color.RED);
654           g.drawString(nomeP, x+5, y+15);
655
656           g.setFont(Tahoma_Plain);
657           g.setColor(Color.BLUE);
658           g.drawString(codP, x+5, y+35);
659
660           g.setColor(Color.BLUE);
661           g.drawString(oreP, x+5, y+55);
662
663           g.setFont(Tahoma_Italic);
664           g.setColor(Color.BLUE);
665           g.drawString(descP, x+5, y+75);
666
667
668           if(first)
669           {
670                   g.setColor(Color.BLACK);
671                   g.drawLine(x_line, y_line, x_line+50, y_line);
672           }
673           else
674           {
675                   g.setColor(Color.BLACK);
676                   g.drawLine(x_line+25, y_line, x_line+50, y_line);
677                   g.drawLine(x_line+25, y_line, x_line+25, y_line-90);
678           }
679   }
680
681   public class Grafico extends JPanel{
682
683           public void paint(Graphics g){
684                   int current_x = 10;
685                   int current_y = 10;
686                   int current_x_line = 0;
687                   int current_y_line = 40;
```

```
688
689                 boolean first_impiegato;
690                 boolean first_progetto;
691
692                 int nprog = 1;
693
694                 g.setColor(new Color(191,207,207));
695                 g.fillRect(0, 0, 1500, 4000);
696
697                 int count = 0;
698
699                 Azienda a = new Azienda();
700                 HashMap impiegatiHash = new HashMap();
701
702                 Impiegato i = new Impiegato();
703                 HashMap progettiHash = new HashMap();
704
705                 Progetto p = new Progetto();
706
707                 Collection keyA;
708                 Collection keyI;
709                 Collection keyP;
710
711                 keyA = ElencoAziende.keySet();
712
713                 for(Iterator iA = keyA.iterator(); iA.hasNext();)
714                 {
715                         String indiceAzienda = (String)iA.next();
716                         a = (Azienda)ElencoAziende.get(indiceAzienda);
717
718                         first_impiegato = true;
719                         current_x = 10;
720                         current_y = 10 + count * 90;
721                         int offset = DisegnaBoxAzienda(g,a,current_x,current_y);
722
723                         impiegatiHash = (HashMap)a.Impiegati.clone();
724                         keyI = impiegatiHash.keySet();
725
726                         for(Iterator iI = keyI.iterator(); iI.hasNext();)
727                         {
728                                 String indiceImpiegato = (String)iI.next();
729                                 i = (Impiegato)impiegatiHash.get(indiceImpiegato);
730
731                                 first_progetto = true;
732                                 current_x = 10 + offset + 50;
733                                 current_y = 10 + count * 90;
734                                 current_x_line = 10 + offset;
735                                 current_y_line = 40 + count * 90;
736
737                                 int offset2 = DisegnaBoxImpiegato(g, i, current_x,
738   current_y, current_x_line, current_y_line, first_impiegato, nprog);
739
740                                 first_impiegato = false;
741                                 nprog = 0;
742
743                                 progettiHash = (HashMap)i.Progetti.clone();
744                                 keyP = progettiHash.keySet();
745
746                                 for(Iterator iP = keyP.iterator(); iP.hasNext();)
747                                 {
748                                         String indiceProgetto = (String)iP.next();
749                                         p = (Progetto)progettiHash.get(indiceProgetto);
750
751                                         current_x = 10 + offset + 50 + offset2 + 50;
752                                         current_y = 10 + count * 90;
753                                         current_x_line = 10 + offset + 50 + offset2;
754                                         current_y_line = 40 + count * 90;
755
756                                         DisegnaBoxProgetto(g, p, current_x, current_y,
757   current_x_line, current_y_line, first_progetto);
758
759                                         first_progetto = false;
760                                         count = count + 1;
761                                         nprog = nprog + 1;
762                                 }
763                         }
764                 }
765         }
766   }
767
768
769   public void StampaGrafico ()
770   {
771         Grafico grafico = new Grafico();
772         grafico.setPreferredSize(new Dimension(1500,4000));
773
774         try
775         { UIManager.setLookAndFeel(UIManager.
776   getSystemLookAndFeelClassName()); }
777         catch(Exception e){}
778
779         JFrame jf = new JFrame("Grafico␣delle␣Aziende");
780
781         JScrollPane jScrollPane1 = new javax.swing.JScrollPane();
782         jScrollPane1.setViewportView(grafico);
783         jScrollPane1.setHorizontalScrollBarPolicy(javax.swing.
784   JScrollPane.HORIZONTAL_SCROLLBAR_ALWAYS);
785         jScrollPane1.setVerticalScrollBarPolicy(javax.swing.
786   JScrollPane.VERTICAL_SCROLLBAR_ALWAYS);
```

```
787           jf.getContentPane().add(jScrollPane1);
788
789           jf.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
790           jf.setBounds(0,0,900,700);
791
792           jf.setVisible(true);
793   }
794
795
796
797
798
799   /*
800           ******************************
801           **           FINE           **
802           ******************************
803   */
804
805
806   public void StampaTutto(){
807           System.out.println("\n\n\n\n\n\n");
808           System.out.println("-------------------------------------");
809           System.out.println("-------------------------------------");
810           System.out.println("----------␣ELENCO␣AZIENDE␣----------");
811           System.out.println("-------------------------------------");
812           System.out.println("-------------------------------------");
813           System.out.println("\n\n");
814
815           Collection key = this.ElencoAziende.keySet();
816           Iterator i = key.iterator();
817
818           while(i.hasNext())
819           {
820                   String indice = (String)i.next();
821                   Azienda temp = (Azienda)this.ElencoAziende.get(indice);
822                   temp.StampaAzienda();
823           }
824
825   }
826   :};
827
828
829
830   // sezione dei Terminali / non Terminali
831
832   terminal ElencoAziende_START_TAG, ElencoAziende_END_TAG;
833   terminal Azienda_START_TAG, Azienda_END_TAG;
834   terminal Sigla_START_TAG, Sigla_END_TAG;
835   terminal Nome_START_TAG, Nome_END_TAG;
836   terminal Sede_START_TAG, Sede_END_TAG;
837   terminal Impiegato_START_TAG, Impiegato_END_TAG;
838   terminal CodiceFiscale_START_TAG, CodiceFiscale_END_TAG;
839   terminal DataNascita_START_TAG, DataNascita_END_TAG;
840   terminal Professione_START_TAG, Professione_END_TAG;
841   terminal Progetto_START_TAG, Progetto_END_TAG;
842   terminal Codice_START_TAG, Codice_END_TAG;
843   terminal Descrizione_START_TAG, Descrizione_END_TAG;
844   terminal NumOre_START_TAG, NumOre_END_TAG;
845   terminal NUMERO, PAROLANUM, DATA, PAROLA, ALTRO;
846   terminal PAROLA_APOSTROFO, PAROLA_NUM_APOSTROFO;
847
848   non terminal XML;
849   non terminal elenco_aziende, azienda, elenco_attributi_azienda,
850           attributi_azienda;
851   non terminal elenco_impiegati, impiegato, elenco_attributi_impiegato,
852           attributi_impiegato;
853   non terminal elenco_progetti, progetto, elenco_attributi_progetto,
854           attributi_progetto;
855
856   non terminal frase_parola_e_num, frase_parola, frase_parola_e_parolanum;
857   non terminal frase_tutto, frase_parola_e_parola_apo, altro;
858
859
860   start with XML;
861
862   XML ::= ElencoAziende_START_TAG elenco_aziende ElencoAziende_END_TAG
863   {:System.out.println("--------PARSER:␣TUTTO␣OK!--------");
864   parser.StampaTutto(); parser.StampaHTML(); parser.StampaGrafico(); :}
865   ;
866
867   elenco_aziende ::= elenco_aziende azienda
868           | azienda
869   ;
870
871   azienda ::= Azienda_START_TAG elenco_attributi_azienda Azienda_END_TAG
872   {:System.out.println("------PARSER:␣AZIENDA!------");
873   parser.InsertAzienda(parser.azienda); parser.azienda.ResetAzienda(); :}
874   ;
875
876   elenco_attributi_azienda ::=  elenco_attributi_azienda attributi_azienda
877           | attributi_azienda
878   ;
879
880   attributi_azienda ::= Sigla_START_TAG frase_parola_e_parolanum:sigla
881   Sigla_END_TAG   {: System.out.println("PARSER:␣SIGLA␣AZIENDA:␣" + sigla);
882   parser.azienda.SetSiglaAzienda((String)sigla); :}
883           | Nome_START_TAG frase_tutto:nome_azienda Nome_END_TAG
884   {: System.out.println("PARSER:␣NOME␣AZIENDA:␣" + nome_azienda);
885   parser.azienda.SetNomeAzienda((String)nome_azienda); :}
```

```
        | Sede_START_TAG frase_parola_e_parola_apo:sede Sede_END_TAG
{: System.out.println("PARSER:␣SEDE␣AZIENDA:␣" + sede);
parser.azienda.SetSedeAzienda((String)sede); :}
        | elenco_impiegati
{: System.out.println("---PARSER:␣IMPIEGATI---"); :}
;

elenco_impiegati ::= elenco_impiegati impiegato
        | impiegato
;

impiegato ::= Impiegato_START_TAG elenco_attributi_impiegato
Impiegato_END_TAG       {: System.out.println("----PARSER:␣IMPIEGATO!");
parser.azienda.InsertImpiegato(parser.impiegato);
parser.impiegato.ResetImpiegato(); :}
;

elenco_attributi_impiegato ::=  elenco_attributi_impiegato
attributi_impiegato
        | attributi_impiegato
;

attributi_impiegato ::= CodiceFiscale_START_TAG PAROLANUM:codice_fis
CodiceFiscale_END_TAG
{: System.out.println("PARSER:␣CODICE␣FISCALE:␣" + codice_fis);
parser.impiegato.SetCodiceFiscale((String)codice_fis); :}
        | Nome_START_TAG frase_parola_e_parola_apo:nome Nome_END_TAG
{: System.out.println("PARSER:␣NOME␣IMPIEGATO:␣" + nome);
parser.impiegato.SetNomeImpiegato((String)nome); :}
        | DataNascita_START_TAG DATA:data_nasc DataNascita_END_TAG
{: System.out.println("PARSER:␣DATA␣DI␣NASCITA:␣" + data_nasc);
parser.impiegato.SetDataNascita((String)data_nasc); :}
        | Professione_START_TAG frase_parola:prof Professione_END_TAG
{: System.out.println("PARSER:␣PROFESSIONE:␣" + prof);
parser.impiegato.SetProfessione((String)prof); :}
        | elenco_progetti
{: System.out.println("---PARSER:␣PROGETTI---"); :}
;

elenco_progetti ::= elenco_progetti progetto
        | progetto
;

progetto ::= Progetto_START_TAG elenco_attributi_progetto
Progetto_END_TAG        {: System.out.println("--␣PARSER:␣PROGETTO!");
parser.impiegato.InsertProgetto(parser.progetto);
parser.progetto.ResetProgetto(); :}
;


elenco_attributi_progetto ::= elenco_attributi_progetto
attributi_progetto
        | attributi_progetto
;

attributi_progetto ::= Codice_START_TAG PAROLANUM:cod_prog
Codice_END_TAG  {: System.out.println("PARSER:␣CODICE␣PROGETTO:␣" + cod_prog);
parser.progetto.SetCodiceProgetto((String)cod_prog); :}
        | Nome_START_TAG frase_parola_e_num:nome_prog Nome_END_TAG
{: System.out.println("PARSER:␣NOME␣PROGETTO:␣" + nome_prog);
parser.progetto.SetNomeProgetto((String)nome_prog); :}
        | Descrizione_START_TAG altro:desc_prog Descrizione_END_TAG
{: System.out.println("PARSER:␣DESCRIZIONE␣PROGETTO:␣" + desc_prog);
parser.progetto.SetDescrizione((String)desc_prog); :}
        | NumOre_START_TAG NUMERO:no_prog NumOre_END_TAG
{: System.out.println("PARSER:␣NUMERO␣ORE:␣" + no_prog);
parser.progetto.SetNumOre(((Integer)no_prog).intValue()); :}
;

frase_parola_e_num ::= frase_parola_e_num:a NUMERO:b {: RESULT = a + "␣" + b; :}
        | frase_parola_e_num:a PAROLA:b {: RESULT = a + "␣" + b; :}
        | PAROLA:a                                      {: RESULT = a; :}
        | NUMERO:a                                      {: RESULT = a; :}
;

frase_parola ::= frase_parola:a PAROLA:b        {: RESULT = a + "␣" + b; :}
        | PAROLA:a                              {: RESULT = a; :}
;


frase_parola_e_parolanum ::= frase_parola_e_parolanum:a PAROLANUM:b
{: RESULT = a + "␣" + b; :}
        | frase_parola_e_parolanum:a PAROLA:b                           {: RESULT = a + "␣" + b; :}
        | PAROLA:a                                                      {: RESULT = a; :}
        | PAROLANUM:a                                                   {: RESULT = a; :}
;

frase_tutto ::= frase_tutto:a PAROLANUM:b       {: RESULT = a + "␣" + b; :}
        | frase_tutto:a PAROLA:b                {: RESULT = a + "␣" + b; :}
        | frase_tutto:a PAROLA_APOSTROFO:b      {: RESULT = a + "␣" + b; :}
        | frase_tutto:a PAROLA_NUM_APOSTROFO:b  {: RESULT = a + "␣" + b; :}
        | frase_tutto:a NUMERO:b                {: RESULT = a + "␣" + b; :}
        | PAROLA:a                              {: RESULT = a; :}
        | PAROLANUM:a                           {: RESULT = a; :}
        | PAROLA_APOSTROFO:a                    {: RESULT = a; :}
        | PAROLA_NUM_APOSTROFO:a                {: RESULT = a; :}
        | NUMERO:a                              {: RESULT = a; :}
;
```

```
985   frase_parola_e_parola_apo::= frase_parola_e_parola_apo:a PAROLA:b
986   {: RESULT = a + "␣" + b; :}
987           | frase_parola_e_parola_apo:a PAROLA_APOSTROFO:b
988   {: RESULT = a + "␣" + b; :}
989           | PAROLA:a                                                    {: RESULT = a; :}
990           | PAROLA_APOSTROFO:a                                          {: RESULT = a; :}
991   ;
992
993   altro ::= altro:a ALTRO:b       {: RESULT = a + "␣" + b; :}
994           | ALTRO:a               {: RESULT = a; :}
995   ;
```