1 Abstract

L' applicativo opera nel contesto della steganografia testuale. L'intento è quello di realizzare una Java Applet che dato un testo contenitore e un dataset di parole chiave, estrapola dal testo contenitore tutti i possibili messaggi ottenuti applicando le regole steganografiche conosciute, quindi cerca se sono presenti una o più parole chiave in ciascun messaggio nascosto. L'idea base è che la scrittura del testo contenitore possa essere fatta solo da un essere umano, in quanto una macchina non è in grado di scrivere frasi di senso compiuto e contestualizzato. Per cui si ipotizza che l'algoritmo steganografico sia una regola di facile comprensione umana e.g estrapolare la 2 lettera di ogni parola. A questo punto non è dato sapere se il testo sotto esame contiene messaggi nascosti, e anche in caso li contenga non è noto quale tecnica steganografica limitatamente a quelle note all'applicativo sia stata usata per nascondere il messaggio. Per sapere se il messaggio estrapolato è significativamente rilevante, l'applicativo confronta la sequenza delle lettere che compongono il messaggio con un dataset di parole chiave. Quando una regola steganografica trova almeno una parola chiave, sia le parole trovate che il messaggio nascosto che le contiene sono salvati su uno stack e stampati a video.

Il funzionamento dell'applicazione può essere partizionato in tre parti:

1. Acquisizione di un dataset di parole chiave e creazione di un albero come mostrato in fig
2. Parsing del testo ed applicazione delle strategie steganografiche
3. Confronto della sequenza di lettere estrapolate dal testo con l'albero dataset.

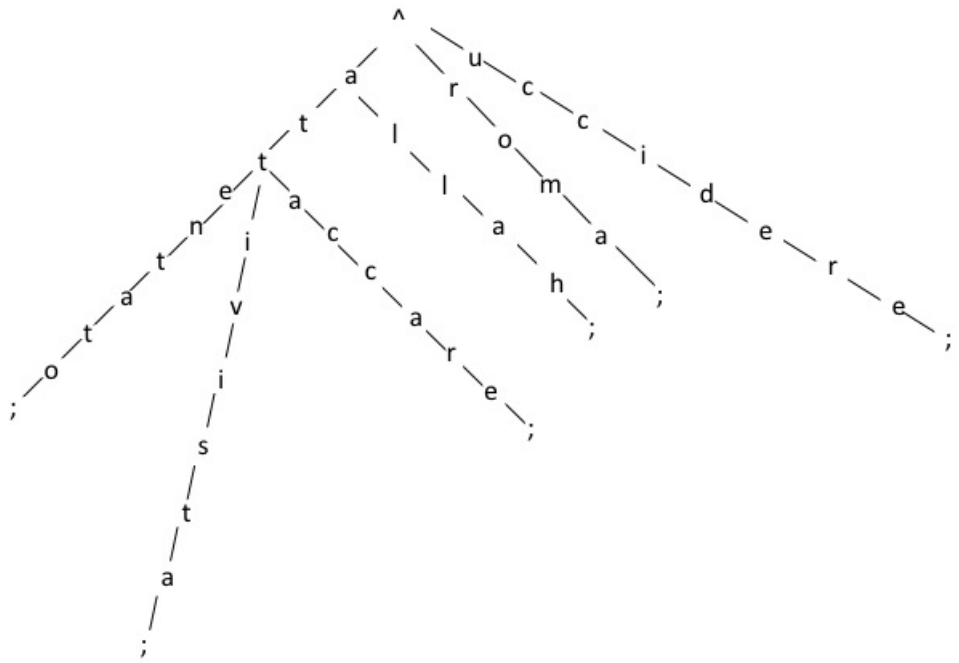
2 Casi d'uso

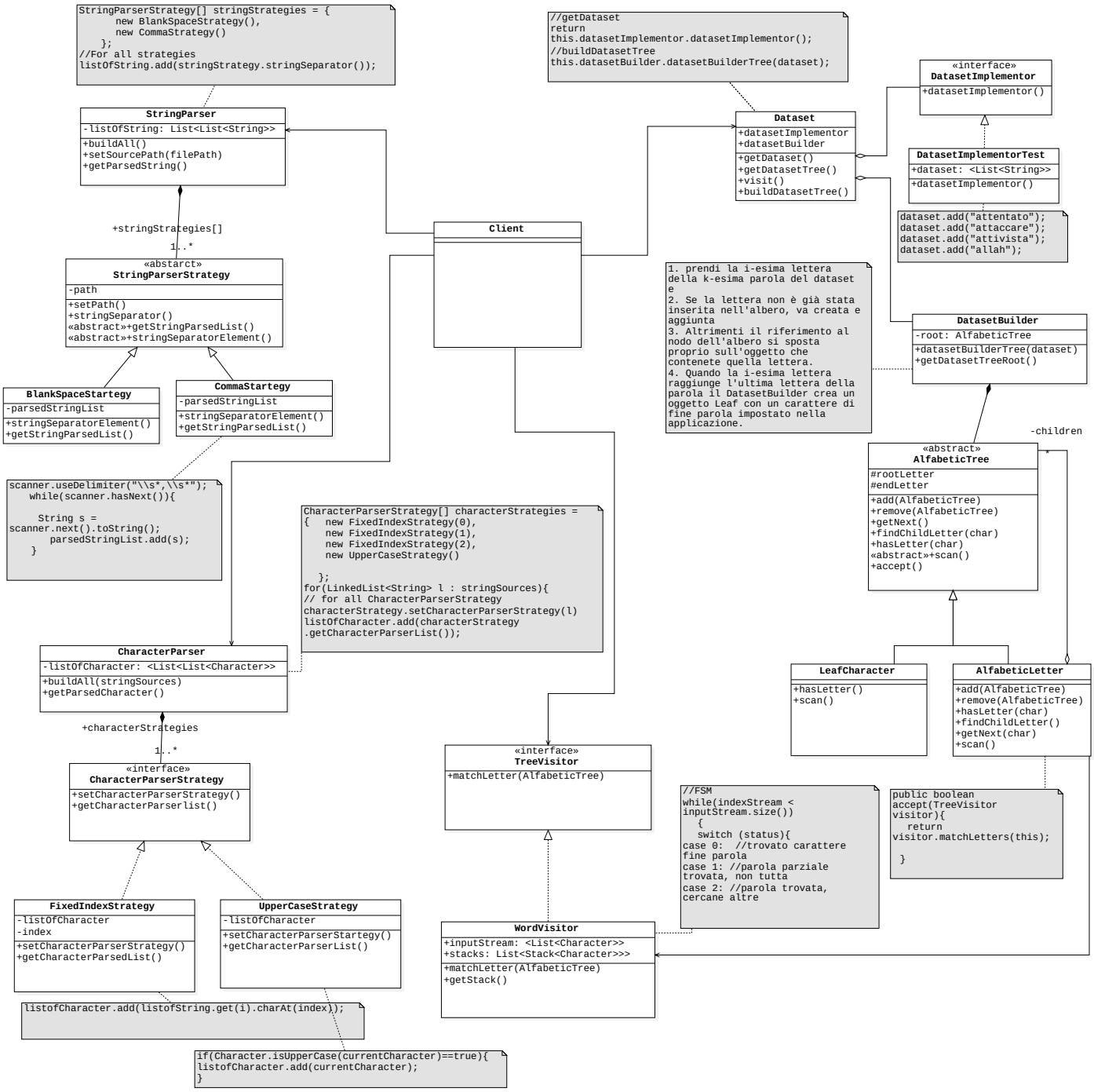
"Cameriere italiano o straniero lavoratore occasionale eccellente lavora orario festivo. Organizzando mostre umanistiche, matrimoni. Occasionalmente volantinaggio in Abruzzo, Lodi, Empoli, Ibiza e Parma.
Messo curriculum accademico, scrivere richiesta adeguata per credenziali telematiche".

"Vendesi, attico in zona, terzolle, tramite agenzia, esperta in, negoziazioni immobiliari, transizioni di capitale, atti di vendità, trasferimenti di proprietà, organizzazione di impresa".

3 Implementazione

Si consideri a titolo di esempio le parole attentato, attaccare, attivista, allah, roma e uccidere. Lo scopo è quello di costruire una struttura ad albero come mostrata in figura. La sequenza delle lettere estrapolate dal testo viene scandita e confrontata con l'albero dataset in modo che quando nella sequenza in ingresso è presente una parola del dataset, l'albero venga navigato lungo tutta quella parola fino a trovare l'elemento foglia, qui indicato con il carattere ' ; '.




StringParserStrategy[] stringStrategies = {
new BlankSpaceStrategy(),
new CommaStrategy()
};
//For all strategies
listOfString.add(stringStrategy.stringSeparator());
StringParser
-listOfString: List<List<String>>
+buildAll()
+setSourcePath(filePath)
+getParsedString()
+stringStrategies[]
1..*
«abstarct»
StringParserStrategy
-path
+setPath()
+stringSeparator()
«abstract»+getStringParsedList()
«abstract»+stringSeparatorElement()
BlankSpaceStartegy
-parsedStringList
+stringSeparatorElement()
+getStringParsedList()
CommaStartegy
-parsedStringList
+stringSeparatorElement()
+getStringParsedList()
scanner.useDelimiter("\\s*,\\s*");
while(scanner.hasNext()){
String s =
scanner.next().toString();
parsedStringList.add(s);
}
CharacterParserStrategy[] characterStrategies =
{ new FixedIndexStrategy(0),
new FixedIndexStrategy(1),
new FixedIndexStrategy(2),
new UpperCaseStrategy()
};
for(LinkedList<String> l : stringSources){
// for all CharacterParserStrategy
characterStrategy.setCharacterParserStrategy(l)
listOfCharacter.add(characterStrategy
.getCharacterParserList());
CharacterParser
-listOfCharacter: <List<List<Character>>
+buildAll(stringSources)
+getParsedCharacter()
+characterStrategies
1..*
«interface»
CharacterParserStrategy
+setCharacterParserStrategy()
+getCharacterParserlist()
FixedIndexStrategy
-listOfCharacter
-index
+setCharacterParserStrategy()
+getCharacterParsedList()
UpperCaseStrategy
-listOfCharacter
+setCharacterParserStartegy()
+getCharacterParserList()
listofCharacter.add(listofString.get(i).charAt(index));
if(Character.isUpperCase(currentCharacter)==true){
listofCharacter.add(currentCharacter);
}
Client
«interface»
TreeVisitor
+matchLetter(AlfabeticTree)
WordVisitor
+inputStream: <List<Character>>
+stacks: List<Stack<Character>>>
+matchLetter(AlfabeticTree)
+getStack()
//FSM
while(indexStream <
inputStream.size())
{
switch (status){
case 0: //trovato carattere
fine parola
case 1: //parola parziale
trovata, non tutta
case 2: //parola trovata,
cercane altre
//getDataset
return
this.datasetImplementor.datasetImplementor();
//buildDatasetTree
this.datasetBuilder.datasetBuilderTree(dataset);
Dataset
+datasetImplementor
+datasetBuilder
+getDataset()
+getDatasetTree()
+visit()
+buildDatasetTree()
«interface»
DatasetImplementor
+datasetImplementor()
DatasetImplementorTest
+dataset: <List<String>>
+datasetImplementor()
dataset.add("attentato");
dataset.add("attaccare");
dataset.add("attivista");
dataset.add("allah");
1. prendi la i-esima lettera
della k-esima parola del dataset
e
2. Se la lettera non è già stata
inserita nell'albero, va creata e
aggiunta
3. Altrimenti il riferimento al
nodo dell'albero si sposta
proprio sull'oggetto che
contenete quella lettera.
4. Quando la i-esima lettera
raggiunge l'ultima lettera della
parola il DatasetBuilder crea un
oggetto Leaf con un carattere di
fine parola impostato nella
applicazione.
DatasetBuilder
-root: AlfabeticTree
+datasetBuilderTree(dataset)
+getDatasetTreeRoot()
-children
*
«abstract»
AlfabeticTree
#rootLetter
#endLetter
+add(AlfabeticTree)
+remove(AlfabeticTree)
+getNext()
+findChildLetter(char)
+hasLetter(char)
«abstract»+scan()
+accept()
LeafCharacter
+hasLetter()
+scan()
AlfabeticLetter
+add(AlfabeticTree)
+remove(AlfabeticTree)
+hasLetter(char)
+findChildLetter()
+getNext(char)
+scan()
public boolean
accept(TreeVisitor
visitor){
return
visitor.matchLetters(this);
}

4 Parsing Strategy

Lo scopo delle strategie steganografiche è quello di prendere un testo ed estrapolare le lettere che formano un messaggio nascosto. Tenendo presente che la strategia steganografica deve essere umanamente comprensibile, si può pensare di suddividere un testo in sotto-blocchi con un criterio di selezione e da ciascun sotto-blocco selezionare una lettera con un altro criterio di selezione. Per esempio, si può suddividere un testo in parole, che equivale ad usare come criterio di selezione gli spazi bianchi tra le parole e poi su ogni parola selezionare la seconda lettera, usando quindi come criterio di selezione quello di prendere tutte le lettere in una data posizione. Un altro esempio è quello di separare il testo usando la punteggiatura, suddividendo il testo in frammenti di testo compresi tra virgole e poi andare a selezionare la prima lettera disponibile subito dopo una virgola. Verranno messe in vita due strategie:

1. StringParser che mette in vita le strategie BlankSpacesStrategy e CommaStrategy per selezionare dal testo le stringhe
2. CharacterParser che mette in vita le strategie FixedIndexStrategy e UpperCaseStrategy per selezionare dalle stringhe precedentemente parsate le lettere che compongono il messaggio nascosto

perché usare il pattern strategy il pattern strategy permette di incapsulare le algoritmi di parsing sotto un unica interfaccia, quindi eseguire tutte le strategie implementate.

```
import java.util.LinkedList;
/*
 * @StringParser has the responsibility to decide
    which concrete strategies are used for parsing
    strings.
 * @buildAll is used to create strategies for parsing
    string, and apply those strategy.
 * @getParsedString return a list of all list of
    string parsed with each strategy.
 *
 * @Note: StringParser act as 'Context' in the pattern
     Strategy.
 */
public class StringParser {
    LinkedList<LinkedList<String>> listOfString = new
       LinkedList<LinkedList<String>>();

    public StringParser() {
    }

    public void setSourcePath(String filePath){

        StringParserStrategy.setPath(filePath);
    }

```

```
    public  void buildAll(){
        StringParserStrategy[] stringStrategies = {
                    new BlankSpaceStrategy(),
                    new CommaStrategy()
                };
        for(StringParserStrategy stringStrategy :
            stringStrategies){

            listOfString.add(stringStrategy.
                stringSeparator());
        }

    }
    public LinkedList<LinkedList<String>>
        getParsedString(){
        return listOfString;
    }

}
```

```
import java.io.IOException;
import java.nio.file.Path;
import java.nio.file.Paths;
import java.util.LinkedList;
import java.util.Scanner;
/*
 * @StringParserStrategy define an abstract class for
    concrete string strategies.
 *                      It uses abstract class to
    favor reuse of scanner code.
 */
public abstract class StringParserStrategy {

    private static Path path;

    public static void setPath(String fileName){
        StringParserStrategy.path = Paths.get(fileName
            );
    }

    public  LinkedList<String> stringSeparator(){

        Scanner scanner = null;
        try {
            scanner = new Scanner(path);
        } catch (IOException e) {

            e.printStackTrace();
        }
        stringSeparatorElement(scanner);
```

```
29          scanner.close();
            return getStringParsedList();
31      }

33      protected abstract void stringSeparatorElement(
            Scanner scanner);
        public abstract  LinkedList<String>
            getStringParsedList();
35
}

import java.util.LinkedList;
2 import java.util.Scanner;
/*
4  * @BlankSpaceStrategy  has the responsibility to
       separate each string using blank spaces,
   *                      or in other word to pick every
        word in the text.
6  * @stringSeparatorElement  uses the scanner Java
       utility to separe words and
   *                          add those word to a list
       of string.
8  * @getStringParsedList return the list of string
       parsed from text with the blank strategy.
   */
10 public class BlankSpaceStrategy extends
       StringParserStrategy{
        private LinkedList <String> parsedStringList=new
            LinkedList<String>();
12      public BlankSpaceStrategy() {

14      }
        @Override
16      protected void stringSeparatorElement(Scanner
            scanner) {

18              scanner.useDelimiter("\\p{javaWhitespace}+")
                    ;
                while(scanner.hasNext()){
20
                      String s = scanner.next().toString();
22                      parsedStringList.add(s);
                }
24      }

26      @Override
        public LinkedList<String> getStringParsedList() {
28
            return this.parsedStringList;
```

```
30      }

32  }
```

```
import java.util.LinkedList;
2   import java.util.Scanner;
    /*
4    * @CommaStrategy   has the responsibility to separete
         each string using comma.
     *                  This is useful in case a comma is
        taken as reference to find the hidden letter.
6    * @stringSeparatorElement  is the method that parse
        the text and return pieces of text between commas.
     * @getStringParsedList is the method that return the
        list of string.
8    */
    public class CommaStrategy extends
       StringParserStrategy{
10      private LinkedList <String> parsedStringList=new
           LinkedList<String>();
        public CommaStrategy() {
12      }

14      @Override
        public void stringSeparatorElement(Scanner scanner
           ) {
16
              scanner.useDelimiter("\\s*,\\s*");
18            while(scanner.hasNext()){

20                  String s = scanner.next().toString();
                      parsedStringList.add(s);
22              }
        }
24      @Override
        public LinkedList<String> getStringParsedList() {
26
            return this.parsedStringList;
28      }

30  }
```

```
import java.util.LinkedList;
/*
 * @CharacterParser has the responsibility to decide
    which concrete strategies are used for parsing
    character.
 * @buildAll is used to create strategies for parsing
    characters, and apply those strategy.
 * @getParsedCharacter return a list of all list of
    characters parsed with each strategy.
 *
 * @Note: CharacterParser act as 'Context' in the
    pattern Strategy.
 */
public class CharacterParser {
    private LinkedList<LinkedList<Character>>
       listOfCharacter = new LinkedList<LinkedList<
       Character>>();
    //public CharacterParserStrategy charstrategy;

    public CharacterParser() {
    }

    public void buildAll(LinkedList<LinkedList<String
       >> stringSources){

        CharacterParserStrategy[] characterStrategies
           = {
            new FixedIndexStrategy(0),
            new FixedIndexStrategy(1),
            new FixedIndexStrategy(2),
            new UpperCaseStrategy()

        };
        for(LinkedList<String> l : stringSources){

            for(CharacterParserStrategy
               characterStrategy : characterStrategies
               ){

                characterStrategy.
                   setCharacterParserStrategy(l);
                listOfCharacter.add(characterStrategy.
                   getCharacterParserList());
            }
        }
    }

    LinkedList<LinkedList<Character>>
       getParsedCharacter(){
         return listOfCharacter;
```

```
        }
38
}
```

```
1  import java.util.LinkedList;
   /*
3   * @CharacterParserStrategy has the responsibility to
        implements all methods of concrete character
        strategy.
    *
5   * @Note CharacterParserStrategy  is the 'Strategy' of
        the Strategy pattern.
    */
7  public interface CharacterParserStrategy {

9

       public void setCharacterParserStrategy(LinkedList<
          String> listofString);
11

       public LinkedList<Character>
          getCharacterParserList();
13

15 }
```

```
1  import java.util.LinkedList;
   /*
3   * @FixedIndexStrategy has the responsibility to apply
        the parsing strategy
    *                     that pick each letter at a
       given index of every string
5   * @setParserCharacterStrategy is the method that
       receive in input a list of string
    *  and give in output a list of character
7   *  @getCharacterParsedList is the method used to
       access the parsed list of character
    */
9  public class FixedIndexStrategy implements
      CharacterParserStrategy{

11     private LinkedList <Character> listofCharacter;
       private int index;
13
       public FixedIndexStrategy( int index) {
15         this.index=index;
       }
17
       public void setCharacterParserStrategy(LinkedList<
          String> listofString) {
19
```

```
            listofCharacter = new LinkedList <Character>()
                ; /// serve per poter distingure le liste
                qundo uno stessa istanza di fixedindex
                visita prima Blank strategy e poi comma.
21
                    for(int i=0; i<listofString.size(); i
                        ++){
23
                            if(listofString.get(i).length
                                () > index) { //la
                                lunghezza  della parola
                                deve essre maggiore dell'
                                index
25
                                listofCharacter.add(
                                    listofString.get(i).
                                    charAt(index));
27
                            }
29                  }
        }
31
        public LinkedList<Character>
            getCharacterParserList() {
33          // TODO Auto-generated method stub
            return listofCharacter;
35      }

37


39  }
```

```
    import java.util.LinkedList;
2   /*
     * @UpperCaseStrategy   has the responsibility to
        apply the parsing strategy
4    *                      that pick every UpperCase
        letter of every string.
     * @setCharacterParserStrategy  is the method that
        receive in input a list of string
6    *                              and give on output a
        list of character.
     * @getCharacterParserList      is the method used to
        return the list of character
8    */
    public class UpperCaseStrategy implements
        CharacterParserStrategy{
10      private LinkedList <Character> listofCharacter;

12      public UpperCaseStrategy() {
```

```
14      }

16      public void setCharacterParserStrategy(LinkedList<
            String> listofString) {
            listofCharacter = new LinkedList <Character>()
                ;
18
            String currentString;
20          Character currentCharacter;
            for(int i=0; i<listofString.size(); i++){
22              currentString =listofString.get(i);
                for(int l=0;l< currentString.length(); l
                    ++){
24                  currentCharacter = currentString.
                        charAt(l);
                    if(Character.isUpperCase(
                        currentCharacter)==true){
26                      currentCharacter=Character.
                            toLowerCase(currentCharacter.
                            charValue());
                        listofCharacter.add(
                            currentCharacter);
28                  }
                }
30
            }
32
        }
34
        public LinkedList<Character>
            getCharacterParserList() {
36          // TODO Auto-generated method stub
            return listofCharacter;
38      }

40  }
```

5 Dataset Implementor

Lo scopo del datasetImplementor è quello di realizzare un bridge per astrarre l'operazione di dump delle stringhe appartenenti al dataset rispetto a come queste vengano implementate; Potrebbero essere implementate tramite un database MySQL o trovarsi in file XML o csv. Per mostrare l'intento è stata realizzata una classe DatasetImplementorTest. Qualunque implementazione venga usata, il metodo datasetImplementor restituisce una lista di stringhe.

```
import java.util.LinkedList;
2   /*
```

```
 * @DatasetImplementor is the interface of concrete Classes
 */
public interface DatasetImplementor {

    public LinkedList <String> datasetImplementor();

}
```

```
import java.util.LinkedList;
/*
 * @DatasetImplementorTest  is a proof of concept instance that aims to
 *                          demonstrate the dump of a dataset.
 *                          The dataset could be implemented as MySQL or csv file or whatever.
 * @datasetImplementor  is a method that dump a set of words and add those words in the list dataset.
 *
 * @Note DatasetImplementor act as the 'ConcreteImplementation' on the bridge patter.
 */
public class DatasetImplementorTest implements DatasetImplementor{

    LinkedList <String> dataset = new LinkedList <String>();

    public DatasetImplementorTest() {
    }

    @Override
    public LinkedList<String> datasetImplementor() {

        dataset.add("attentato");
        dataset.add("attaccare");
        dataset.add("attivista");
        dataset.add("allah");
        dataset.add("spacciare");
        dataset.add("uccidere");
        dataset.add("bomba");
        dataset.add("roma");
        dataset.add("ingegneria");
        dataset.add("del");
        dataset.add("software");

        dataset.add("dog");
        dataset.add("cat");
        return dataset;
```

```
35      }

37  }
```

6 Dataset Builder

Lo scopo del DatasetBuilder è quello di realizzare l'albero che come illustrato in fig vedi sopra rappresenta tutte le lettere delle parole appartenenti al dataset. DatasetBuilder inserisce le lettere delle parole del dataset a partire da una radice individuata da un carattere speciale noto all'applicazione. Questo espediente è reso necessario dal fatto che le parole del dataset iniziano con lettere diverse a cui è necessario dare un carattere genitore. L'albero viene costruito semplicemente aggiungendo le lettere delle parole come figli del nodo rootelement all'inizio e poi via via scorrendo l'albero ogni volta che una lettera è già stata inserita problema delle radici di parole comuni. Quando viene raggiunta l'ultima lettera di ogni parola il DatasetBuilder provvede a mettere in vita un Leaf che contiene un carattere speciale che indica la fine di una parola sul albero.

```
1   import java.util.LinkedList;

3   public class DatasetBuilder {
        private AlfabeticTree root;
5
        public DatasetBuilder() {
7
            root=null;
9       }
        public AlfabeticTree getDatasetTreeRoot(){
11          return this.root;
        }
13      public void datasetBuilderTree(LinkedList <String>
            dataset){
            this.root= new AlfabeticLetter('0');
15          AlfabeticTree index=root;
            char c;
17          for(int i=0; i < dataset.size(); i++){
                for(int k=0; k<=dataset.get(i).length(); k
                    ++){
19                  /*prendi ciascuna lettera di ogni
                        Stringa*/

21                  /*aggiungi in fondo a ogni parola un
                        volore di end word. Questo e il
                        Leaf dell albero */
                    if(k==dataset.get(i).length()){
23                  c = AlfabeticTree.endLetter;
                    index.add(new LeafCharacter(c));
25                  }
```

```
                    else {/* prendi la i-esima lettera
                       della k-esima parola del dataset */
27                  c = dataset.get(i).charAt(k);
                    }
29                  /* Se la lettera non e gia stata
                       inserita nell albero va creata e
                       aggiunta */
                    if(index.hasLetter(c)==false){
31                      index.add(new AlfabeticLetter(c));
                        index = index.findChildLetter(c);
33                  }
                    /* altrimenti il riferimento si sposta
                        propio sull oggetto contenete
                       quella lettera*/
35                  else {
                        index = index.findChildLetter(c);

37                  }


39
                }
41              index = this.root;
            }
43


45      }
    }
```

7 Dataset

Lo scopo di Dataset è quello di fornire al Client i metodi per delegare al DatasetImplementor il dump del dataset e al DatasetBuilder la costruzione dell'albero.

```
    import java.util.LinkedList;
2   /*
     * @Dataset has two responsibility , first dump the
        list of string hosted somewhere
4    *                       second, build a composite tree
         structure using letter found on text.
     *@getDatasetTree        ritorna la radice dell'albero
        creato
6    *@Visita
     */
8   public class Dataset{

10      DatasetImplementor datasetImplementor;
        DatasetBuilder datasetBuilder;
```

```
12      public Dataset(DatasetImplementor
            datasetImplementor, DatasetBuilder
            datasetBuilder){
            this.datasetImplementor = datasetImplementor;
14          this.datasetBuilder=datasetBuilder;
        }
16
        public LinkedList <String> getDataset(){
18          return this.datasetImplementor.
                datasetImplementor();
        }
20
        public AlfabeticTree getDatasetTree(){
22          return this.datasetBuilder.getDatasetTreeRoot
                ();
        }
24
        public void visit(){
26
            if(this.datasetBuilder.getDatasetTreeRoot()==
                null){
28              throw new Error("Albero non creato. ");
            }
30          this.datasetBuilder.getDatasetTreeRoot().scan
                ();
        }
32
        public void buildDatasetTree(LinkedList <String>
            dataset) {
34
            this.datasetBuilder.datasetBuilderTree(dataset
                );
36
        }
38
    }
```

8 Dataset tree

```
1


3   public abstract class AlfabeticTree {
        char letter;
5       static protected char rootLetter = '0';
        static protected char endLetter = ';';
7


9       public boolean accept(TreeVisitor visitor){
```

```
            return visitor.matchLetters(this);
    };
    public abstract void scan();
    public AlfabeticTree getNext(int i){
        throw new UnsupportedOperationException("
            Operation not supported");
    }
    public AlfabeticTree findChildLetter(char c){
        throw new UnsupportedOperationException("
            Operation not supported");
    }
    public boolean hasLetter(char c){
        throw new UnsupportedOperationException("
            Operation not supported");
    }
    public void add(AlfabeticTree node){
        throw new UnsupportedOperationException("
            Operation not supported");
    };
    public void remove(AlfabeticTree node){
        throw new UnsupportedOperationException("
            Operation not supported");
    };


}
```

```
import java.util.LinkedList;

public class AlfabeticLetter extends AlfabeticTree{


    private LinkedList <AlfabeticTree> children = new
        LinkedList <AlfabeticTree> ();

    public AlfabeticLetter(char letter) {

        this.letter=letter;

        }

    @Override
    public AlfabeticTree getNext(int i){

                AlfabeticTree tmp = children.get(i);

                return tmp;



```

```
    }

    @Override
    public boolean hasLetter(char c){

        /*
         */
        if(children.isEmpty()) {

            return false;
        }

        for(int i=0; i<children.size(); i++){

            if(c == children.get(i).letter){


                return true;

            };
        }

        return false;

    }

    @Override
    public AlfabeticTree findChildLetter(char c){
        for(int i=0; i<children.size(); i++){
            if( c == children.get(i).letter) {
                return children.get(i);
            }
        }
        return null;
    }
    @Override
    public void scan() {
        System.out.print(this.letter);

        for(int i=0; i<children.size(); i++){
            children.get(i).scan();
        }
    }
    @Override
    public void add(AlfabeticTree node){


        children.add(node);

    }
```

```
    @Override
74  public void remove(AlfabeticTree node){

76

        children.remove(node);
78

    }
80

82 }


2  public class LeafCharacter extends AlfabeticTree{
    char c;
4   public LeafCharacter(char c) {

6       this.c=c;
    }
8

    @Override
10  public void scan() {

12  }

14  @Override
    public boolean hasLetter(char in){
16

        if(this.c==in){
18          return true;
        }
20

        return false;
22  }


24


26 }
```

9 Word match

WordVisitor implementa una FSM che ha l'intento di verificare se nella lista di lettere ottenuta applicando le strategie di steganografiche si trova una o più parole del dataset. L'intento è realizzato andando a confrontare la lista di caratteri ottenuta dalla strategia steganografica inputStream con i caratteri presenti sull'albero. Quando una sequenza di lettere sull'inputStream viene trovata sui figli dell'albero, l'indice dell'albero avanza sulle lettere trovate. Se l'indice dell'albero avanza fino a raggiungere il carattere di fine parola implementato su Leaf, questo implica che quella parola è stata trovata. La parola trovata sarà immagazzinata su uno stack. Naturalmente l'FSM deve poter riconoscere le

sequenze che non portano ad una parola, quindi resettare l'indice dell'albero e svuotare lo stack.

```
2
public interface TreeVisitor {
4
    public boolean matchLetters(AlfabeticTree root);
6

8 }
```

```
2 import java.util.ArrayList;

4 import java.util.LinkedList;
import java.util.List;
6 import java.util.Stack;
public class WordVisitor implements TreeVisitor{
8     private LinkedList <Character> inputStream;
    private  List<Stack<Character>> stacks;
10
    private int countedWord;
12     private int status;
    private int indexStream;
14     //Map <Integer,List<Character>> outputStream = new
        HashMap <Integer,List<Character>>();
    private AlfabeticTree rootref;
16     public WordVisitor() {

18
    }
20     public WordVisitor(LinkedList <Character>
       inputStream, List<Stack<Character>> stacks){
        this.inputStream=inputStream;
22         this.stacks=stacks;

24     }
    public List<Stack<Character>> getStack(){
26         return stacks;
    }
28
    @Override
30     public boolean matchLetters(AlfabeticTree index){
        this.rootref=index;
32
        //System.out.println("InputStream: " +
           inputStream);
34
        boolean found = false;
36
```

```
        /*
38       * FSM receives an input list of letters
             inputStream and search for sequences of
             letters
         * that match keyword represented in the tree
             structure index.
40       * If one or many words are found it print a
             success message and print those words.
         */
42      this.status=0;
        this.indexStream=0;
44      this.countedWord=0;
        inputStream.add(AlfabeticTree.endLetter);
46      //outputStream.put(countedWord, new Stack());
        //List<Stack<Character>> stacks = new
           ArrayList<Stack<Character>>();
48      stacks.add(countedWord, new Stack<Character>()
           );
        while(indexStream < inputStream.size())
50      {

52      switch (status){

54              case 0:

56                  /* When you reach the last
                       character of a word you find an
                        endLetter character on the
                       tree structure*
                     */
58
                    if(true==index.hasLetter(
                       AlfabeticTree.endLetter)){
60
                        status = 2; //sequences of
                           characters has match an
                           full word on the tree
62                      //System.out.println("status
                           == 2");

64                  }
                    else {
66
                    /* If the current character isn't
                       found between children of
                       current index tree,
68                   * Pick another character (
                        incrementing indexStream),
                        restore the index to the root
                        of the tree
```

```
                    * and restart from state zero to
                       search new words.
70                  */
                        if(false==index.hasLetter(
                           inputStream.get(
                           indexStream).charValue
                           ())){
72                      ++indexStream;
                        index=this.rootref;
74                      status = 0; //no character
                            matches, restart.
                            if(stacks.get(
                               countedWord).
                               isEmpty()==false){
76                              stacks.get(
                                   countedWord).
                                   removeAllElements
                                   ();
                                /*When a
                                   character is
                                   not found and
                                   the whole word
                                    is not
                                   matched,
78                                                  remove
                                                     from
                                                     the
                                                     stac
                                                     any
                                                     char
                                                     coll
                                                     */

                                /*for(int r=(
                                   stacks.get(
                                   countedWord).
                                   size())-1; r>0;
                                   r--){
80                                  System.out.
                                       println("
                                       Butta via "
                                        + stacks.
                                       get(
```

```
                                        countedWord
                                        ).get(r));

82                                  }*/
                                }
84                          }

86                  /* If the current character is
                       found between children of
                       current index tree,
                     * set the status to one.
88                   */
                            else {
90
                            status = 1; //sequences of
                                 characters has match a
                                 the first part of a
                                word, but it's still
                                incomplete
92
                            //stacks.get(countedWord).
                                add(inputStream.get(
                                indexStream));
94
                            }
96                  }
                    break;
98
                case 1:
100
                    /* When a character is found, the
                       index move down on this
                       character.
102                  * Then a new character is picked
                        and the process restart from
                        state zero with the new index
                     */
104             //  System.out.println("Aggiungi "+
                   inputStream.get(indexStream));
                    stacks.get(countedWord).add(
                       inputStream.get(indexStream));
106                     index = (AlfabeticLetter)
                           index.findChildLetter(
                           inputStream.get(indexStream
                           ).charValue());
                        ++indexStream;
108                     status = 0;

110
```

```
112
                        break;
114
                        /* When a sequence of a character
                           matches a word notify the
                           success.
116                         * Then restore the index to the
                             root of the tree and continue
                             searching others words.
                         */
118
                    case 2:
120

122                         index=rootref;
                            status=0;
124                         stacks.add(++countedWord, new
                               Stack<Character>());

126                         System.out.println("Found "+(
                               countedWord)+" words.");
                            found=true;
128                         break;

130             }

132         }
            //System.out.println(stacks);
134         return found;
        }
136

138 }
```