

UNIVERSITA' DEGLI STUDI DI NAPOLI "FEDERICO II"

CORSO DI LAUREA IN INGEGNERIA
INFORMATICA

**ESAME DI INGEGNERIA DEL
SOFTWARE**

SISTEMA SOCIETA' PNEUMATICI

PROF. STEFANO RUSSO

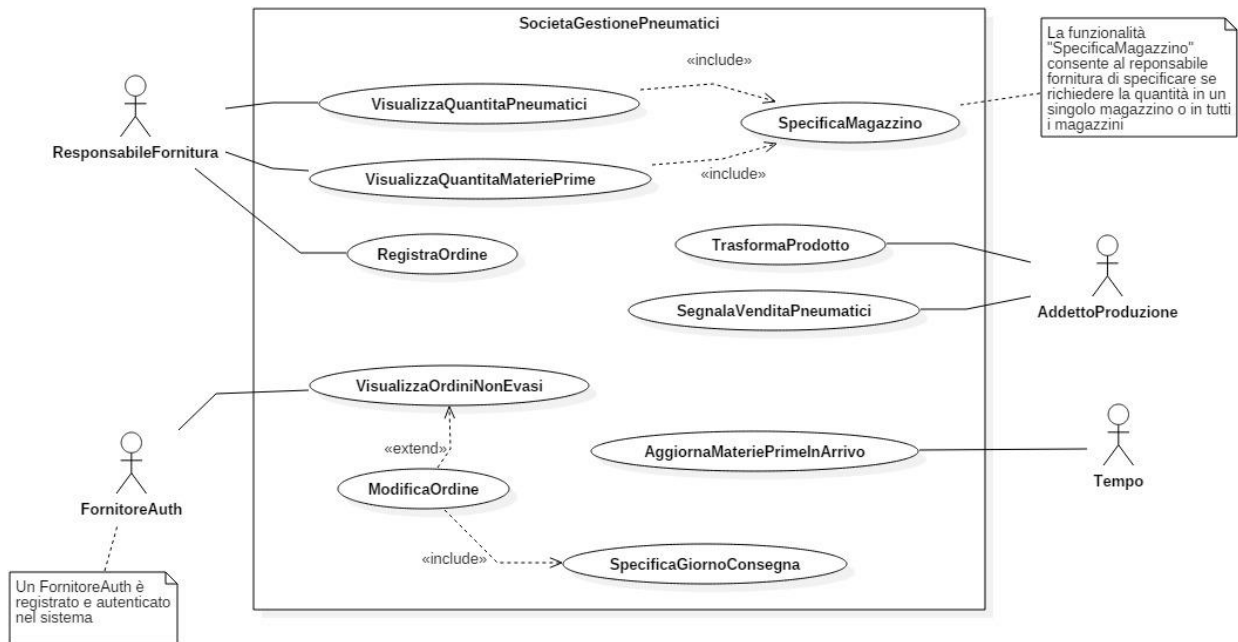
DANIELE VERGARA

N46001562



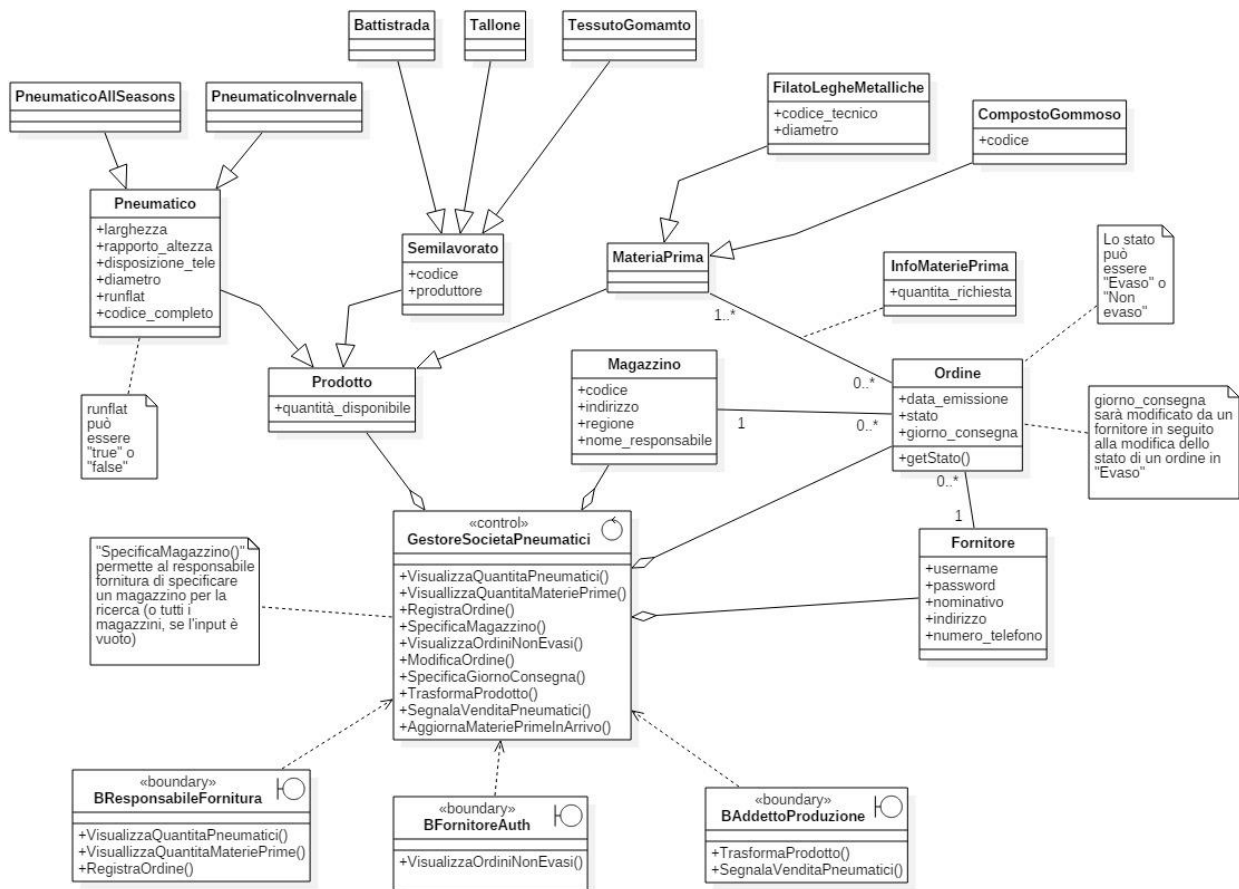
1 - UML

1.1 – Diagramma dei casi d'uso

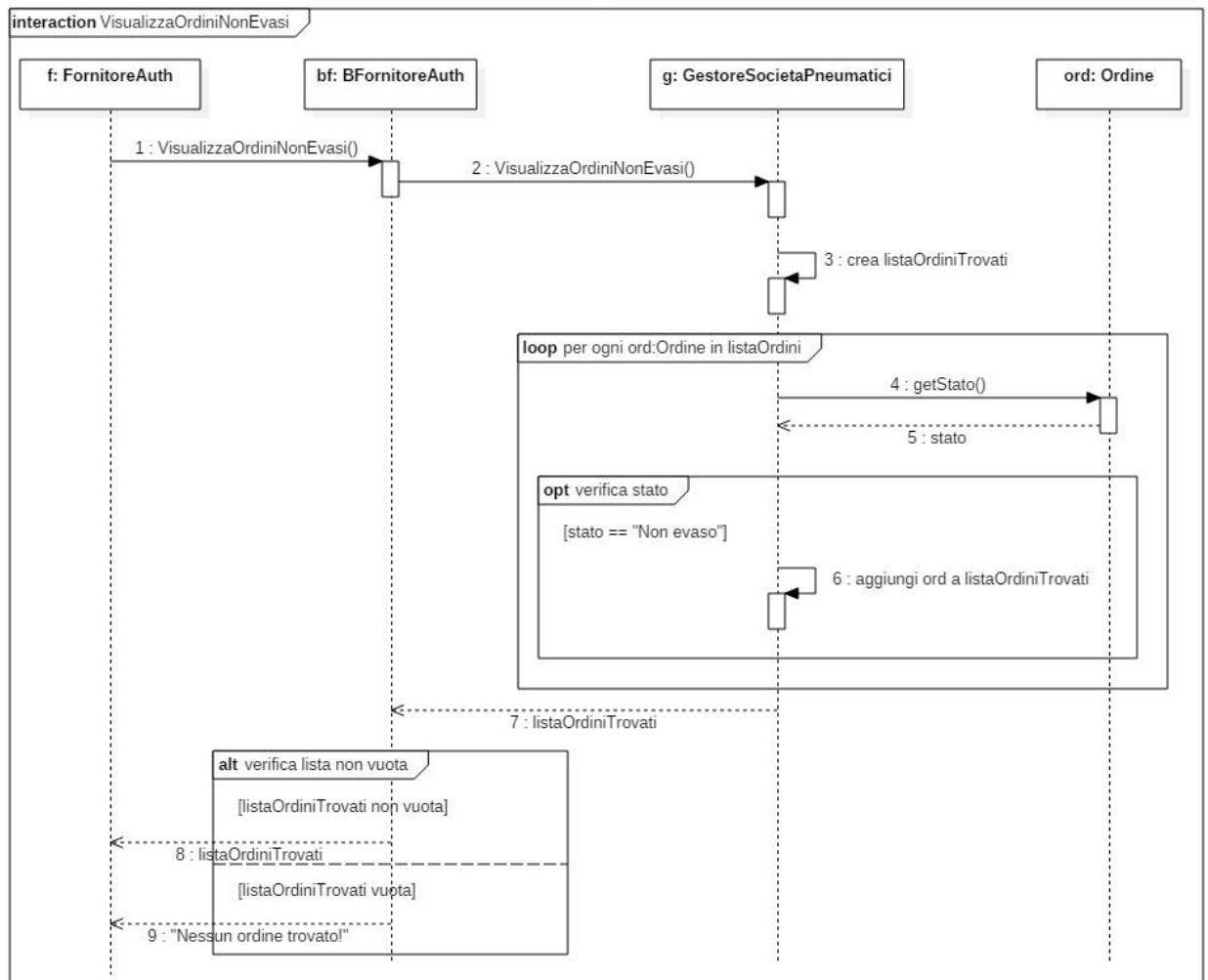


1.2 – Diagrammi di analisi

1.2.1 – Diagramma delle classi raffinato

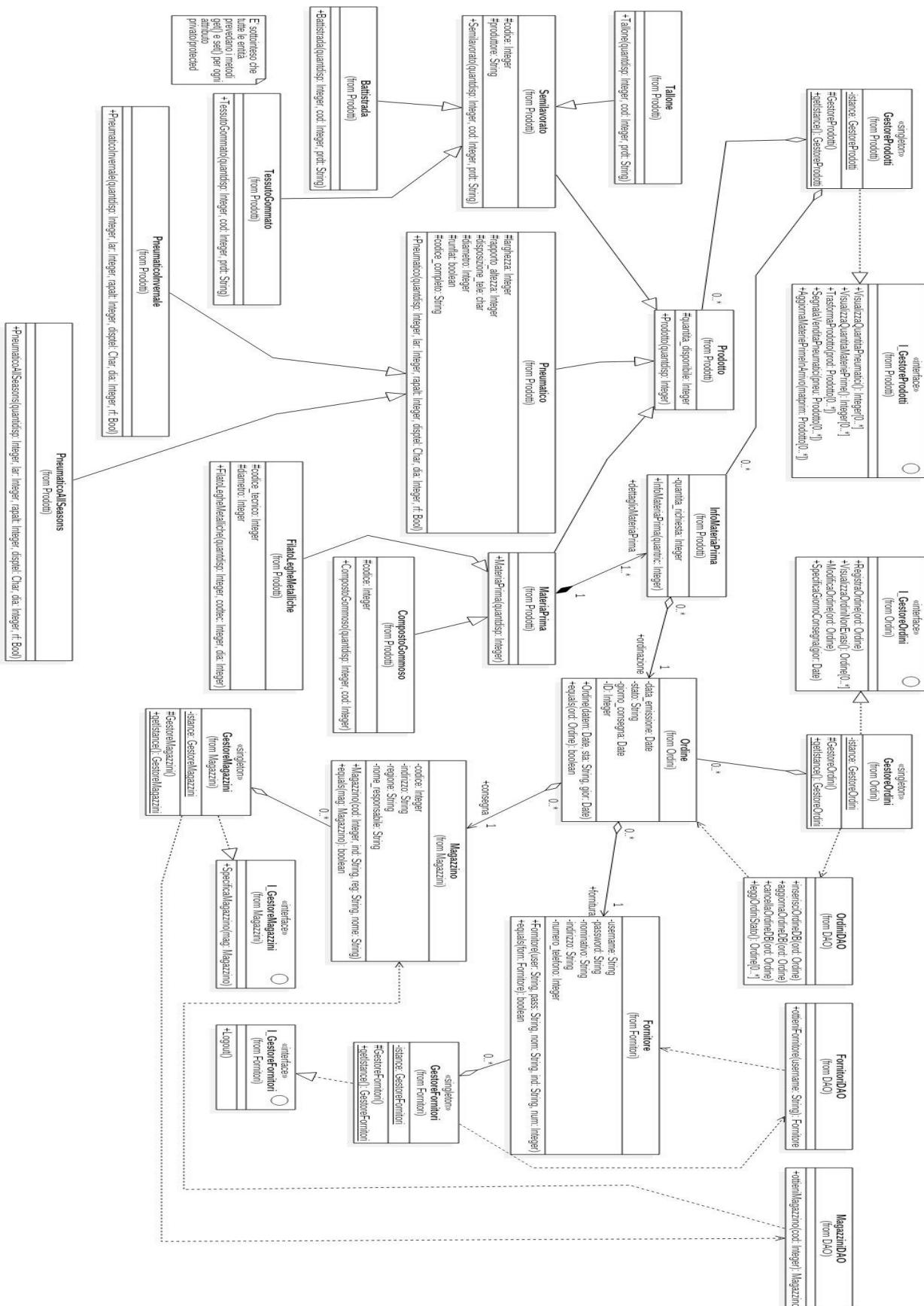


1.2.2 – Diagramma di sequenza raffinato

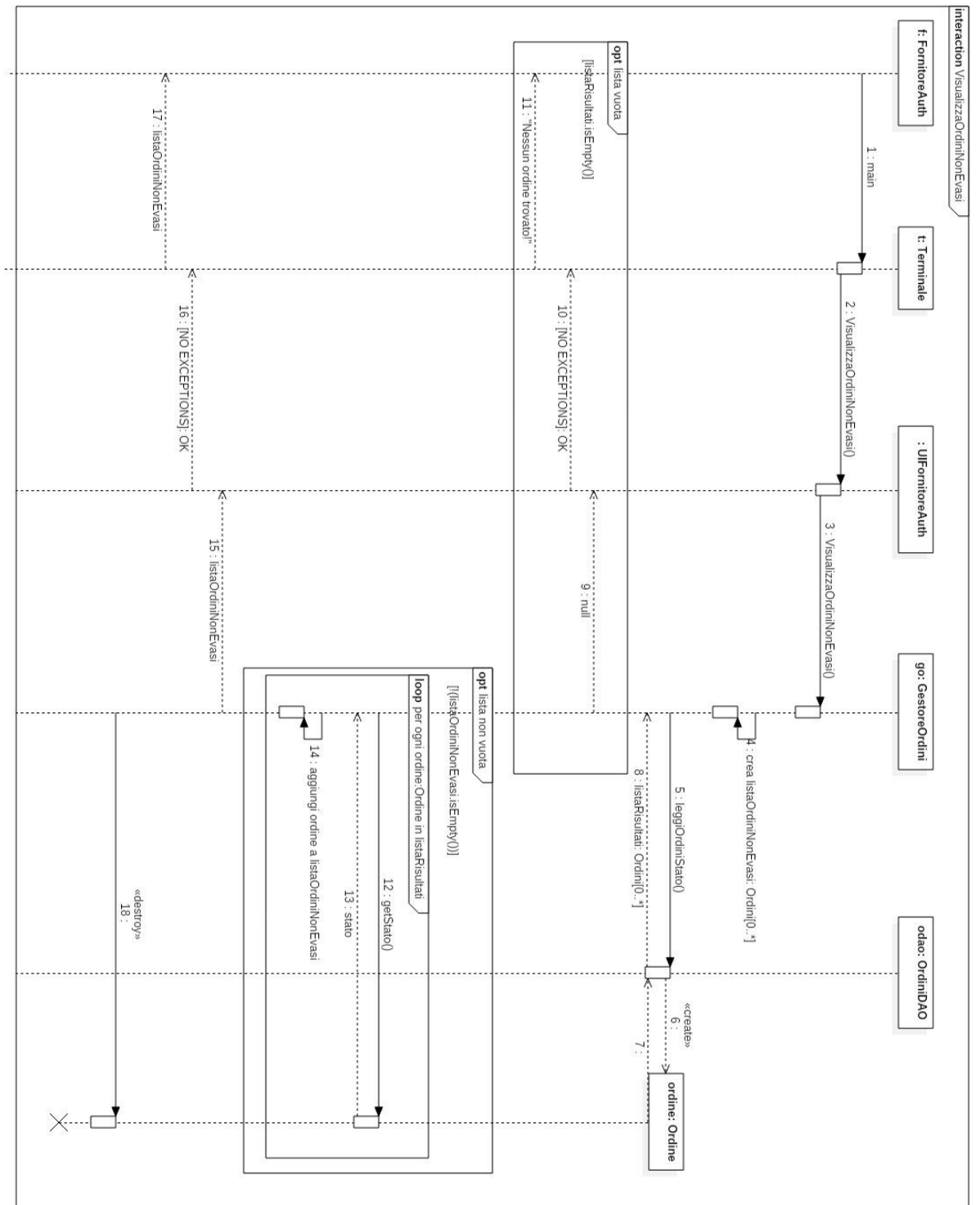


1.3 – Diagrammi di design

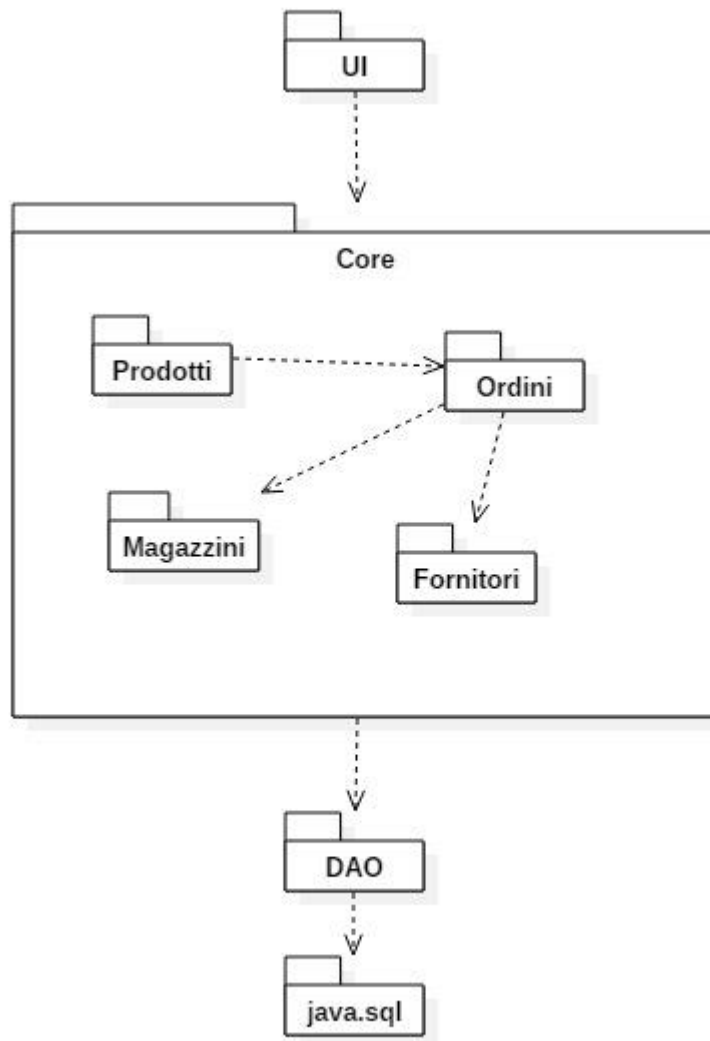
1.3.1 – Diagramma delle classi di design



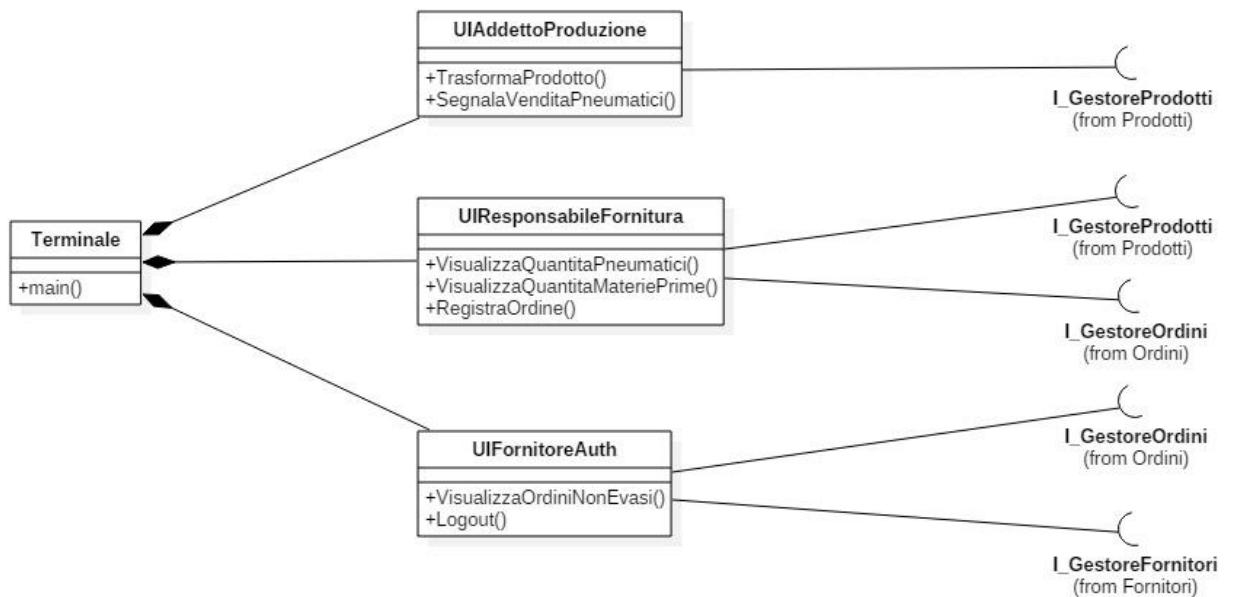
1.3.2 – Diagramma di sequenza raffinato



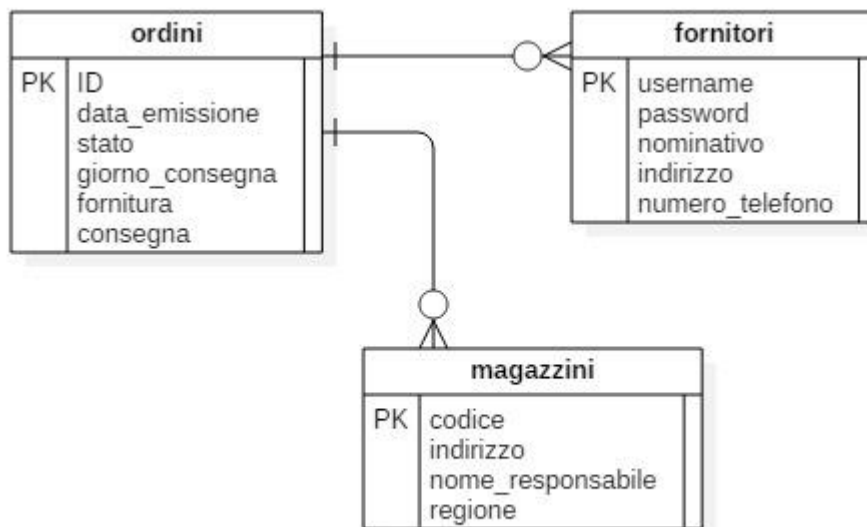
1.3.3 – Diagramma dei package



1.3.4 – Diagramma UI



1.3.5 – Diagramma ER



2 – Java

2.1 – Package Core

2.1.1 – Fornitore.java

```
3 package Core.Fornitori;
4
5 /**
6  *
7  */
8 public class Fornitore {
9
10     /**
11      * Default constructor
12      */
13     public Fornitore() {
14     }
15
16     /**
17      *
18      */
19     private String username;
20
21     /**
22      *
23      */
24     private String password;
25
26     /**
27      *
28      */
29     private String nominativo;
30
31     /**
32      *
33      */
34     private String indirizzo;
35
36     /**
37      *
38      */
39     private Integer numero_telefono;
40
41
42     /**
43      * @param user
```



```

44     * @param pass
45     * @param nom
46     * @param ind
47     * @param num
48     */
49     public Fornitore(String user, String pass, String nom, String ind, Integer num) {
50         this.indirizzo = ind;
51         this.username = user;
52         this.nominativo = nom;
53         this.password = pass;
54         this.numero_telefono = num;
55     }
56
57
58     public String getUsername() {
59         return username;
60     }
61
62
63     public void setUsername(String username) {
64         this.username = username;
65     }
66
67
68     public String getPassword() {
69         return password;
70     }
71
72
73     public void setPassword(String password) {
74         this.password = password;
75     }
76
77
78     public String getNominativo() {
79         return nominativo;
80     }
81
82
83     public void setNominativo(String nominativo) {
84         this.nominativo = nominativo;
85     }
86
87
88     public String getIndirizzo() {
89         return indirizzo;
90     }
91
92
93     public void setIndirizzo(String indirizzo) {
94         this.indirizzo = indirizzo;
95     }
96
97
98     public Integer getNumero_telefono() {
99         return numero_telefono;
100    }
101
102
103    public void setNumero_telefono(Integer numero_telefono) {
104        this.numero_telefono = numero_telefono;
105    }
106
107    public boolean equals(Fornitore f) {
108        return((this.username).equals(f.username) && (this.password).equals(f.password) &&
109        (this.numero_telefono).equals(f.numero_telefono) && (this.nominativo).equals(f.nominativo) &&
110        (this.indirizzo).equals(f.indirizzo));
111    }

```

2.1.2 – GestoreFornitori.java

```

2     package Core.Fornitori;
3
4     /**
5     *

```

```

6      */
7      public class GestoreFornitori implements I_GestoreFornitori {
8
9          /**
10         *
11         */
12         private static GestoreFornitori istance;
13
14
15         /**
16         *
17         */
18         protected GestoreFornitori() {
19             // TODO implement here
20         }
21
22         /**
23         * @return
24         */
25         public static GestoreFornitori getInstance() {
26             if(istance == null)
27                 istance = new GestoreFornitori();
28             return istance;
29         }
30
31         /**
32         *
33         */
34         public void Logout() {
35             System.out.println("Logout effettuato!");
36             System.out.println("Arrivederci, chiusura applicazione in corso...");
37
38             try {
39                 Thread.sleep(2000);
40             } catch (InterruptedException ex) {
41                 Thread.currentThread().interrupt();
42             }
43
44             System.exit(0);
45         }
46     }
47 }

```

2.1.3 – I_GestoreFornitori.java

```

3      package Core.Fornitori;
4
5      /**
6      *
7      */
8      public interface I_GestoreFornitori {
9
10         /**
11         *
12         */
13         public void Logout();
14     }
15 }

```

2.1.4 – Magazzino.java

```

3      package Core.Magazzini;
4
5      /**
6      *
7      */
8      public class Magazzino {
9
10         /**
11         * Default constructor
12         */
13         public Magazzino() {
14         }
15
16         /**
17         *
18         */
19         private Integer codice;

```

```

20
21     /**
22      *
23      */
24     private String indirizzo;
25
26     /**
27      *
28      */
29     private String regione;
30
31     /**
32      *
33      */
34     private String nome_responsabile;
35
36
37     /**
38      * @param cod
39      * @param ind
40      * @param reg
41      * @param nome
42      */
43     public Magazzino(Integer cod, String ind, String reg, String nome) {
44         this.codice = cod;
45         this.indirizzo = ind;
46         this.regione = reg;
47         this.nome_responsabile = nome;
48     }
49
50
51     public Integer getCodice() {
52         return codice;
53     }
54
55
56     public void setCodice(Integer codice) {
57         this.codice = codice;
58     }
59
60
61     public String getIndirizzo() {
62         return indirizzo;
63     }
64
65
66     public void setIndirizzo(String indirizzo) {
67         this.indirizzo = indirizzo;
68     }
69
70
71     public String getRegione() {
72         return regione;
73     }
74
75
76     public void setRegione(String regione) {
77         this.regione = regione;
78     }
79
80
81     public String getNome_responsabile() {
82         return nome_responsabile;
83     }
84
85
86     public void setNome_responsabile(String nome_responsabile) {
87         this.nome_responsabile = nome_responsabile;
88     }
89
90     public boolean equals(Magazzino m) {
91         return ((this.codice).equals(m.codice) && (this.indirizzo).equals(m.indirizzo) &&
92 (this.nome_responsabile).equals(m.nome_responsabile) && (this.regione).equals(m.regione));
93     }
94 }

```

2.1.5 – GestoreMagazzini.java

```
3 package Core.Magazzini;
4
5 /**
6  *
7  */
8 public class GestoreMagazzini implements I_GestoreMagazzini {
9
10     /**
11      *
12      */
13     private static GestoreMagazzini instance;
14
15
16     /**
17      *
18      */
19     protected GestoreMagazzini() {
20         // TODO implement here
21     }
22
23     /**
24      * @return
25      */
26     public static GestoreMagazzini getIstance() {
27         if(instance == null)
28             instance = new GestoreMagazzini();
29         return instance;
30     }
31
32     /**
33      * @param mag
34      */
35     public void SpecificaMagazzino(Magazzino mag) {
36         // TODO implement here
37     }
38
39 }
```

2.1.6 – I_GestoreMagazzini.java

```
3 package Core.Magazzini;
4
5 /**
6  *
7  */
8 public interface I_GestoreMagazzini {
9
10     /**
11      * @param mag
12      */
13     public void SpecificaMagazzino(Magazzino mag);
14
15 }
```

2.1.7 – Ordine.java

```
3 package Core.Ordini;
4
5 import java.util.*;
6 import Core.Magazzini.*;
7 import Core.Fornitori.*;
8
9 /**
10  *
11  */
12 public class Ordine {
13
14     /**
15      * Default constructor
16      */
17     public Ordine() {
18         this.data_emissione = null;
19         this.stato = null;
20         this.giorno_consegna = null;
21     }
22 }
```

```

23     /**
24     *
25     */
26     private Date data_emissione;
27
28     /**
29     *
30     */
31     private String stato;
32
33     /**
34     *
35     */
36     private Date giorno_consegna;
37
38     /**
39     *
40     */
41     private Integer ID;
42
43     /**
44     *
45     */
46     private Magazzino consegna;
47
48
49     /**
50     *
51     */
52     private Fornitore fornitura;
53
54     /**
55     * @param datem
56     * @param sta
57     * @param gior
58     */
59     public Ordine(Integer newID, Date datem, String sta, Date gior, Magazzino mag, Fornitore forn) {
60         this.ID = newID;
61         this.data_emissione = datem;
62         this.stato = sta;
63         this.giorno_consegna = gior;
64         this.consegna = mag;
65         this.fornitura = forn;
66     }
67
68     public Date getData_emissione() {
69         return data_emissione;
70     }
71
72     public void setData_emissione(Date data_emissione) {
73         this.data_emissione = data_emissione;
74     }
75
76     public String getStato() {
77         return stato;
78     }
79
80     public void setStato(String stato) {
81         this.stato = stato;
82     }
83
84     public Date getGiorno_consegna() {
85         return giorno_consegna;
86     }
87
88     public void setGiorno_consegna(Date giorno_consegna) {
89         this.giorno_consegna = giorno_consegna;
90     }
91
92     public Integer getID() {
93         return ID;
94     }
95
96     public void setID(Integer iD) {
97         ID = iD;
98     }

```

```

99
100 public Magazzino getConsegna() {
101     return consegna;
102 }
103
104 public void setConsegna(Magazzino consegna) {
105     this.consegna = consegna;
106 }
107
108 public Fornitore getFornitura() {
109     return fornitura;
110 }
111
112 public void setFornitura(Fornitore fornitura) {
113     this.fornitura = fornitura;
114 }
115
116 public boolean equals(Ordine o) {
117     return ((this.ID).equals(o.ID) && ((this.data_emissione.getDate() ==
o.data_emissione.getDate() && this.data_emissione.getMonth() == o.data_emissione.getMonth())) &&
(this.stato).equals(o.stato) && (this.consegna).equals(o.consegna) &&
(this.fornitura).equals(o.fornitura));
118 }
119
120 }

```

2.1.8 – GestoreOrdini.java

```

3 package Core.Ordini;
4
5 import java.util.*;
6 import DAO.*;
7
8 /**
9  *
10 */
11 public class GestoreOrdini implements I_GestoreOrdini {
12
13     /**
14      *
15      */
16     private static GestoreOrdini instance;
17
18
19     /**
20      *
21      */
22     protected GestoreOrdini() {
23         // TODO implement here
24     }
25
26     /**
27      * @return
28      */
29     public static GestoreOrdini getInstance() {
30         if(instance==null)
31             instance=new GestoreOrdini();
32         return instance;
33     }
34
35     /**
36      * @param ord
37      */
38     @Override
39     public void RegistraOrdine(Ordine ord) {
40         // TODO implement here
41     }
42
43     /**
44      * @return
45      */
46     @Override
47     public ArrayList<Ordine> VisualizzaOrdiniNonEvasi() {
48         ArrayList<Ordine> listaOrdiniNonEvasi = new ArrayList<Ordine>();
49         OrdiniDAO ordDAO = new OrdiniDAO();
50         listaOrdiniNonEvasi = ordDAO.leggiOrdiniStato();
51         return listaOrdiniNonEvasi;

```

```

52     }
53
54     /**
55     * @param ord
56     */
57     @Override
58     public void ModificaOrdine(Ordine ord) {
59         // TODO implement here
60     }
61
62     /**
63     * @param gior
64     */
65     @Override
66     public void SpecificaGiornoConsegna(Date gior) {
67         // TODO implement here
68     }
69
70 }

```

2.1.9 – I_GestoreOrdini.java

```

3     package Core.Ordini;
4
5     import java.util.*;
6
7     /**
8     *
9     */
10    public interface I_GestoreOrdini {
11
12        /**
13        * @param ord
14        */
15        public void RegistraOrdine(Ordine ord);
16
17        /**
18        * @return
19        */
20        public ArrayList<Ordine> VisualizzaOrdiniNonEvasi();
21
22        /**
23        * @param ord
24        */
25        public void ModificaOrdine(Ordine ord);
26
27        /**
28        * @param gior
29        */
30        public void SpecificaGiornoConsegna(Date gior);
31
32    }

```

2.2– Package DAO

2.2.1 – DBManager.java

```

3     package DAO;
4
5     import java.sql.Connection;
6     import java.sql.DriverManager;
7     import java.sql.SQLException;
8
9     public class DBManager
10    {
11        private static java.sql.Connection connection;
12
13        public static Connection OpenConnection( )
14        {
15            try
16            {
17                Class.forName("com.mysql.jdbc.Driver").newInstance( );
18                connection =
19                DriverManager.getConnection("jdbc:mysql://localhost/pneumaticiDB?user=root&password=root");
20                return connection;
21            }
22            catch (Exception e)
23            {
24                e.printStackTrace();
25            }
26        }
27    }

```

```

20     }
21
22     catch (ClassNotFoundException e)
23     {
24         // TODO Auto-generated catch block
25         e.printStackTrace();
26         return null;
27     }
28     catch (SQLException e)
29     {
30         // TODO Auto-generated catch block
31         e.printStackTrace();
32         return null;
33     }
34     catch (InstantiationException e)
35     {
36         // TODO Auto-generated catch block
37         e.printStackTrace();
38         return null;
39     }
40     catch (IllegalAccessException e)
41     {
42         // TODO Auto-generated catch block
43         e.printStackTrace();
44         return null;
45     }
46 }
47
48 public static void CloseConnection( )
49 {
50     try
51     {
52         connection.close( );
53     }
54
55     catch (SQLException e)
56     {
57         // TODO Auto-generated catch block
58         e.printStackTrace();
59     }
60 }
61 }
62
63

```

2.2.2 – FornitoriDAO.java

```

3     package DAO;
4
5     import java.sql.ResultSet;
6     import java.sql.SQLException;
7     import java.sql.Statement;
8     //import java.util.*;
9     import Core.Fornitori.*;
10    //import Core.Magazzini.Magazzino;
11
12    public class FornitoriDAO {
13
14        private static java.sql.Connection connection;
15
16        public Fornitore ottieniFornitore(String username) {
17            connection = DBManager.OpenConnection( );
18
19            /*if (connection != null) {
20                System.out.println("Connesso con successo al database");
21            }*/
22
23            Fornitore fornitore = new Fornitore( );
24
25            try
26            {
27                String tabella = "fornitori";
28                String query_fornitori = "SELECT * FROM " + tabella + " WHERE username='" +
username + "'";
29                Statement st = connection.createStatement();

```



```

30         ResultSet rs = st.executeQuery(query_fornitori);
31
32         while (rs.next( ))
33         {
34             fornitore.setUsername(rs.getString("username"));
35             fornitore.setPassword(rs.getString("password"));
36             fornitore.setIndirizzo(rs.getString("indirizzo"));
37             fornitore.setNominativo(rs.getString("nominativo"));
38             fornitore.setNumero_telefono(rs.getInt("numero_telefono"));
39         }
40
41         st.close( );
42     }
43     catch (SQLException e)
44     {
45         e.printStackTrace( );
46     }
47
48     DBManager.CloseConnection( );
49
50     return fornitore;
51 }
52 }

```

2.2.3 – MagazziniDAO.java

```

3     package DAO;
4
5     import java.sql.ResultSet;
6     import java.sql.SQLException;
7     import java.sql.Statement;
8
9     //import java.util.*;
10    import Core.Magazzini.*;
11
12
13    public class MagazziniDAO {
14        private static java.sql.Connection connection;
15
16        public Magazzino ottieniMagazzino(int cod) {
17            connection = DBManager.OpenConnection( );
18
19            /*if (connection != null) {
20                System.out.println("Connesso con successo al database");
21            }*/
22
23            Magazzino magazzino = new Magazzino( );
24
25            try
26            {
27                String tabella = "magazzini";
28                Integer.toString(cod);
29                String query_magazzini = "SELECT * FROM " + tabella + " WHERE codice='" + cod + "'";
30                Statement st = connection.createStatement();
31                ResultSet rs = st.executeQuery(query_magazzini);
32
33                while (rs.next( ))
34                {
35                    magazzino.setCodice(rs.getInt("codice"));
36                    magazzino.setIndirizzo(rs.getString("indirizzo"));
37                    magazzino.setNome_responsabile(rs.getString("nome_responsabile"));
38                    magazzino.setRegione(rs.getString("regione"));
39                }
40
41                st.close( );
42            }
43            catch (SQLException e)
44            {
45                e.printStackTrace( );
46            }
47
48            DBManager.CloseConnection( );
49
50            return magazzino;
51        }

```

52 }

2.2.4 – OrdiniDAO.java

```
3     package DAO;
4
5     import java.sql.ResultSet;
6     import java.sql.SQLException;
7     import java.sql.Statement;
8
9     import java.util.ArrayList;
10
11     import Core.Ordini.*;
12
13     /**
14      *
15      */
16     public class OrdiniDAO {
17
18         private static java.sql.Connection connection;
19
20         /**
21          * Default constructor
22          */
23         public OrdiniDAO() {
24         }
25
26         /**
27          * @param ord
28          */
29         public void inserisciOrdineDB(Ordine ord) {
30             // TODO implement here
31         }
32
33         /**
34          * @param ord
35          */
36         public void aggiornaOrdineDB(Ordine ord) {
37             // TODO implement here
38         }
39
40         /**
41          * @param ord
42          */
43         public void cancellaOrdineDB(Ordine ord) {
44             // TODO implement here
45         }
46
47         /**
48          * @return
49          */
50         public ArrayList<Ordine> leggiOrdiniStato() {
51             ArrayList<Ordine> listaOrdiniNonEvasi = new ArrayList<Ordine>();
52             MagazziniDAO magdao = new MagazziniDAO();
53             FornitoriDAO forndao = new FornitoriDAO();
54
55             String userforn;
56             int codmag;
57
58             connection = DBManager.OpenConnection( );
59
60             /*if (connection != null) {
61                 System.out.println("Connesso con successo al database");
62             }*/
63
64             try
65             {
66                 String tabella = "ordini";
67                 String stato = "Non evaso";
68                 String query_ordini_non_evasi = "SELECT * FROM " + tabella + " WHERE stato='" + stato
69                 + "'";
70
71                 Statement st = connection.createStatement();
72                 ResultSet rs = st.executeQuery(query_ordini_non_evasi);
73
74                 while (rs.next( ))
```

```

73         {
74             Ordine ordine = new Ordine( );
75             ordine.setID(rs.getInt("ID"));
76             ordine.setStato(rs.getString("stato"));
77             ordine.setData_emissione(rs.getDate("data_emissione"));
78             ordine.setGiorno_consegna(rs.getDate("giorno_consegna"));
79
80             userforn = rs.getString("fornitura");
81             ordine.setFornitura(forndao.ottieniFornitore(userforn));
82
83             codmag = rs.getInt("consegna");
84             ordine.setConsegna(magdao.ottieniMagazzino(codmag));
85
86             listaOrdiniNonEvasi.add(ordine);
87         }
88
89         st.close( );
90     }
91     catch (SQLException e)
92     {
93         e.printStackTrace( );
94     }
95
96     DBManager.CloseConnection( );
97
98     if (listaOrdiniNonEvasi.isEmpty()) {
99         System.out.println("Nessun ordine trovato!");
100         return null;
101     }
102     else return listaOrdiniNonEvasi;
103 }
104
105 }

```

2.3 – Package Test

2.3.1 – ordiniNonEvasiTest.java

```

3  package Test;
4
5  import java.util.Date;
6
7  import org.junit.Test;
8
9  import Core.Ordini.*;
10 import Core.Fornitori.*;
11 import Core.Magazzini.*;
12
13 public class ordiniNonEvasiTest {
14
15     GestoreOrdini gestore = GestoreOrdini.getInstance();
16
17     @Test
18     public void testListaNonVuota() {
19         assert(!(gestore.VisualizzaOrdiniNonEvasi().isEmpty()));
20     }
21
22     @Test
23     public void testNumeroElementiLista() {
24         assert((gestore.VisualizzaOrdiniNonEvasi().size() == 3));
25     }
26
27     @Test
28     public void testGiornoConsegnaNullo() {
29         boolean giornoNonNullo = true;
30
31         for (Ordine o : gestore.VisualizzaOrdiniNonEvasi())
32             giornoNonNullo = giornoNonNullo && (o.getGiorno_consegna() == null);
33
34         assert(giornoNonNullo);
35     }
36
37     @Test
38     public void testOrdinePresente() {
39         Magazzino m = new Magazzino(346, "via Roma 4", "Campania", "Stefano");

```

```

40     Fornitore f = new Fornitore("pirelli", "f12017", "Tronchetti Provera", "Via Alberto Pirelli
25", 345098);
41
42     Date d = new Date();
43     d.setYear(2017);
44     d.setMonth(5);
45     d.setDate(29);
46
47     Ordine ord = new Ordine(165, d, "Non evaso", null, m, f);
48
49     boolean presente = false;
50
51     for (Ordine o : gestore.VisualizzaOrdiniNonEvasi()) {
52         presente = presente || o.equals(ord);
53     }
54     assert(presente);
55 }
56 }

```

2.4 – Package UI

2.4.1 – UIFornitoreAuth.java

```

3  package UI;
4
5  import java.util.*;
6  import java.io.InputStreamReader;
7  import java.io.BufferedReader;
8  import java.io.IOException;
9
10 import Core.Ordini.*;
11 import Core.Fornitori.*;
12 import Core.Magazzini.*;
13
14 /**
15  *
16  */
17 public class UIFornitoreAuth {
18
19     /**
20      * Default constructor
21      */
22     public UIFornitoreAuth() {
23     }
24
25     public void mainFornitoreAuth() {
26         System.out.println("Benvenuto nel sistema di gestione pneumatici!");
27
28         GestoreOrdini gestord;
29         gestord = GestoreOrdini.getInstance();
30         ArrayList<Ordine> listaOrdiniNonEvasi = new ArrayList<Ordine>();
31
32         GestoreFornitori gestforn;
33         gestforn = GestoreFornitori.getInstance();
34
35         System.out.println(" - Menu - ");
36         System.out.println(" 1) Visualizza tutti gli ordini che non sono ancora stati evasi");
37         System.out.println(" 2) Logout");
38
39         InputStreamReader tastiera = new InputStreamReader(System.in);
40         BufferedReader bufferTastiera = new BufferedReader(tastiera);
41
42         String codice = null;
43
44         System.out.println("Premi il tasto corrispondente alla funzionalità che vuoi utilizzare:");
45
46         try {
47             codice = bufferTastiera.readLine();
48         } catch (IOException e) {
49             System.out.println("Errore nella lettura da tastiera, riprovare: ");
50             e.printStackTrace();
51         }
52
53         switch(codice) {

```

```

54         case "1": {
55             listaOrdiniNonEvasi = gestord.VisualizzaOrdiniNonEvasi();
56             Fornitore fornstamp = new Fornitore();
57             Magazzino magstamp = new Magazzino();
58             for (Ordine ord : listaOrdiniNonEvasi) {
59                 System.out.println("Info ordine:");
60                 System.out.println("ID: " + ord.getID() + " - Stato: " + ord.getStato() + " - Data
emissione: " + ord.getData_emissione() + " - Giorno consegna: " + ord.getGiorno_consegna() + " -
Fornitore: " + (ord.getFornitura()).getUsername() + " - Magazzino: " + (ord.getConsegna()).getCodice()
);
61                 fornstamp = ord.getFornitura();
62                 System.out.println("");
63                 System.out.println("Info fornitore associato:");
64                 System.out.println("[Username: " + fornstamp.getUsername() + " - Password: " +
fornstamp.getPassword() + " - Nominativo: " + fornstamp.getNominativo() + " - Indirizzo: " +
fornstamp.getIndirizzo() + " - Numero telefono: " + fornstamp.getNumero_telefono() + "]");
65                 magstamp = ord.getConsegna();
66                 System.out.println("Info magazzino associato:");
67                 System.out.println("[Codice: " + magstamp.getCodice() + " - Nome responsabile: " +
magstamp.getNome_responsabile() + " - Indirizzo: " + magstamp.getIndirizzo() + " - Regione: " +
magstamp.getRegione() + "]");
68                 System.out.println("");
69                 System.out.println("");
70             }
71             break;
72         }
73         case "2": {
74             gestforn.Logout();
75             break;
76         }
77         default: {
78             System.out.println("Nessuna funzionalità corrisponde al carattere
digitato!");
79             break;
80         }
81     }
82 }
83
84     System.out.println("Applicazione in chiusura...");
85 }
86
87 /**
88  *
89  */
90 public void Logout() {
91     // TODO implement here
92 }
93
94 /**
95  * @param ord
96  */
97 public void RegistraOrdine(Ordine ord) {
98     // TODO implement here
99 }
100
101 /**
102  * @return
103  */
104 public ArrayList<Ordine> VisualizzaOrdiniNonEvasi() {
105     // TODO implement here
106     return null;
107 }
108
109 /**
110  * @param ord
111  */
112 public void ModificaOrdine(Ordine ord) {
113     // TODO implement here
114 }
115
116 /**
117  * @param gior
118  */
119 public void SpecificaGiornoConsegna(Date gior) {
120     // TODO implement here
121 }

```

```

122
123 }
2.4.2 – Terminale.java
3 package UI;
4
5 /**
6  *
7  */
8 public class Terminale {
9
10     /**
11      *
12      */
13     public static void main(String[] args){
14
15         System.out.println("Tipologia utente attualmente loggato: Fornitore");
16         UIFornitoreAuth uiform = new UIFornitoreAuth();
17         uiform.mainFornitoreAuth();
18     }
19
20 }
21

```

3 – Test info

3.1 – Tabella test cases

T C	Descrizione Test Case	Classi di Equivalenza Coperte	Precondizioni	Input	Output Attesi	Post- Condizioni Attese	Output Otteneruti	Post- Condizioni Ottenerate	Esito
testListaNonVuota	Si testa la lista per assicurarsi che non sia vuota	Lista non vuota	La lista di ordini da controllare esiste	-	Risultato in termini di vero/falso	La lista è stata controllata per cercare almeno un elemento	Risultato in termini di vero/falso	La lista è stata controllata per cercare almeno un elemento	Pass
testNumeroElementiLista	Si conta il numero di elementi in lista per assicurarsi che corrisponda a quello atteso	Numero elementi lista	La lista di ordini da controllare esiste e il numero di elementi attesi è stato definito (3)	-	Numero elementi in lista, risultato in termini di vero/falso	Il numero di elementi è stato contato	Risultato in termini di vero/falso	Il numero di elementi è stato contato	Pass
testGiornoConsegnaNullo	Si verifica che il giorno di consegna degli ordini non evasi sia nullo	Giorno Not Null	Tutti gli ordini in lista siano "Non evaso"	-	Risultato in termini di vero/falso	Tutti i giorni di consegna degli ordini sono stati scanditi	Risultato in termini di vero/falso	Tutti i giorni di consegna degli ordini sono stati scanditi	Pass
testOrdinePresente	Si verifica che un ordine specifico, restituito dalla funzionalità, esista effettivamente	Ordine presente	La lista abbia almeno un ordine al suo interno e l'ordine da testare sia stato preparato	-	Risultato in termini di vero/falso	Tutti gli ordini della lista sono stato confrontati con quello di riferimento	Risultato in termini di vero/falso	Tutti gli ordini della lista sono stato confrontati con quello di riferimento	Pass

3.2 – Risultati test cases



















Finished after 0,482 seconds

Runs: 4/4 Errors: 0 Failures: 0

Test.ordiniNonEvasiTest [Runner: JUnit 4] (0,170 s)

- testGiornoConsegnaNullo (0,039 s)
- testOrdinePresente (0,043 s)
- testNumeroElementiLista (0,047 s)
- testListaNonVuota (0,040 s)

3.3 – Coverage

Element	Coverage	Covered Instructio...	Missed Instructions	Total Instructions
▼  VERGARA_N46001562	 50,4 %	638	628	1.266
▼  src	 50,4 %	638	628	1.266
>  Core.Prodotti	 0,0 %	0	320	320
>  UI	 0,0 %	0	159	159
>  Core.Fornitori	 62,9 %	73	43	116
>  DAO	 85,3 %	249	43	292
>  Core.Magazzini	 69,8 %	60	26	86
>  Test	 87,2 %	130	19	149
>  Core.Ordini	 87,5 %	126	18	144

4. SQL

4.1 – Creazione schema e tabelle

```
CREATE SCHEMA `pneumaticidb` ;
```

```
CREATE TABLE `pneumaticidb`.`fornitori` (  
  `username` VARCHAR(45) NOT NULL,  
  `password` VARCHAR(45) NULL,  
  `nominativo` VARCHAR(45) NULL,  
  `indirizzo` VARCHAR(45) NULL,  
  `numero_telefono` INT NULL,  
  PRIMARY KEY (`username`));
```

```
CREATE TABLE `pneumaticidb`.`magazzini` (  
  `codice` INT NOT NULL,  
  `indirizzo` VARCHAR(45) NULL,  
  `nome_responsabile` VARCHAR(45) NULL,  
  `regione` VARCHAR(45) NULL,  
  PRIMARY KEY (`codice`));
```

```
CREATE TABLE `pneumaticidb`.`ordini` (  
  `ID` INT NOT NULL,  
  `data_emissione` DATE NULL,  
  `stato` VARCHAR(45) NULL,  
  `giorno_consegna` DATE NULL,  
  `fornitura` VARCHAR(45) NULL,  
  `consegna` INT NULL,  
  PRIMARY KEY (`ID`),  
  INDEX `ordini_fornitura_idx` (`fornitura` ASC),
```

```

INDEX `ordini_consegna_idx` (`consegna` ASC),

CONSTRAINT `ordini_fornitura`

FOREIGN KEY (`fornitura`)

REFERENCES `pneumaticidb`.`fornitori` (`username`)

ON DELETE CASCADE

ON UPDATE CASCADE,

CONSTRAINT `ordini_consegna`

FOREIGN KEY (`consegna`)

REFERENCES `pneumaticidb`.`magazzini` (`codice`)

ON DELETE CASCADE

ON UPDATE CASCADE);

```

4.2 – Popolamento database

```

INSERT INTO `pneumaticidb`.`fornitori` (`username`, `password`, `nominativo`, `indirizzo`, `numero_telefono`) VALUES ('bridgestone', 'gomme', 'Marco Galbiati', 'Via Murari 16', '335648');

```

```

INSERT INTO `pneumaticidb`.`fornitori` (`username`, `password`, `nominativo`, `indirizzo`, `numero_telefono`) VALUES ('pirelli', 'f12017', 'Tronchetti Provera', 'Via Alberto Pirelli 25', '345098');

```

```

INSERT INTO `pneumaticidb`.`fornitori` (`username`, `password`, `nominativo`, `indirizzo`, `numero_telefono`) VALUES ('michelin', 'omino', 'Lorenzo Rosso', 'Corso Romania 546', '333002');

```

```

INSERT INTO `pneumaticidb`.`fornitori` (`username`, `password`, `nominativo`, `indirizzo`, `numero_telefono`) VALUES ('continental', 'performance', 'Alessandro De Martino', 'Via Rondoni Pietro 1', '345882');

```

```

INSERT INTO `pneumaticidb`.`fornitori` (`username`, `password`, `nominativo`, `indirizzo`, `numero_telefono`) VALUES ('hankook', 'eco', 'Carlo Citarella', 'Viale Edison 110', '320430');

```

```

INSERT INTO `pneumaticidb`.`magazzini` (`codice`, `indirizzo`, `nome_responsabile`, `regione`) VALUES ('346', 'Via Roma 4', 'Stefano', 'Campania');

```

```

INSERT INTO `pneumaticidb`.`magazzini` (`codice`, `indirizzo`, `nome_responsabile`, `regione`) VALUES ('822', 'Corso Italia 34', 'Alfredo', 'Piemonte');

```

```

INSERT INTO `pneumaticidb`.`magazzini` (`codice`, `indirizzo`, `nome_responsabile`, `regione`) VALUES ('912', 'Via Delle Siepi', 'Gaetano', 'Veneto');

```

```

INSERT INTO `pneumaticidb`.`ordini` (`ID`, `data_emissione`, `stato`, `fornitura`, `consegna`) VALUES ('165', '2017-06-29', 'Non evaso', 'pirelli', '346');

```

```

INSERT INTO `pneumaticidb`.`ordini` (`ID`, `data_emissione`, `stato`, `giorno_consegna`, `fornitura`, `consegna`) VALUES ('290', '2017-06-15', 'Evaso', '2017-07-10', 'hankook', '912');

```

```

INSERT INTO `pneumaticidb`.`ordini` (`ID`, `data_emissione`, `stato`, `fornitura`, `consegna`) VALUES ('144', '2017-06-21', 'Non Evaso', 'bridgestone', '822');

```

```

INSERT INTO `pneumaticidb`.`ordini` (`ID`, `data_emissione`, `stato`, `fornitura`, `consegna`) VALUES ('382', '2017-07-14', 'Non evaso', 'michelin', '912');

```

```

INSERT INTO `pneumaticidb`.`ordini` (`ID`, `data_emissione`, `stato`, `giorno_consegna`, `fornitura`, `consegna`) VALUES ('176', '2017-07-12', 'Evaso', '2017-07-14', 'pirelli', '822');

```

```

INSERT INTO `pneumaticidb`.`ordini` (`ID`, `data_emissione`, `stato`, `giorno_consegna`, `fornitura`, `consegna`) VALUES ('227', '2017-06-10', 'Evaso', '2017-06-27', 'continental', '346');

```