```
import java.io.*;

/**
 * Classe di implementazione dell'Interfaccia relativa al Registro Frequenze.
 *
 * L'oggetto e' serializabile.
 *
 * @author Filippo Fontanelli , Francesca Brogi
 */

public class RegistroFrequenzeImp implements RegistroFrequenzeInterfaccia {

    // ************************************************

    private static final long serialVersionUID = 4L;

    // Rappresentazione

    /**
     * Lista contenente le Lezioni.
     */
    private RegLezioni lex;
    /**
     * Lista contenente gli Studenti.
     */
    private RegStudenti stu;
    /**
     * Nome del Registro.
     */
    private String name;
    /**
     * Nome del file del Registro.
     */
    private String fileName;
    /**
     * Numero minimo di lezioni.
     */
    private int nObbligatorio;

    // Costruttori
    /**
     *
     * Costruttore che crea un nuovo Registro Frequenze. Inizializzando il
     * fileName e il Nome con i parametri passati e le altre variabili con
     * elementi di default. Se un parametro e' vuoto o nullo viene sollevata la
     * relativa eccezione.
     *
     * @param fileName
     *            Stringa rappresentante il nome del file, sia binario che
     *            testuale(in entrambi i casi sensa estenzione), dove e' salvato
     *            il registo corrente.
     * @param Name
     *            Stringa rappresentante il nome del Registro.
```

```
     *
     * @exception <code>NullPointerException</code> se il nome o il fileName
     *            passati &egrave; <code>null</code>.
     * @exception <code>IllegalArgumentException</code> se il nome o il fileName
     *            passati sono uguali alla stringa vuota.
     */
    public RegistroFrequenzeImp(String fileName, String Name)
            throws NullPointerException, IllegalArgumentException {
        if (Name == null || fileName == null) {
            throw new NullPointerException(
                    "RegistroFrequenzeImp::Costruttore::null");
        }
        if (Name.equals("") || fileName.equals("")) {
            throw new IllegalArgumentException(
                    "RegistroFrequenzeImp::Costruttore::NullPointerException");
        }
        this.fileName = fileName + ".txt";
        this.name = fileName;
        stu = new RegStudenti(); // inizializza alla lista studenti vuota
        lex = new RegLezioni();// inizializza alla lista lezioni vuota
        nObbligatorio = 0;
        // this.loadTesto();
    }

    /**
     *
     * Costruttore che crea un nuovo Registro Frequenze. Inizializzando il
     * fileName con il parametri passato e le altre variabili con elementi di
     * default. Se un parametro e' vuoto o nullo viene sollevata la relativa
     * eccezione.
     *
     * @param FileName
     *            Stringa rappresentante il nome del file, sia binario che
     *            testuale(in entrambi i casi sensa estenzione), dove e' salvato
     *            il registo corrente.
     *
     * @exception <code>NullPointerException</code> se il nome o il fileName
     *            passati &egrave; <code>null</code>.
     * @exception <code>IllegalArgumentException</code> se il nome o il fileName
     *            passati sono uguali alla stringa vuota.
     */
    public RegistroFrequenzeImp(String FileName) throws NullPointerException,
            IllegalArgumentException {
        if (FileName == null) {
            throw new NullPointerException(
                    "RegistroFrequenzeImp::Costruttore::null");
        }
        if (FileName.equals("")) {
            throw new IllegalArgumentException(
                    "RegistroFrequenzeImp::Costruttore::NullPointerException");
        }
        this.fileName = FileName;
        this.name = FileName;
```

```
        stu = new RegStudenti(); // inizializza alla lista studenti vuota
        lex = new RegLezioni();// inizializza alla lista lezioni vuota
        nObbligatorio = 0;
    }

    /**
     *
     * Costruttore che crea un nuovo Registro Frequenze. Se un parametro e'
     * vuoto o nullo viene sollevata la relativa eccezione.
     *
     * @param fileName
     *            Stringa rappresentante il nome del file, sia binario che
     *            testuale(in entrambi i casi sensa estenzione), dove e' salvato
     *            il registo corrente.
     * @param name
     *            Stringa rappresentante il nome del Registro.
     * @param lex
     *            RegLezioni rappresentante il registro delle Lezioni.
     * @param stu
     *            RegStudenti rappresentante il registro degli Studenti.
     * @param nObbligatorio
     *            int rappresentante il numero minimo di lezioni necessario per
     *            l'ammissione all'esame.
     *
     * @exception <code>NullPointerException</code> se uno dei parametri passati
     *            &egrave; <code>null</code>.
     * @exception <code>IllegalArgumentException</code> se uno dei parametri
     *            passati è uguale alla stringa vuota e se il nObbligatorio < 0.
     */
    public RegistroFrequenzeImp(String fileName, String name, RegLezioni lex,
            RegStudenti stu, int nObbligatorio) throws NullPointerException,
            IllegalArgumentException {
        if (name == null || fileName == null || lex == null || stu == null) {
            throw new NullPointerException(
                    "RegistroFrequenzeImp::Costruttore::null");
        }
        if (name.equals("") || fileName.equals("") || nObbligatorio < 0) {
            throw new IllegalArgumentException(
                    "RegistroFrequenzeImp::Costruttore::NullPointerException");
        }
        this.fileName = fileName;
        this.lex = lex;
        this.name = name;
        this.stu = stu;
        if (nObbligatorio < lex.size())
            this.nObbligatorio = nObbligatorio;
        else
            this.nObbligatorio = lex.size();
    }

    // Osservatori
    public RegLezioni getLezioni() {
        return lex;
```

```
    }

    public int getNumObbligatorio() {
        return nObbligatorio;
    }

    public RegStudenti getStudenti() {
        return stu;
    }

    public String getName() {
        return name;
    }

    public String getFileName() {
        return fileName;
    }

    // Modificatori

    public void saveTesto() throws java.io.IOException {

        PrintWriter out = null;
        try {
            // incapsula in BufferedReader un file aperto in lettura
            out = new PrintWriter(new BufferedWriter(new FileWriter(fileName + ".txt"
))));

            out.print("Nome del Corso con frequenza obbligatoria :");
            out.println(name);

            out.println();
            out.println("Numero minimo di presenze :" + this.nObbligatorio);
            out.println();

            out.println();
            out.println("Elenco delle lezioni e degli studenti :");
            out.println();

            out.println("**Start Lezioni**");

            out.println(lex.toString());

            out.println("**End Lezioni**");

            out.println();

            out.println("**Start Studenti**");

            out.println(stu.toString());

            out.println("**End Studenti**");
```

```
            out.println();

            out.println("**Start Registro**");
            out.println("       Registro delle Presenze :");

            for (int i = 0; i < lex.size(); i++) {
                out.println("---Lezione:---");

                out.println(lex.getElemReg().get(i).toString());

                out.println("---Studenti:---");

                for (int k = 0; k < stu.size(); k++) {
                    if (stu.getElemReg().get(k).getList().isIn(
                            lex.getElemReg().get(i))) {
                        out.println(stu.getElemReg().get(k).toString());
                    }
                }
                out.println("********************");
            }

        out.println("**End Registro**");
            out.println("**Fine**");

        } finally {
            // si cerca di chiudere comunque il file di output
            if (out != null)
                out.close();
        }

    }

    public void loadTesto() throws java.io.IOException {

        lex = new RegLezioni(); // inizializza alla lista vuota
        stu = new RegStudenti();

        BufferedReader in;
        // incapsula in BufferedReader un file aperto in lettura
        in = new BufferedReader(new FileReader(fileName + ".txt"));

        String line = in.readLine();
        if (!in.equals(null)) {

            String[] app;

            while (!line.equals("**Fine**")) {

                app = line.split(":", 5);
                this.name = app[1].trim();
                // imposto il nome al nuovo registro dopo averlo letto da
                // file
```

```
                while (!app[0].equals("Numero minimo di presenze ")) {
                    line = in.readLine();
                    app = line.split(":", 5);
                }
                app = line.split(":", 5);
                this.nObbligatorio = Integer.parseInt(app[1].trim());
                nObbligatorio = Integer.parseInt(app[1].trim());

                while (!line.equals("**Start Lezioni**")) {
                    line = in.readLine();
                }

                // imposto l'elenco delle lezioni, creando nuove istanze di
                // esse in reglezioni
                line = in.readLine();
                while (!line.equals("**End Lezioni**")) {
                    if (!line.equals("**End Lezioni**") && !line.equals(null)
                            && !line.equals("")) {
                        app = line.split(" - ");
                        lex.addLezione(new Lezione(Data.parseData(app[0].trim()),
                                app[1].trim(), app[2].trim(), new RegStudenti(), 0));
                        line = in.readLine();
                    } else
                        line = in.readLine();
                }

                while (!line.equals("**Start Studenti**")) {
                    line = in.readLine();

                }


                // imposto l'elenco degli studenti, creando nuove istanze di
                // esse in regstudenti
                line = in.readLine();
                while (!line.equals("**End Studenti**")) {
                    if (!line.equals("**End Studenti**") && !line.equals("")
                            && !line.equals(null)) {
                        app = line.split(" - ");
                        stu.addStudenti(new Studente(app[0].trim(), app[1].trim(),
Integer
                                .parseInt(app[2].trim()), new RegLezioni(), 0));
                        line = in.readLine();
                    } else
                        line = in.readLine();
                }
                while (!line.equals("**End Registro**")) {
                    if (!line.equals("**End Registro**")) {

                        while (!line.equals("**Start Registro**")) {
                            line = in.readLine();
                        }
```

```
                        line = in.readLine();
                        line = in.readLine();

                        if (line.equals("---Lezione:---")) {
                            line = in.readLine();
                            app = line.split(" - ", 3);

                            if (lex.isIn(new Lezione(Data.parseData(app[0]),
                                    app[1], app[2], new RegStudenti(), 0))) {
                                Lezione appoggio = new Lezione(Data
                                        .parseData(app[0].trim()), app[1].trim(), app
    [2].trim(),
                                        new RegStudenti(), 0);
                                int indexL = lex.getElemReg().indexOf(
                                        (Lezione) appoggio);
                                line = in.readLine();
                                line = in.readLine();
                                Studente appoggio2;
                                while (!line.equals("********************")) {
                                    if (!line.equals("********************")
                                            && !line.equals("")
                                            && !line.equals(null)) {
                                        app = line.split(" - ");
                                        appoggio2 = new Studente(app[0].trim(),
                                                app[1].trim(), Integer
                                                        .parseInt(app[2]),
                                                new RegLezioni(), 0);
                                        int index = stu.getElemReg().indexOf(
                                                (Studente) appoggio2);
                                        stu.getElemReg().get(index)
                                                .addElemList(appoggio);
                                        lex.getElemReg().get(indexL)
                                                .addElemList(appoggio2);

                                        line = in.readLine();
                                    } else
                                        line = in.readLine();
                                }

                            }
                        }
                        line = in.readLine();
                    }
                    line = in.readLine();
                    return;
                }
                if (nObbligatorio > lex.size())
                    this.nObbligatorio = lex.size();
            }

        }
        return;
    }
```

```
    public void loadLezioni(String fileName) {

        BufferedReader in;
        boolean ok = false;
        int i = 0;

        try {
            // incapsula in BufferedReader un file aperto in lettura
            in = new BufferedReader(new FileReader(fileName));
            String[] app;
            String line = in.readLine();

            if (!in.equals(null)) {

                while (!line.equals("**Fine**")) {

                    i++;

                    if (!line.equals("**Fine**") && !line.equals("")
                            && !line.equals(null)) {

                        app = line.split(" - ");

                        if (!getLezioni().isIn(
                                new Lezione(Data.parseData(app[0].trim()), app[1].
trim(),

                                app[2].trim(), new RegStudenti(), 0))) {

                            getLezioni().addLezione(
                                    new Lezione(Data.parseData(app[0]), app[1],

                                    app[2], new RegStudenti(), 0));
                        }
                        line = in.readLine();
                    } else
                        line = in.readLine();
                    if (i == 1000)
                        line = "**Fine**";
                }
            }
            ok = true;
        } catch (IOException e) {
            ok = false;
        } catch (NumberFormatException e) {
            ok = false;
        } catch (NullPointerException e) {
            System.out.println(" Dati presenti su file errati!!\n");
            ok = false;
        }
        if (ok)
            System.out
```

```
                    .println("\n\t\t****File Lezioni Caricato Correttamente****\n");
        else
            System.out
                    .println("\n\t\t****Errore durante il Caricamento del File
Lezioni****\n");
        return;
    }

    public void loadStudenti(String fileName) {

        BufferedReader in;
        int i = 0;
        boolean ok = true;
        try {
            // incapsula in BufferedReader un file aperto in lettura
            in = new BufferedReader(new FileReader(fileName));
            String[] app;
            String line = in.readLine();
            if (!in.equals(null)) {

                while (!line.equals("**Fine**")) {
                    i++;
                    if (!line.equals("**Fine**") && !line.equals("")
                            && !line.equals(null)) {
                        app = line.split(" - ");
                        if (!getStudenti()
                                .isIn(
                                        new Studente(app[0].trim(), app[1].trim(),
Integer
                                                .parseInt(app[2].trim()),
                                                new RegLezioni(), 0))) {
                            getStudenti().addStudenti(
                                    new Studente(app[0].trim(), app[1].trim(),
Integer
                                            .parseInt(app[2].trim()),
                                            new RegLezioni(), 0));
                        }
                        line = in.readLine();
                    } else
                        line = in.readLine();
                    if (i == 1000)
                        line = "**Fine**";
                }

            }
        } catch (IOException e) {
            ok = false;
        } catch (NumberFormatException e) {
            ok = false;
        } catch (NullPointerException e) {
            System.out.println(" Dati presenti su file errati!!\n");
            ok = false;
        } catch (DuplicatedStudentiException e) {
```

```
        }
        if (ok)
            System.out
                    .println("\n\t\t****File Lezioni Caricato Correttamente****\n");
        else
            System.out
                    .println("\n\t\t****Errore durante il Caricamento del File
Studenti****\n");
        return;
    }

    public void printReg() {

        System.out.println("                              Registro delle Presenze :\n")
;

        for (int i = 0; i < lex.size(); i++) {
            System.out.println("\t\t---Lezione:---");

            System.out.println(lex.getElemReg().get(i).toString());

            System.out.println("\t\t---Studenti:---");

            for (int k = 0; k < stu.size(); k++) {
                if (stu.getElemReg().get(k).getList().isIn(
                        lex.getElemReg().get(i))) {
                    System.out.println("\t" + (k + 1) + "--"
                            + stu.getElemReg().get(k).toString());
                }
            }
            System.out.println("********************");
        }

        System.out.println("**End Registro**");
        System.out.println("**Fine**");
    }

    public void setNumObbligatorio(int x) {
        this.nObbligatorio = x;
    }

    public RegistroFrequenzeImp loadBinario(String title) throws IOException,
            ClassNotFoundException {

        FileInputStream in = new FileInputStream(title);
        ObjectInputStream s = new ObjectInputStream(in);

        RegistroFrequenzeImp app = null;

        try {
            app = (RegistroFrequenzeImp) s.readObject();
        } catch (IOException e) {
            System.out.println("**Caricamento del file non riuscito\n");
```

```
            } catch (ClassNotFoundException e) {
                System.out.println("**Caricamento del file non riuscito\n");
            }
            ;
            return app;
        }

        public void saveBinario() throws IOException, ClassNotFoundException {
            FileOutputStream out = new FileOutputStream(this.name + ".bin");
            ObjectOutputStream s = new ObjectOutputStream(out);

            s.writeObject(this);

            s.flush();
        }

    }
```