```
import java.io.Serializable;
import java.util.List;
import java.util.Vector;

/**
 * Lista di elementi omogenei di tipo Lezione. E' modificabile e ordinata
 * secondo i criteri di ordinamento stabiliti per gli oggetti di tipo Lezione.
 * Gli elementi della lista differiscono fra loro per data,nome e argometo.
 *
 * L'oggetto e' serializabile.
 *
 * @author Filippo Fontanelli , Francesca Brogi
 */
public class RegLezioni implements Serializable {

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

    private static final long serialVersionUID = 3L;

    // Rappresentazione

    /**
     * Collezione di oggetti di tipo Lezioni. Lista di Lezioni ordinate per
     * data, nome e argometo.
     */
    private List<Lezione> list;

    // Costruttori

    /**
     * Costruttore.
     *
     * Inizializza <code>list</code> alla Lista Lezioni vuota.
     */
    public RegLezioni() {
        list = new Vector<Lezione>();
    }

    /**
     * Costruttore.
     *
     * Inizializza <code>list</code> alla Lista Lezioni. Inizializza
     * <code>list</code> alla Lista Studenti.
     *
     * @param list
     *            List<Studente> rappresentante la lista delle Lezioni.
     *
     * @exception <code>NullPointerException</code> se la lista &egrave;
     *            <code>null</code>.
     *
     */
    public RegLezioni(List<Lezione> list) {
        if (list == null) {
```

```
            throw new NullPointerException("RegLezioni::Costruttore::null");
        }
        this.list = list;
    }

    // Visitatori

    /**
     * Restituisce il numero di <Lezione> presenti in <code>list</code>.
     *
     * @return numero di elementi di list.
     */
    public int size() {
        return list.size();
    }

    /**
     * Restituisce la <code>list</code> di <Lezione>.
     *
     * @return List contentente la lista delle lezioni.
     */
    public List<Lezione> getElemReg() {
        return list;
    }

    /**
     * Restituisce la rappresentazione testuale della Lista Lezioni.
     *
     * @return String contentente le informazioni su tutti gli Lezioni di
     *         <code>list</code>.
     */
    public String toString() {
        String s = "";

        for (int i = 0; i < list.size(); i++)
            s += list.get(i).toString() + "\n";

        return s;
    }

    /**
     * E' un algoritmo di ricerca binaria sulla collezione <code>list</code>.
     * Stabilisce se <code>s</code> e' presente nella collezione, basandosi sui
     * criteri di uguaglianza degli oggetti di tipo <Lezione>. (Due oggetti
     * <Lezione> sono uguali quando hanno identico Data, Nome e Argomento).
     *
     * @param s
     *            oggetto di tipo <Lezione>. Dobbiamo stabilirne la presenza
     *            nella collezione.
     * @exception <code>NullPointerException</code> se la Lezione passata come
     *            parametro e' null.
     * @return <code>true</code> se <code>s</code> e' presente in list,
     *         <code>false</code> altrimenti.
```

```
     */
    public boolean isIn(Lezione s) throws NullPointerException {

        if (s == null)
            throw new NullPointerException("RegLezioni::isIn");

        if (list.size() == 0)
            return false;

        if (list.indexOf(s) < 0)
            return false;
        else
            return true;
    }

    // Modificatori

    /**
     * Aggiunge una Lezione passata come parametro alla collezione. Se
     * <code>l</code> e' null, solleva NullPointerException. Se <code>l</code>e'
     * gia' presente nella collezione, solleva DuplicatedLezioniException.
     * Altrimenti inserisce <code>l</code> in <code>list</code> mantenendo
     * l'ordinamento.
     *
     * @param l
     *            oggetto di tipo Studente che deve essere inserito nella
     *            collezione di dati.
     *
     * @exception <code>DuplicatedLezioniException</code> se nell'archivio e'
     *            presente un duplicato dello Studente che stiamo tentando di
     *            inserire.
     *
     * @exception <code>NullPointerException</code> se se la Lezione passata
     *            come parametro e' null.
     */
    public void addLezione(Lezione l) throws DuplicatedLezioniException,
            NullPointerException {

        if (l == null)
            throw new NullPointerException("RegLezioni::addLezione");
        // di seguito si fa il controllo sull'eventuale presenza di un duplicato
        if (list.indexOf((Lezione) l) >= 0)
            throw new DuplicatedLezioniException("RegLezioni::addLezione");

        if (list.size() == 0) {
            list.add(l);
        } else {
            if (list.get(size() - 1).compareTo(l) < 0) {
                list.add(size(), l);
            } else {
                for (int i = 0; i < list.size(); i++)
                    if (list.get(i).compareTo(l) > 0) {
                        list.add(i, l);
```

```
                    return; // una volta inserito l'elemento, si esce dal
                    // ciclo.
                }
            }
        }
    }

    /**
     * Rimuove la Lezione passata come parametro alla collezione. Se
     * <code>l</code> e' null, solleva NullPointerException. Se <code>l</code>
     * non e' presente nella collezione, solleva IllegalArgumentException.
     * Altrimenti elimina <code>l</code> da <code>list</code> mantenendo
     * l'ordinamento.
     *
     * @param l
     *            oggetto di tipo Studenti che deve essere inserito nella
     *            collezione di dati.
     *
     * @exception <code>NotExistentLezioniException</code> se nell'archivio non
     *            e' presente lo Studente che stiamo tentando di rimuovere.
     *
     * @exception <code>NullPointerException</code> se lo Studente passato come
     *            parametro e' null.
     */
    public void removeLezioni(Lezione l) throws NotExistentLezioniException,
            NullPointerException {

        int indexS;
        if (l == null)
            throw new NullPointerException("RegLezioni::removeStudenti");
        // di seguito si fa il controllo sull'eventuale presenza di un duplicato

        if (list.indexOf((Lezione) l) < 0)
            throw new NotExistentLezioniException("RegLezioni::removeStudenti");
        // ottengo l'indice dello studente e lo elimino

        indexS = list.indexOf((Lezione) l);
        list.remove(indexS);

    }

    /**
     * Modifica la Lezione relativa all'indice <code>index</code> passato come
     * parametro.
     *
     * @param index
     *            indice di riferimento della Lezione nella collezione.
     *
     */
    public void modify(int index) {

        System.out.println("Dettagli sulla lezione :");
        System.out.println(this.list.get(index).toString());
```

```
        System.out.println("Vuoi Modificare il Nome? (Y / N)");
        char ok = SafeInput.readChar();

        if (ok == 'y' || ok == 'Y') {
            System.out.println("Inserire il Nome :\n");
            String nome = SafeInput.readLineSafe();
            if (!nome.equals("")) {
                this.list.get(index).setNome(nome);
            }
        }

        System.out.println("Vuoi Modificare l'argomento? (Y / N)");
        ok = SafeInput.readChar();
        ;

        if (ok == 'y' || ok == 'Y') {
            System.out
                    .println("Inserire l'argomento ( * per lasciarlo invariato) :\n")
;
            String argomento = SafeInput.readLineSafe();
            if (!argomento.equals("")) {
                this.list.get(index).setArgomento(argomento);
            }
        }

        System.out.println("Vuoi Modificare la data? (y / n)");
        ok = SafeInput.readChar();

        if (ok == 'y' || ok == 'Y') {
            System.out.println("Inserire la data :\n");
            Data date = SafeInput.readDate();
            this.list.get(index).setData(date);
        }

        return;
    }

    /**
     * Cerca la Lezione relativa al Nome <code>n</code> passato come parametro.
     *
     * @param n
     *            nome di riferimento della Lezione nella collezione.
     *
     * @return int rappresentante l'indice di riferiemnto dela Lezione con Nome
     *         uguale a <code>n</code>.
     */
    public int cercaNome(String n) {
        boolean trovato = false;
        int i = 0;
        while (trovato == false && i < list.size()) {
            if (this.list.get(i).getNome().equals(n)) {
                trovato = true;
```

```
            } else
                i++;
        }
        if (i == this.list.size()) {
            return -1;
        } else
            return i;
    }

    /**
     * Cerca la Lezione relativa alla prima occorrenza della Data <code>n</code>
     * passato come parametro.
     *
     * @param n
     *            data di riferimento della Lezione nella collezione.
     *
     * @return int rappresentante l'indice di riferiemnto dela
     *         <code>Lezione</code> con Data uguale a <code>n</code>.
     */
    public int cercaData(Data n) {
        boolean trovato = false;
        int i = 0;
        while (trovato == false && i < list.size()) {
            if (this.list.get(i).getDate().equals((Data) n)) {
                trovato = true;
            } else
                i++;
        }
        if (i == this.list.size()) {
            return -1;
        } else
            return i;
    }

}
```