```
import java.io.Serializable; /*
 * To change this template, choose Tools | Templates
 * and open the template in the editor.
 */

/**
 * Tipo record modificabile relativo ai dati della Lezione. L'oggetto e'
 * serializabile.
 *
 * @author Filippo Fontanelli , Francesca Brogi
 */
public class Lezione implements Serializable {

    /**
     * Numero della lezione.
     */
    private static int num;
    /**
     * Data della lezione.
     */
    private Data date;
    /**
     * Nome della lezione.
     */
    private String name;
    /**
     * Argomento della lezione.
     */
    private String argomento;
    /**
     * Studenti presenti alla lezione.
     */
    private RegStudenti studenti;
    /**
     * Numero degli Studenti che sono stati presenti alla Lezione.
     */
    private int numPresenze;

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

    private static final long serialVersionUID = 5L;

    // Costruttori

    /**
     *
     * Costruttore.
     *
     * Crea una nuova lezione. Se un parametro e' vuoto o nullo viene sollevata
     * la relativa eccezione.
     *
     * @param date
     *             Data rappresentante la data.
```

```
     * @param name
     *            Stringa rappresentante il nome.
     * @param argomento
     *            Stringa rappresentante l'argomento.
     * @param stuPresenti
     *            RegStudenti rappresentante la lista degli Studenti.
     * @param numPresenze
     *            int rappresentante il numero di presenze.
     * @exception <code>NullPointerException</code> se il name o il argomento o
     *            date passati &egrave; <code>null</code>.
     *            <code>IllegalArgumentException</code> se il name o l'argomento
     *            passati sono uguali alla stringa vuota o se date e' unguale
     *            alla data vuota.
     */
    public Lezione(Data date, String name, String argomento,
            RegStudenti stuPresenti, int numPresenze)
            throws NullPointerException, IllegalArgumentException {

        if ((name == null) || (argomento == null) || date == null
                || stuPresenti == null) {
            throw new NullPointerException("Lezioni::Costruttore::null");
        }
        if ((name.equals("")) || (argomento.equals("")) || date == new Data()
                || numPresenze < 0) {
            throw new IllegalArgumentException(
                    "Lezioni::Costruttore::IllegalArgumentException");
        }
        num++;
        this.name = name;
        this.date = date;
        this.argomento = argomento;
        this.studenti = stuPresenti;
        this.numPresenze = numPresenze;
    }

    /**
     * Costruttore di una Lezione con dati vuoti.
     */
    public Lezione() {
        this.name = "";
        this.date = null;
        this.argomento = "";
        this.studenti = new RegStudenti();
        this.numPresenze = 0;
    }

    /**
     * Restituisce il num della lezione.
     *
     * @return int contente il numero.
     */
    public int getNum() {
        return num;
```

```
}

// Osservatori

/**
 * Restituisce la data della lezione.
 *
 * @return Data contente la data.
 */
public Data getDate() {
    return date;
}

/**
 * Restituisce il nome della lezione.
 *
 * @return String contente il nome.
 */
public String getNome() {
    return name;
}

/**
 * Restituisce l'argomento della lezione.
 *
 * @return String contente l'argometo.
 */
public String getArgomento() {
    return argomento;
}

/**
 * Restituisce la lista degli Studenti che erano presenti alla Lezione.
 *
 * @return RegStudenti contiene la lista degli Studenti.
 */
public RegStudenti getList() {
    return studenti;
}

/**
 * Restituisce il Numero di Studenti presenti alla Lezione.
 *
 * @return int contiene il numero delle presenze.
 */
public int getNumPresenze() {
    return this.numPresenze;
}

/**
 * Restituisce la rappresentazione testuale dell'oggetto.
 *
 *
```

```
     * Esempio: "15/11/2008 - Introduzione - Introduzione al lingaggio c++".
     */
    public String toString() {
        return date.toString() + " - " + name + " - " + argomento;
    }

    // Elaboratori

    /**
     * Compara oggetti di tipo Lezioni. Non distingue fra maiuscole e minuscole.
     *
     * @param obj
     *            oggetto di tipo Lezioni.
     *
     * @return un intero > 0 se la num corrente precede la Lezioni obj, 0 se
     *         sono uguali, un intero < 0 altrimenti cioe': > 0 se num > obj.num
     *         < 0 se num < obj.num SE num = obj.num: > 0 se nome > obj.nome < 0
     *         se nome < obj.nome SE nome = obj.nome: > 0 se matricola >
     *         obj.matricola < 0 se matricola < obj.matricola 0 se matricola =
     *         obj.matricola
     *
     * @throws <code>IllegalArgumentexception<code> se loggetto passato come
parametro e' <code>null<code>.
     */
    public int compareTo(Lezione obj) throws IllegalArgumentException {

        if (obj == null) {
            throw new IllegalArgumentException("Lezione::compareTo");
        }
        int diff = this.date.compareTo(obj.date);
        if (diff != 0) {
            return diff;
        } else {
            diff = name.compareToIgnoreCase(obj.name);
            if (diff != 0) {
                return diff;
            } else {
                diff = argomento.compareToIgnoreCase(obj.argomento);
                if (diff != 0) {
                    return diff;
                }
            }
        }
        return diff;

    }

    /**
     * Metodo costruito sulla base del compareTo.
     *
     * @param obj
     *            un oggetto qualsiasi.
     *
```

```
     * @return
     *         <code>true<code> se i due oggetti hanno stesso cognome, stesso nome e
stessa matricola
     *         altrimenti ritorna <code>false<code>.
     */
    public boolean equals(Object obj) {

        if (obj == null) {
            return false;
        }
        if (!(obj instanceof Lezione)) {
            return false;
        }
        return (this.compareTo((Lezione) obj) == 0);
    }

    // Modificatori

    /**
     * Aggiunge nella Lista degli Studenti il parametro passato, mantendo la
     * <code>Lista</code> ordinata secondo l'ordine ottenuto tramite il
     * compareTo e aumenta di un'unita' il numero delle presenze.
     *
     * @param s
     *            oggetto di tipo Studente da inserire, se non gia' presente e
     *            <em>diverso</em> da null.
     *
     * @exception <code>IllegalArgumentException</code> se l'argomento passato
     *            &egrave; <code>null</code>.
     *            <code>DuplicatedLezioniException</code> se l'argomento passato
     *            e' gia presente nell'elenco.
     */
    public void addElemList(Studente s) throws DuplicatedLezioniException,
            NullPointerException {

        if (s == null)
            throw new NullPointerException("Lezioni::addElemList");
        // di seguito si fa il controllo sull'eventuale presenza di un duplicato
        if (studenti.getElemReg().indexOf((Studente) s) >= 0)
            throw new DuplicatedLezioniException("Lezioni::addElemList");
        this.numPresenze++;
        if (studenti.size() == 0) {
            studenti.getElemReg().add(s);
        } else {
            if (studenti.getElemReg().get(studenti.size() - 1).compareTo(s) < 0) {
                studenti.getElemReg().add(studenti.size(), s);
            } else {
                for (int i = 0; i < studenti.size(); i++)
                    if (studenti.getElemReg().get(i).compareTo(s) > 0) {
                        studenti.getElemReg().add(i, s);
                        return; // una volta inserito l'elemento, si esce dal
                        // ciclo.
                    }
```

```
        }
    }
}

/**
 * Rimuove dalla Lista degli Studenti il parametro passato, e diminuisce di
 * un'unita' il numero delle presenze.
 *
 * @param s
 *            oggetto di tipo Studente da cancellare, se presente e
 *            <em>diverso</em> da null.
 *
 * @exception <code>IllegalArgumentException</code> se l'argomento passato
 *            &egrave; <code>null</code>. <code>NullPointerException</code>
 *            se l'argomento passato non e' presente nell'elenco.
 */
public void removeElemList(Studente s) throws NotExistentStudentiException,
        NullPointerException {
    if (s == null)
        throw new NullPointerException("Lezioni::removeElemList");
    // di seguito si fa il controllo sull'eventuale presenza di un duplicato
    if (!studenti.getElemReg().contains((Studente) s))
        throw new NotExistentStudentiException("Lezioni::removeElemList");
    studenti.removeStudenti(s);
    this.numPresenze--;
}

/**
 * Modifica la Data della Lezione.
 *
 * @param obj
 *            un oggetto di tipo Data.
 */
public void setData(Data obj) {
    this.date = obj;
}

/**
 * Modifica il Nome dello Studente.
 *
 * @param obj
 *            un oggetto di tipo String.
 */
public void setNome(String obj) {
    this.name = obj;
}

/**
 * Modifica l'Argomento della Lezione.
 *
 * @param obj
 *            un oggetto di tipo String.
 */
```

```
    public void setArgomento(String obj) {
        this.argomento = obj;
    }
}
```