```
import java.io.Serializable;

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

    /**
     * Nome dello studente.
     */
    private String nome;
    /**
     * Cognome dello studente.
     */
    private String cognome;
    /**
     * Matricola dello studente.
     */
    private int matricola;
    /**
     * Lezioni a cui lo studente e' stato presente.
     */
    private RegLezioni lezioni;
    /**
     * Numero delle Lezioni a cui lo studente era presente.
     */
    private int numPresenze;

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

    private static final long serialVersionUID = 1L;

    // Costruttori
    /**
     *
     * Costruttore.
     *
     * Crea un nuovo Studente. Se un parametro e' vuoto o nullo viene sollevata
     * la relativa eccezione.
     *
     * @param nome
     *            Stringa rappresentante il nome.
     * @param cognome
     *            Stringa rappresentante il cognome.
     * @param matricola
     *            int rappresentante la matricola.
     * @param lezioni
     *            RegLezioni rappresentante la lista delle Lezioni.
     * @param numPresenze
     *            int rappresentante il numero di presenze.
```

```
     * @exception <code>NullPointerException</code> se il nome o il cognome o
     *            lezioni passati &egrave; <code>null</code>.
     *            <code>IllegalArgumentException</code> se il nome o il cognome
     *            passati sono uguali alla stringa vuota o se la matricola e'
     *            <=0 o il numPresenze e' <0.
     */
    public Studente(String cognome, String nome, int matricola,
            RegLezioni lezioni, int numPresenze) throws NullPointerException,
            IllegalArgumentException {
        if (nome == null || cognome == null || lezioni == null) {
            throw new NullPointerException("Studenti::Costruttore::null");
        }
        if (nome.equals("") || cognome.equals("") || matricola <= 0
                || numPresenze < 0) {
            throw new IllegalArgumentException(
                    "Studenti::Costruttore::IllegalArgumentException");
        }
        this.nome = nome;
        this.cognome = cognome;
        this.matricola = matricola;
        this.lezioni = new RegLezioni();
        this.numPresenze = numPresenze;
    }

    /**
     * Costruisce uno Studente con dati vuoti.
     */
    public Studente() {
        this.nome = "";
        this.cognome = "";
        this.matricola = 0;
        this.lezioni = new RegLezioni();
        this.numPresenze = 0;
    }

    // Osservatori

    /**
     * Restituisce il nome dello Studente.
     *
     * @return String contente il nome.
     */
    public String getNome() {
        return nome;
    }

    /**
     * Restituisce il cognome dello Studente.
     *
     * @return String contente il cognome.
     */
    public String getCognome() {
        return cognome;
```

```
    }

    /**
     * Restituisce la matricola dello Studente.
     *
     * @return int contente la matricola.
     */
    public int getMatricola() {
        return matricola;
    }

    /**
     * Restituisce la lista delle Lezioni a cui lo Studente e' stato presente.
     *
     * @return RegLezioni contiene la lista delle Lezioni.
     */
    public RegLezioni getList() {
        return lezioni;
    }

    /**
     * Restituisce il Numero di presenze dello Studente.
     *
     * @return int contiene il numero delle presenze.
     */
    public int getNumPresenze() {
        return this.numPresenze;
    }

    /**
     * Restituisce la rappresentazione testuale dell'oggetto.
     *
     *
     * Esempio: "Pagni Andrea 40002".
     */
    public String toString() {
        return cognome + " - " + nome + " - " + matricola;
    }

    // Elaboratori

    /**
     * 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 Studente)) {
            return false;
        }
        return (this.compareTo((Studente) obj) == 0);
    }

    /**
     * Compara oggetti di tipo Studenti. Non distingue fra maiuscole e
     * minuscole.
     *
     * @param obj
     *            oggetto di tipo Studenti.
     *
     * @return un intero > 0 se la Studenti corrente precede lo Studenti obj, 0
     *         se sono uguali, un intero < 0 altrimenti cioe': > 0 se cognome
     *         >obj.cognome < 0 se cognome < obj.cognome SE cognome =
     *         obj.cognome: > 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(Studente obj) throws NullPointerException {

        if (obj == null) {
            throw new NullPointerException("Studenti::compareTo");
        }
        int diff = cognome.compareToIgnoreCase(obj.cognome);
        if (diff != 0) {
            return diff;
        } else {
            diff = nome.compareToIgnoreCase(obj.nome);
            if (diff != 0) {
                return diff;
            } else {
                diff = this.matricola - obj.matricola;
                if (diff != 0) {
                    return diff;
                }
            }
        }
        return diff;
    }

    // Modificatori
    /**
     * Aggiunge nella Lista delle lezioni il parametro passato, mantendo la
     * lista ordinata secondo l'ordine ottenuto tramite il compareTo e aumenta
```

```
     * di un'unita' il numero delle presenze.
     *
     * @param s
     *            oggetto di tipo Lezione 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(Lezione s) throws DuplicatedLezioniException,
            NullPointerException {

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

    /**
     * Rimuove dalla Lista delle lezioni il parametro passato, e diminuisce di
     * un'unita' il numero delle presenze.
     *
     * @param s
     *            oggetto di tipo Lezione 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(Lezione s) throws NotExistentLezioniException,
            IllegalArgumentException {
        if (s == null)
            throw new IllegalArgumentException("Studnete::removeElemList");
        // di seguito si fa il controllo sull'eventuale presenza di un duplicato
```

```
            if (lezioni.getElemReg().indexOf((Lezione) s) < 0)
                throw new NotExistentLezioniException("Studnete::removeElemList");
            lezioni.removeLezioni(s);
            this.numPresenze--;
        }

        /**
         * Modifica il Cognome dello Studente.
         *
         * @param obj
         *            un oggetto di tipo String.
         */
        public void setCognome(String obj) {
            cognome = obj;
        }

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

        /**
         * Modifica la Matricola dello Studente.
         *
         * @param obj
         *            un oggetto di tipo int.
         */
        public void setMatricola(int obj) {
            matricola = obj;
        }
    }
```