```
/**
 * Classe di Test.
 *
 * @author Filippo Fontanelli , Francesca Brogi
 *
 */
public class testClassStudenti {

    /**
     * @param args
     */
    public static void main(String[] args) {
        // TODO Auto-generated method stub
        try {
            System.out
                    .println("\n================================\nProvo la classe
'Studente'...\n================================");
            Studente[] dip = new Studente[5];
            System.out.println(".");
            // PROVO COSTRUTTORI
            try {
                // Caso Null
                dip[0] = new Studente(null, "Rossi", 25, new RegLezioni(), 10);
                System.out
                        .println("Attento: il costrutture deve lanciare una
'NullPointerException' \nse il nome dello studente e' null: correggi e riprova.");
            } catch (NullPointerException e) {
            }

            try {
                dip[0] = new Studente("Mario", null, 25, new RegLezioni(), 10);
                System.out
                        .println("Attento: il costrutture deve lanciare una
'NullPointerException' \nse il cognome dello studente e' null: correggi e riprova.");
            } catch (NullPointerException e) {
            }

            try {
                dip[0] = new Studente("Mario", "Rossi", 25, null, 10);
                System.out
                        .println("Attento: il costrutture deve lanciare una
'NullPointerException' \nse il registro lezioni e' null: correggi e riprova.");
            } catch (NullPointerException e) {
            }
            // Caso Stringa vuota ed interi < 0
            try {
                dip[0] = new Studente("Mario", "", 25, new RegLezioni(), 10);
                System.out
                        .println("Attento: il costrutture deve lanciare una
'IllegalArgumentException' \nse il cognome dello studente e' la stringa vuota:
correggi e riprova.");
            } catch (IllegalArgumentException e) {
            }
```

```
            try {
                dip[0] = new Studente("", "Rossi", 25, new RegLezioni(), 10);
                System.out
                        .println("Attento: il costrutture deve lanciare una
'IllegalArgumentException' \nse il nome dello studente e' la stringa vuota: correggi
e riprova.");
            } catch (IllegalArgumentException e) {
            }

            try {
                dip[0] = new Studente("Mario", "Rossi", -2, new RegLezioni(),
                        10);
                System.out
                        .println("Attento: il costrutture deve lanciare una
'IllegalArgumentException' \nse la matricola e' minore di 0: correggi e riprova.");
            } catch (IllegalArgumentException e) {
            }

            try {
                dip[0] = new Studente("Mario", "Rossi", 2, new RegLezioni(), -1);
                System.out
                        .println("Attento: il costrutture deve lanciare una
'IllegalArgumentException' \nse il numero presenze e' minore di 0: correggi e riprova
.");

            } catch (IllegalArgumentException e) {
            }

            System.out.println("..");

            // PROVO GETTERS
            dip[0] = new Studente("Mario", "Rossi", 2, new RegLezioni(), 5);
            if (!dip[0].getNome().equals("Mario"))
                System.out
                        .println("Attento: il valore restituito da getNome()()\n non
corrisponde al valore passato al costruttore");
            if (!dip[0].getCognome().equals("Rossi"))
                System.out
                        .println("Attento: il valore restituito da getCognome()\n non
corrisponde al valore passato al costruttore");
            if (dip[0].getMatricola() != 2)
                System.out
                        .println("Attento: il valore restituito da getMatricola()\n
non corrisponde al valore passato al costruttore");
            if (dip[0].getNumPresenze() != 5)
                System.out
                        .println("Attento: il valore restituito da getNumPresenze()\n
non corrisponde al valore passato al costruttore");

            // PROVO COMPARETO
            dip[0] = new Studente("Mario", "Rossi", 25, new RegLezioni(), 1); //
            dip[1] = new Studente("Mario", "Rossi", 27, new RegLezioni(), 1);//
```

```
            dip[2] = new Studente("Mario", "Rossi", 28, new RegLezioni(), 1); // 14
            dip[3] = new Studente("Mario", "Rossi", 29, new RegLezioni(), 1); // 10
            dip[4] = new Studente("Mario", "Rossi", 29, new RegLezioni(), 1); // 10

            System.out.println("...");

            if (dip[1].compareTo(dip[2]) >= 0 || dip[2].compareTo(dip[1]) <= 0
                    || dip[0].compareTo(dip[1]) >= 0
                    || dip[1].compareTo(dip[0]) <= 0
                    || dip[2].compareTo(dip[3]) >= 0
                    || dip[3].compareTo(dip[2]) <= 0
                    || dip[3].compareTo(dip[4]) != 0)
                System.out
                        .println("Attento: sembra che compareTo() non confronti\n
correttamente gli Studenti");

            System.out.println("....");

            // PROVO EQUALS
            try {
                if (dip[0].equals(null) || dip[0].equals("ciao")
                        || dip[0].equals(dip[1]) || dip[0].equals(dip[2])
                        || dip[0].equals(dip[3]) || !dip[3].equals(dip[4]))
                    System.out
                            .println("Attento: sembra che equals() non confronti\n
correttamente gli account");
            } catch (RuntimeException e) {
                System.out
                        .println("Attento: ho provato il metodo equals, e ho
catturato una "
                                + e
                                + ".\n Equals non deve mai lanciare un'eccezione!!!
Controlla...");
            }

            System.out.println(".....");

            // PROVO TOSTRING
            String s = dip[2].toString();
            if (s == null || s.indexOf("Mario") == -1
                    || s.indexOf("Rossi") == -1 || s.indexOf("28") == -1)
                System.out
                        .println("Attento: sembra che toString() non restituisca\n
tutte le informazioni richieste");

            System.out.println("......");

            // PROVO addElemList
            try {
                dip[0].addElemList(null);
                System.out
                        .println("Attento: il metodo addElemList deve lanciare una
'IllegalArgumentException' \nse tento di inserire una lezione null: correggi e
```

```
riprova. ");

            } catch (NullPointerException e) {
            }
            Lezione app = new Lezione(new Data(12, 10, 2008), "Prima", "dopo",
                    new RegStudenti(), 0);
            try {
                dip[0].addElemList(app);
                dip[0].addElemList(app);
                System.out
                        .println("Attento: il metodo addElemList deve lanciare una
'DuplicatedLezioniException' \nse tento di inserire una lezione che e' gia presente :
correggi e riprova. ");
            } catch (DuplicatedLezioniException e) {
            }

            Lezione app2 = new Lezione(new Data(15, 10, 2008), "Adesso",
                    "Subito", new RegStudenti(), 0);
            dip[0].addElemList(app2);
            if (dip[0].getList().getElemReg().indexOf((Lezione) app2) < dip[0]
                    .getList().getElemReg().indexOf((Lezione) app))
                System.out
                        .println("Attento: il metodo addElemList deve mantenere l
'elenco delle lezioni ordinato : correggi e riprova. ");

            System.out.println(".......");

            // PROVO removeElemList
            try {
                dip[0].removeElemList(null);
                System.out
                        .println("Attento: il metodo removeElemList deve lanciare una
'IllegalArgumentException' \nse tento di rimuovere una lezione null: correggi e
riprova. ");

            } catch (IllegalArgumentException e) {
            }
            Lezione rem = new Lezione(new Data(29, 10, 2008), "Prima", "dopo",
                    new RegStudenti(), 0);
            try {
                dip[0].removeElemList(rem);
                System.out
                        .println("Attento: il metodo removeElemList deve lanciare una
'NullPointerException' \nse tento di rimuovere una lezione che non e' presente nell
'archivio : correggi e riprova. ");
            } catch (NotExistentLezioniException e) {
            }

            dip[0].addElemList(rem);
            int prima = dip[0].getNumPresenze();
            dip[0].removeElemList(rem);
            int dopo = dip[0].getNumPresenze();
            if (prima != (dopo + 1))
```

```
                System.out
                        .println("Attento: il metodo removeElemList deve diminuire il
numero delle presenze dopo ogni cancellazione : correggi e riprova. ");

            /**
             * // PROVO penale() try{ if (dip[2].penale() != 0.0 ||
             * dip[3].penale() != 1 || dip[1].penale() != 1.5) die("Attento:
             * sembra che penale() non si comporti come richiesto: controlla e
             * riprova..."); } catch (RuntimeException e) { die("Attento: ho
             * provato il metodo penale(), e ho catturato una " + e + ".\n
             * Penale() non dovrebbe lanciare un'eccezione. Controlla..."); }
             *
             *
             */
        } catch (Exception ecc) {
            System.out
                    .println("\n OOPS!!! Stavo provando la tua classe 'Studente', \
nma si e' verificata un'eccezione:");
            ecc.printStackTrace();
            System.exit(-1);
        }

        System.out.println("OK: non sono riuscito a trovare alcun errore.");
        System.out.println("===========================================");

    }
}
```