UNIVERSITÀ
DEGLI STUDI DI BARI
ALDO MORO

A&G

Data

30-ott-2023

Esame:

INTEGRAZIONE E TEST DI SISTEMI SOFTWARE

A.A 2022-2023

Realizzato da:

Andrea Gioia 740825 -- a.gioia24@studenti.uniba.it

Gabriele Antonio dell'Isola 735392 -- g.dellisola@studenti.uniba.it

UNIVERSITÀ
DEGLI STUDI DI BARI
ALDO MORO

Sommario

UNIVERSITÀ DEGLI STUDI DI BARI ALDO MORO

HOMEWORK 1

Codice assegnato:

```java
class Student {
    String firstName;
    String lastName;
    double averageGrade;
    double lastGrade;

    public Student(String firstName, String lastName, double averageGrade, double lastGrade) {
        if (firstName == null || lastName == null) {
            throw new IllegalArgumentException("Il nome e il cognome dello studente non possono essere nulli.");
        }
        if (Double.isNaN(averageGrade) || Double.isNaN(lastGrade)) {
            throw new IllegalArgumentException("La media dei voti e l'ultimo voto dello studente non possono essere nulli.");
        }

        this.firstName = firstName;
        this.lastName = lastName;
        this.averageGrade = averageGrade;
        this.lastGrade = lastGrade;
    }

    public boolean nuovaMedia() {
        if (lastGrade >= 18 && lastGrade <= 30) {
            double newAverage = (averageGrade + lastGrade) / 2.0;
            return newAverage >= averageGrade;
        }
        return false;
    }



    public String toString() {
        DecimalFormat df = new DecimalFormat("0.00");
        return firstName + " " + lastName + ": Media " + df.format(averageGrade) + ", Ultimo voto " + lastGrade;
    }
}
```

```java
public class MedieVoti {
    public static void main(String[] args) {
        String[] firstNames = {
                "MARIO", "LUCA", "MARCO", "GABRIELE", "ANDREA",
                "LETIZIA", "CLAUDIA", "PAOLO", "ENRICO", "VINCENZO"
        };

        String[] lastNames = {
                "ROSSI", "BIANCHI", "TOGNI", "ISOLA", "GIOIA",
                "GUERRA", "GIALLI", "VERDI", "FERMI", "TESTA"
        };

        Student[] students = new Student[10];

        for (int i = 0; i < students.length; i++) {
            double averageGrade = 18 + Math.random() * 12;
            double lastGrade = 18 + Math.random() * 12;
            averageGrade = Math.round(averageGrade * 100.0) / 100.0;
            lastGrade = Math.round(lastGrade * 100.0) / 100.0;
            students[i] = new Student(firstNames[i], lastNames[i], averageGrade, lastGrade);
        }

        for (Student student : students) {
            System.out.println(student);
            System.out.println("Nuova media supera quella precedente? " + student.nuovaMedia());
            System.out.println(); // Empty line for readability
        }
    }
}
```

1) Understanding the requirements

Obiettivo della classe

La classe crea e stampa una serie di 10 studenti con due valori numerici generati randomicamente, la media dello studente e l'ultimo voto ricevuto dallo studente, in seguito esegue un semplice controllo per verificare che la nuova media risultante superi o meno quella creata in precedenza. Quindi testeremo le classi Student e nuovaMedia.

Input

- String firstName: testo rappresentante il nome dello studente
- String lastName: testo rappresentante il cognome dello studente
- Double averageGrade: valore della media dello studente
- Double lastGrade: valore dell'ultimo voto dello studente

Output

- Il metodo restituisce l'elenco degli studenti con il valore della media aggiornata, l'ultimo voto e un valore booleano true nel caso in cui la nuova media superi la precedente o false in caso contrario, l'ultimo voto deve rientrare nel range dei voti universitari 18-30.

2) Explore what the program does for various inputs

In questa fase sono state effettuate delle esplorazioni, con diversi input per capire al meglio come il programma si comporta in diversi casi

In questo caso abbiamo un voto finale superiore al limite massimo consentito, cioè 30

```
@Test
void testNuovaMediaConVotoSuperioreUgualeA30() {
    // Verifica che il metodo nuovaMedia() restituisca false quando l'ultimo voto è al di sopra del limite superiore (30)
    student = new Student("Mario", "Rossi", 25.0, 31.0);
    assertFalse(student.nuovaMedia());
```

In questo caso, invece abbiamo un voto finale inferiore al limite minimo consentito, cioè 18

```
@Test
void testNuovaMediaConVotoInferioreUgualeA18() {
    // Verifica che il metodo nuovaMedia() restituisca false quando l'ultimo voto è al di sotto del limite inferiore (18)
    student = new Student("Mario", "Rossi", 25.0, 17.0);
    assertFalse(student.nuovaMedia());
```

In questi due casi invece abbiamo il voto finale che rientra nei casi limite, 30 e 18

```
// Verifica che il metodo nuovaMedia() restituisca true quando l'ultimo voto è uguale al limite superiore (30)
student = new Student("Mario", "Rossi", 25.0, 30.0);
assertTrue(student.nuovaMedia());
```

```
// Verifica che il metodo nuovaMedia() restituisca true quando l'ultimo voto è uguale al limite inferiore (18)
student = new Student("Mario", "Rossi", 18.0, 18.0);
assertTrue(student.nuovaMedia());
```

In questo caso l'ultimo voto rientra nel range consentito 18-30

```
@Test
void testNuovaMediaUltimoVotoConsentito() {
    // Verifica che il metodo nuovaMedia() restituisca false quando l'ultimo voto è nell'intervallo consentito
    student = new Student("Mario", "Rossi", 25.0, 26.0);
    assertTrue(student.nuovaMedia());
```

3) Explore inputs, outputs and identify partitions

Input individuati

Text	Text	Number	Number
String firstName	String lastName	Double averageGrade	Double lastGrade
Nome	Cognome	Valore compreso tra 18-30	Valore compreso tra 18-30
Null	Null	Valore non compreso tra 18-30	Valore non compreso tra 18-30
		NaN	NaN

Combinazioni possibili

#	Nome	Cognome	Media	Ultimo Voto tra 18-30
1	si	si	si	Si
2	si	si	NaN	Si
3	si	si	si	NaN
4	si	si	NaN	NaN

5	Null	si	si	Si
6	si	Null	si	Si
7	Null	Null	Si	Si
8	Si	Null	NaN	Si
9	si	Null	SI	NaN
10	Si	Null	NaN	NaN
11	Null	Null	NaN	Si
12	Null	Null	Si	NaN
13	Si	Si	Si	NaN
14	Null	Si	Si	NaN
15	Si	Si	NaN	NaN
16	Null	Null	No	No
17	Si	Si	No	Si
18	Si	Si	No	No
19	Null	Si	No	Si
20	Si	Null	No	Si
21	Null	Null	No	Si
22	Si	Null	No	Si
23	Null	Null	No	No
24	Null	Null	NaN	NaN

Queste sono alcune delle possibili combinazioni, non verranno testate tutte ma possono essere prese come esempio per trovare i casi limite da testare.

Output attesi

Testuale	Numerico
nome	media
cognome	Ultimo voto

4) Boundary Case

In questo caso i casi limite riguardano soprattutto i valori dell'ultimo voto, La funzione NuovaMedia() usa il valore dell'ultimo voto per verificare se la nuova media superi la precedente, i casi limite quindi sono rappresentati dall'ultimo voto uguale o superiore a 30, inferiore o uguale a 18. Nel caso del costruttore invece i casi limite sono rappresentati da firstName o lastName Null o i valori di averageGrade o lastGrade come NaN (Not a Number). Stando a questi casi limite possiamo definire i test da eseguire:

5) Device test cases

T1	La Stringa firstName è Null
T2	La stringa lastName è Null
T3	Il double averageGrade è NaN
T4	Il double lastGrade è NaN
T5	lastGrade >= 30
T6	lastGrade <= 18
T7	18 <= lastGrade <= 30

6) Automate test case

```
@Test
//T1
void testNomeNull() {
    // Verifica che il costruttore sollevi un'eccezione quando il nome è nullo
    assertThrows(IllegalArgumentException.class, () -> new Student(null, "Rossi", 25.0, 20.0));
}

@Test
//T2
void testCognomeNull() {
    // Verifica che il costruttore sollevi un'eccezione quando il cognome è nullo
    assertThrows(IllegalArgumentException.class, () -> new Student("Mario", null, 25.0, 20.0));
}

@Test
//T3
void testMediaNaN() {
    // Verifica che il costruttore sollevi un'eccezione quando la media dei voti è NaN
    assertThrows(IllegalArgumentException.class, () -> new Student("Mario", "Rossi", Double.NaN, 20.0));
}

@Test
//T4
void testUltimoVotoNaN() {
    // Verifica che il costruttore sollevi un'eccezione quando l'ultimo voto è NaN
    assertThrows(IllegalArgumentException.class, () -> new Student("Mario", "Rossi", 25.0, Double.NaN));
}
```

```
@Test
//T5
void testNuovaMediaConVotoSuperioreUgualeA30() {
    // Verifica che il metodo nuovaMedia() restituisca false quando l'ultimo voto è al di sopra del limite superiore (30)
    student = new Student("Mario", "Rossi", 25.0, 31.0);
    assertFalse(student.nuovaMedia());

    // Verifica che il metodo nuovaMedia() restituisca true quando l'ultimo voto è uguale al limite superiore (30)
    student = new Student("Mario", "Rossi", 25.0, 30.0);
    assertTrue(student.nuovaMedia());
}

@Test
//T6
void testNuovaMediaConVotoInferioreUgualeA18() {
    // Verifica che il metodo nuovaMedia() restituisca false quando l'ultimo voto è al di sotto del limite inferiore (18)
    student = new Student("Mario", "Rossi", 25.0, 17.0);
    assertFalse(student.nuovaMedia());

    // Verifica che il metodo nuovaMedia() restituisca true quando l'ultimo voto è uguale al limite inferiore (18)
    student = new Student("Mario", "Rossi", 18.0, 18.0);
    assertTrue(student.nuovaMedia());
}

@Test
//T7
void testNuovaMediaUltimoVotoConsentito() {
    // Verifica che il metodo nuovaMedia() restituisca false quando l'ultimo voto è nell'intervallo consentito
    student = new Student("Mario", "Rossi", 25.0, 26.0);
    assertTrue(student.nuovaMedia());
}
```

Casi di test

Test Scenario ID	nullNaNCase	**Test Case Description**	Con questa suite di test testiamo i casi in cui siano presenti parametri null o NaN

Test:

#	**Inputs**	**Expected Output**	**Actual Output**	**Test Result**
1	firstName = null	`IllegalArgumentException`	`IllegalArgumentException`	Pass
2	lastName = null	`IllegalArgumentException`	`IllegalArgumentException`	Pass
3	averageGrade = NaN	`IllegalArgumentException`	`IllegalArgumentException`	Pass
4	lastGrade = NaN	`IllegalArgumentException`	`IllegalArgumentException`	Pass

Test Scenario ID	lastGradeNonConforme	**Test Case Description**	Con questa suite di test testiamo i casi in cui il parametro lastGrade non è conforme alle aspettative

Test:

#	**Inputs**	**Expected Output**	**Actual Output**	**Test Result**
5	lastGrade >= 30	`False se lastGrade > 30` True se lastGrade = 30	`False se lastGrade > 30` True se lastGrade = 30	Pass
6	lastGrade <= 18	`False se lastGrade < 18` True se lastGrade = 18	`False se lastGrade < 18` True se lastGrade = 18	Pass
7	18 <= lastGrade <= 18	`True`	`True`	Pass

7) Augment the test suite with creativity and experience

In questa fase abbiamo riscontrato come mancasse un test che comprendesse una combinazione dei casi errati che abbiamo implementato di seguito nel Test 8

```
@Test
//T8
void testStudentCostruttoreConNomeENaN() {
    // Verifica che il costruttore gestisca correttamente il caso con nome nullo e valutazioni NaN
    assertThrows(IllegalArgumentException.class, () -> new Student(null, null, Double.NaN, Double.NaN));
}
```

Test Scenario ID	NullNaNCombined	**Test Case Description**	Con questo test verifichiamo il caso in cui il costruttore abbia sia dei valori null che valori NaN

Test:

#	**Inputs**	**Expected Output**	**Actual Output**	**Test Result**
8	firstName= null	`IllegalArgumentException`	`IllegalArgumentException`	Pass

	lastName = null averageGrade = NaN lastGrade = NaN			

HOMEWORK 2

Come primo passo in questo secondo homework abbiamo eseguito una Code Coverage del codice del primo homework constatando una copertura al 72% in quanto il generatore randomico dei voti e il costruttore della classe student non risultavano coperti, la code coverage ha evidenziato come siamo riusciti a testare con successo le classi Student e nuovaMedia come da programma

MedieVoti.java

```java
package CodiceMedie;

import java.text.DecimalFormat;

class Student {
    String firstName;
    String lastName;
    double averageGrade;
    double lastGrade;

    public Student(String firstName, String lastName, double averageGrade, double lastGrade) {
        if (firstName == null || lastName == null) {
            throw new IllegalArgumentException("Il nome e il cognome dello studente non possono essere nulli.");
        }
        if (Double.isNaN(averageGrade) || Double.isNaN(lastGrade)) {
            throw new IllegalArgumentException("La media dei voti e l'ultimo voto dello studente non possono essere nulli.");
        }

        this.firstName = firstName;
        this.lastName = lastName;
        this.averageGrade = averageGrade;
        this.lastGrade = lastGrade;
    }

    public boolean nuovaMedia() {
        if (lastGrade >= 18 && lastGrade <= 30) {
            double newAverage = (averageGrade + lastGrade) / 2.0;
            return newAverage >= averageGrade;
        }
        return false;
    }

    public String toString() {
        DecimalFormat df = new DecimalFormat("0.00");
        return firstName + " " + lastName + ": Media " + df.format(averageGrade) + ", Ultimo voto " + lastGrade;
    }
}

public class MedieVoti {
    public static void main(String[] args) {
        String[] firstNames = {
                "MARIO", "LUCA", "MARCO", "GABRIELE", "ANDREA",
                "LETIZIA", "CLAUDIA", "PAOLO", "ENRICO", "VINCENZO"
        };

        String[] lastNames = {
                "ROSSI", "BIANCHI", "TOGNI", "ISOLA", "GIOIA",
                "GUERRA", "GIALLI", "VERDI", "FERMI", "TESTA"
        };

        Student[] students = new Student[10];

        for (int i = 0; i < students.length; i++) {
            double averageGrade = 18 + Math.random() * 12;
            double lastGrade = Math.random() * 30;
            averageGrade = Math.round(averageGrade * 100.0) / 100.0;
            lastGrade = Math.round(lastGrade * 100.0) / 100.0;
            students[i] = new Student(firstNames[i], lastNames[i], averageGrade, lastGrade);
        }

        for (Student student : students) {
            System.out.println(student);
            System.out.println("Nuova media supera quella precedente? " + student.nuovaMedia());
            System.out.println(); // Empty line for readability
        }
    }
}
```

Abbiamo deciso di testare anche la classe MedieVoti con un test aggiuntivo, tuttavia, testare il main MedieVoti risulta complicato dato che gli input contenuti sono generati casualmente runtime. Invece di

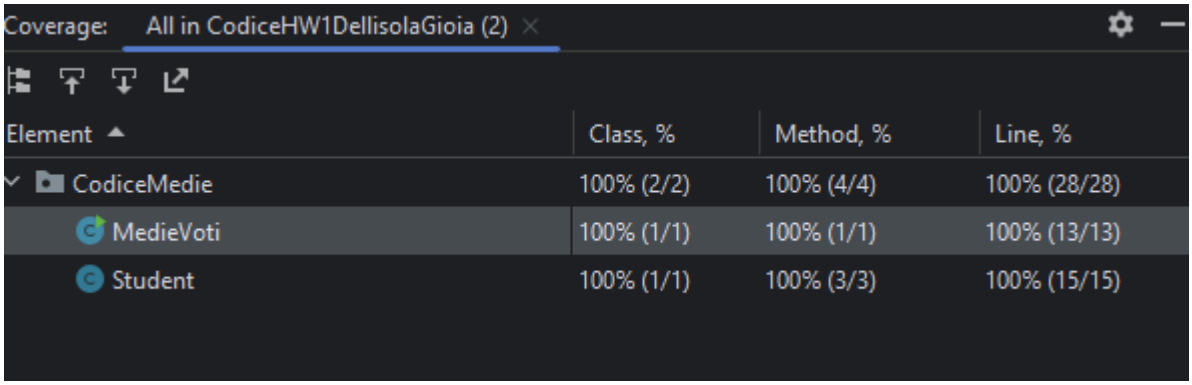
testare il metodo in sé abbiamo deciso di testare se il metodo non sollevi eccezioni anomale:

```java
@Test
void testMedieVotiNoEccezioni() {

    // MedieVoti non deve sollevare eccezioni
    assertDoesNotThrow(() -> MedieVoti.main(new String[]{}));
}

private void assertDoesNotThrow(Runnable runnable) {
    try {
        runnable.run();
    } catch (Exception e) {
        fail("Expected no exception, but got: " + e.getClass());
    }
}
```

Una volta aggiunto il test abbiamo rieffettuato l'analisi di code coverage che ha evidenziato una copertura del 100%



Il Codice dell'Homework 2

```java
public class Persona {
    protected int eta;
    protected boolean haPatente;
    protected boolean haAssicurazione;

    public Persona() {
        Random random = new Random();
        eta = random.nextInt(62)+ 18; // Genera un'età casuale tra 18 e 79
        haPatente = random.nextBoolean(); // Genera casualmente true o false per la patente
        haAssicurazione = random.nextBoolean(); // Genera casualmente true o false per l'assicurazione

    }

    public void eseguiTestPatente() {
        System.out.println("Età: " + eta);
        System.out.println("Ha la patente: " + haPatente);
        System.out.println("Ha l'assicurazione: " + haAssicurazione);

        if (haPatente) {
            if (haAssicurazione) {
                System.out.println("Puoi guidare il tuo veicolo, hai la patente e l'assicurazione.");
            } else {
                System.out.println("Puoi guidare auto di altri, hai la patente anche se non hai l'assicurazione.");
            }
        } else {
            System.out.println("Non puoi guidare un'auto.");
        }
    }

    public static void main(String[] args) {
        for (int i = 1; i <= 5; i++) {
            System.out.println("Persona " + i + ":");
            Persona persona = new Persona();
            persona.eseguiTestPatente();
            System.out.println();
        }
    }
```

Il codice scelto è la classe Persona all'interno del package controlloPatente, vengono generate 5 persone con 3 valori randomici:

- Età: int che va da 18 a 79;
- haPatente: boolean;
- haAssicurazione: boolean;

in seguito, vengono stampate a schermo le informazioni sulle persone generate e con una serie di if si verifica se possono o meno guidare, se la persona ha la patente ma non l'assicurazione può guidare l'auto di altre persone, se ha la patente e l'assicurazione può guidare la sua auto (e quella di altri) ma se non ha la patente non può guidare, il risultato viene stampato a schermo con dei messaggi dedicati per ogni combinazione.

Il numero minimo di test individuato per ricoprire la code coverage al 100% è 5, i test sono riportati di seguito:

```java
@Test
public void testPersona_Si_Patente_No_Assicurazione() {
    Persona persona = new PersonaSoloConPatente();
    persona.eseguiTestPatente();

    String expectedOutput = "Puoi guidare auto di altri, hai la patente anche se non hai l'assicurazione.";
    assertTrue(outContent.toString().contains(expectedOutput));

}

@Test
public void testPersona_No_Patente_Si_Assicurazione() {
    Persona persona = new PersonaSoloConAssicurazione();
    persona.eseguiTestPatente();

    String expectedOutput = "Non puoi guidare un'auto.";
    assertTrue(outContent.toString().contains(expectedOutput));

}

@Test
public void testPersona_No_Patente_No_Assicurazione() {
    Persona persona = new PersonaSenzaPatenteEAssicurazione();
    persona.eseguiTestPatente();

    String expectedOutput = "Non puoi guidare un'auto.";
    assertTrue(outContent.toString().contains(expectedOutput));

}

@Test
public void testPersona_Si_Patente_Si_Aassicurazione() {
    Persona persona = new PersonaConPatenteEAassicurazione();
    persona.eseguiTestPatente();

    String expectedOutput = "Puoi guidare il tuo veicolo, hai la patente e l'assicurazione.";
    assertTrue(outContent.toString().contains(expectedOutput));
}
@Test
public void testOutputdelMain() {
    Persona.main(new String[0]); // Chiamo il main


    String actualOutput = outContent.toString();
    assertFalse(actualOutput.isEmpty()); // Verifico che l'output non sia vuoto
}
```

I 5 test coprono diversi casi:

- Il primo test verifica il caso in cui la persona ha la patente ma non l'assicurazione;
- Il secondo test verifica il caso in cui la persona non ha la patente ma ha l'assicurazione;
- Il terzo test verifica il caso in cui la persona non ha né la patente né l'assicurazione;
- Il quarto test verifica il caso in cui la persona ha sia la patente che l'assicurazione:
- Il quinto test verifica che l'output atteso non risulti vuoto.

1) Understanding the requirements

Obiettivo della classe

La classe restituisce un output differente in base al confronto dei valori booleani presi in input.

Input

Età: da 18 a 79.

haPatente: boolean true/false.

haAssicurazione: boolean true/false.

Output

Int: Età;

Boolean: haPatente,

Boolean: haAssicurazione;

String: “Puoi guidare il tuo veicolo, hai la patente e l'assicurazione.”;

String: "Puoi guidare auto di altri, hai la patente anche se non hai l'assicurazione.";

String: "Non puoi guidare un'auto.".

2) Explore what the program does for various inputs

In questa fase sono stati ripresi i primi 4 test eseguiti prima che esaminano l’output in base a diverse combinazioni di input.

```
@Test
public void testPersona_Si_Patente_No_Assicurazione() {
    Persona persona = new PersonaSoloConPatente();
    persona.eseguiTestPatente();

    String expectedOutput = "Puoi guidare auto di altri, hai la patente anche se non hai l'assicurazione.";
    assertTrue(outContent.toString().contains(expectedOutput));

}

@Test
public void testPersona_No_Patente_Si_Assicurazione() {
    Persona persona = new PersonaSoloConAssicurazione();
    persona.eseguiTestPatente();

    String expectedOutput = "Non puoi guidare un'auto.";
    assertTrue(outContent.toString().contains(expectedOutput));

}

@Test
public void testPersona_No_Patente_No_Assicurazione() {
    Persona persona = new PersonaSenzaPatenteEAssicurazione();
    persona.eseguiTestPatente();

    String expectedOutput = "Non puoi guidare un'auto.";
    assertTrue(outContent.toString().contains(expectedOutput));

}

@Test
public void testPersona_Si_Patente_Si_Aassicurazione() {
    Persona persona = new PersonaConPatenteEAassicurazione();
    persona.eseguiTestPatente();

    String expectedOutput = "Puoi guidare il tuo veicolo, hai la patente e l'assicurazione.";
    assertTrue(outContent.toString().contains(expectedOutput));
}
```

3) Explore inputs, outputs and identify partitions

Input individuati

Number	Boolean	Boolean
Int età	Boolean haPatente	Boolean haAssicurazione
Valore compreso tra 18-79	True	True
NaN	False	False

Combinazioni possibili

#	Età	haPatente	haAssicurazione
1	NaN	True	True
2	NaN	True	False

3	NaN	False	True
4	NaN	False	False
5	18-79	True	True
6	18-79	True	False
7	18-79	False	True
8	18-79	False	False

Queste sono le combinazioni possibili, anche qui non sono state testate tutte le combinazioni ma solo quelle con un input numerico valido.

Output attesi

#	haPatente	haAssicurazione	Output
1	True	True	"Puoi guidare il tuo veicolo, hai la patente e l'assicurazione."
2	True	False	"Puoi guidare auto di altri, hai la patente anche se non hai l'assicurazione."
3	False	True	"Non puoi guidare un'auto."
4	False	False	"Non puoi guidare un'auto."

4) Boundary Case

Per quanto riguarda i boundary case abbiamo individuato i casi in cui l'età è NaN, le precedenti combinazioni riscontrate e i casi in cui i boolean sono null.

5) Device test cases

Abbiamo aggiunto ai precedenti 5 test, due ulteriori casi di test con lo scopo di verificare che l'età non sia NaN e che i booleani non siano null. I precedenti casi di test andranno dal T1 al T5 con T6-T7 per i due nuovi test aggiunti.

- T1 haPatente: True | haAssicurazione: False
- T2 haPatente: False |haAssicurazione: True
- T3 haPatente: False | haAssicurazione: False
- T4 haPatente: True | haAssicurazione: True
- T5 Output non vuoto
- T6 Booleani non vuoti
- T7 Età non NaN

```
@Test
public void testBooleaniNonVuoti() {
    Persona persona = new Persona();

    assertNotNull(persona.getHaPatente());
    assertNotNull(persona.getHaAssicurazione());
}

@Test
public void testEtàNonNaN() {
    Persona persona = new Persona();

    double eta = persona.getEta();
    assertFalse(Double.isNaN(eta));
}
```

Abbiamo in seguito effettuato nuovamente la code coverage riscontrando una copertura del 100% del codice anche a seguito dell'aggiunta di questi due test.