Documentazione di Progetto

Dipartimento di Informatica

Corso di laurea "Informatica e Tecnologie per la Produzione del software"

Integrazione e Test di Sistemi Software

A.A. 2022/2023

Prof.ssa **Azzurra Ragone**

Nome gruppo: **GB_Team**

Membri		Contatti
Benedetto Bellarte	Mat: **744402**	b.bellarte1@studenti.uniba.it
Giovanni Cosi	Mat: **735629**	g.cosi8@studenti.uniba.it

Sommario

Homework 1 4
Introduzione 4
Descrizione 4
Codice 4
1 Comprensione dei requisiti, degli input e degli output 5
2 Analisi dell'output del programma al variare dell'input 6
3 Possibili input e output e partizioni 7
4 Analisi dei boundaries 8
5 Progettazione dei casi di test 9
6 Automazione dei casi di test 11
7 Casi di test aggiuntivi 13
Homework 2 14
Task 1 14
Introduzione 14
Risultati analisi JaCoCo 14
Criterio di code coverage 14
Homework 2 18
Task 2 18
Introduzione 18
Descrizione 18
Codice 18
Specification-Based Test 19
1 Comprensione dei requisiti, degli input e degli output 19
2 Analisi dell'output del programma al variare dell'input 20
3 Possibili input e output e partizioni 20
4 Analisi dei boundaries 22
5 Progettazione dei casi di test 22
6 Automazione dei casi di test 23
7 Casi di test aggiuntivi 23
Structural Test 24
Introduzione 24
Creazione dei casi di test 24
Condizione if (1) 24
Condizione if (2) 25
Condizione if (3) 26

Condizione if (4) 27
Condizione if-else (1) 27
Condizione if-else (2) 28
Condizione if-else (3) 31
Risultati 32
Homework 3 33
Introduzione 33
Descrizione 33
Codice 33
Implementazione Property-Based Test 34
Property-Based Test di setPrezzoNetto() 34
Property-Based Test di getIva() 34
Report Statistiche Property-Based Test 36
invalidSetPrezzoNettoTest() 36
passSetPrezzoNettoTest() 37
nullGetIvaTest() 37
notNullGetIvaTest() 38

Homework 1

Progettare e implementare una suite di test black-box (specification-based testing)

Introduzione

Descrizione

La funzione `isMaggiorennePhrase()` è un metodo implementato con Java 19. Questo metodo prende in input quattro parametri e restituisce una stringa: nome, cognome, eta e dataNascita. Presi questi quattro parametri, la funzione elabora il nome e il cognome, capitalizzando le due variabili. Inoltre, afferma se una persona è maggiorenne oppure no, in base all'età e alla data di nascita.

Tuttavia, vi sono una serie di condizioni affinché la funzione restituisca una risposta corretta. Tali condizioni sono:

a. Nome e Cognome devono essere diversi da null e blank;
b. Nome e Cognome devono essere diversi tra loro;
c. La lunghezza del Nome deve essere maggiore o uguale alla lunghezza del Cognome;
d. Età (ovviamente) deve essere maggiore o uguale a 0 e non può essere null;
e. Data di nascita non può essere null, né maggiore della data odierna e deve essere coerente all'età inserita (ossia il periodo di tempo in anni tra la data odierna e la data di nascita deve corrispondere all'età inserita).

Codice

```java
public static String isMaggiorennePhrase(String nome, String cognome, int eta, Date dataNascita) {

    // Verifica che nome e/o cognome non siano null
    if (nome == null || cognome == null)
        return "Nome o cognome null";

    // Verifica che nome e/o cognome contengano solo lettere
    if (nome.matches(".*\\d.*") || cognome.matches(".*\\d.*"))
        return "Nome o cognome non valido";

    // Verifica che nome e/o cognome non sia una stringa vuota
    if (nome.isBlank() || cognome.isBlank())
        return "Nome e/o cognome vuoto";

    // Verifica che nome e cognome siano diversi
    if (nome.equals(cognome))
        return "Nome e cognome sono uguali";

    // Nome deve essere maggiore uguale rispetto al cognome e non può essere uguale
    if (nome.length() < cognome.length())
        return "Nome minore del cognome";

    // Verifica che l'età non sia minore di zero
    if (eta < 0)
        return "Età minore di zero";

    // Verifica che dataNascita non sia null
    if (dataNascita == null)
        return "dataNascita null";

    // Verifica che la data di nascita non sia maggiore della data odierna
    if(dataNascita.after(Date.valueOf(LocalDate.now())))
        return "Data nascita maggiore della data odierna";

    // Verifica che l'età data in input coincida con la data di nascita data in input
    int diffYears = Period.between(dataNascita.toLocalDate(), LocalDate.now()).getYears();
    if (diffYears != eta)
        return "Età non coincide con la data di nascita " + diffYears;
```

```java
    // Capitalizzazione del nome
    nome = nome.toLowerCase();
    char[] stringArray = nome.toCharArray();
    boolean spazi = true;
    for(int i = 0; i < stringArray.length; i++)
        if(Character.isLetter(stringArray[i])) {
            if(spazi) {
                stringArray[i] = Character.toUpperCase(stringArray[i]);
                spazi = false;
            }
        }
        else
            spazi = true;
    nome = String.valueOf(stringArray);

    // Capitalizzazione del cognome
    cognome = cognome.toLowerCase();
    stringArray = cognome.toCharArray();
    spazi = true;
    for (int i = 0; i < stringArray.length; i++)
        if(Character.isLetter(stringArray[i])) {
            if(spazi) {
                stringArray[i] = Character.toUpperCase(stringArray[i]);
                spazi = false;
            }
        }
        else
            spazi = true;
    cognome = String.valueOf(stringArray);

    String isMaggiorenneString = null;
    if (eta >= 18) // Controllo dell'età
        isMaggiorenneString = "è maggiorenne";
    else
        isMaggiorenneString = "non è maggiorenne";

    // Verifica se la data di nascita inserita sia il giorno del compleanno confrontata con la data odierna
    if (dataNascita.toLocalDate().getDayOfMonth() == LocalDate.now().getDayOfMonth() &&
            dataNascita.toLocalDate().getMonthValue() == LocalDate.now().getMonthValue())
        return "Buon compleanno " + nome + " " + cognome + "!";

    return nome + " " + cognome + " " + isMaggiorenneString + "!";
}
```

1 Comprensione dei requisiti, degli input e degli output

```
public String isMaggiorennePhrase()
```

Obbiettivo:

Funzione che prende in input il nome, il cognome, l'età, la data di nascita di una persona e restituisce una frase che indica se questa è maggiorenne o meno oppure se è il suo compleanno. Inoltre, se un qualche dato di input è errato viene restituita una stringa che fornisce una motivazione dell'errore.

Input:

- nome: Stringa che rappresenta il nome della persona
- cognome: Stringa che rappresenta il cognome della persona
- eta: Intero che rappresenta l'età della persona
- dataNascita: Oggetto di tipo Java.sql.Date che rappresenta la data di nascita della persona

Return: String

- "<Nome> <Cognome> è maggiorenne!" → Se la persona è maggiorenne;
- "<Nome> <Cognome> non è maggiorenne!" → Se la persona non è maggiorenne;

- “Buon compleanno <Nome> <Cognome>!” → Se il giorno e il mese di nascita coincidono con la data odierna;
- “Nome e/o cognome null” → Se non sono stati passati in input alla funzione il nome e/o il cognome;
- “Nome e/o cognome non valido” → Se si è passati una stringa contenente simboli o numeri in input alla funzione in corrispondenza del nome e/o del cognome;
- “Nome e/o cognome vuoto” → Se si è passati una stringa vuota in input alla funzione in corrispondenza del nome e/o del cognome;
- “Nome e cognome sono uguali” → Se il nome e il cognome coincidono;
- “Nome minore del cognome” → Se la lunghezza del nome e minore della lunghezza del cognome;
- “Età minore di zero” → Se l’età è minore di zero;
- “dataNascita null” → Se non viene passata alcuna data alla funzione in corrispondenza della data di nascita
- “Data nascita maggiore della data odierna” → Se la data di nascita è maggiore della data odierna;
- “Età non coincide con la data di nascita <differenza_anni>” → Se il periodo di tempo in anni tra la data odierna e la data di nascita deve corrispondere all’età inserita.

2 Analisi dell’output del programma al variare dell’input

Partiamo prima con il capire cosa fa il metodo, leggendo le specifiche e la documentazione associata alla funzione.

Il metodo `isMaggiorennePhrase()` permette di verificare se un utente sia maggiorenne oppure no. Inoltre, i dati che vengono passati alla funzione devono rispettare determinate condizioni:

1. nome e cognome diversi da null o blank;
2. nome e cognome diversi tra loro;
3. la lunghezza del nome deve essere maggiore o oguale a quella del cognome;
4. età non può essere un numero negativo;
5. la data di nascita deve essere coerente con l'eta che viene data in input;
6. la data di nascita deve essere minore o uguale della data odierna.

La funzione infine effettua la capitalizzazione sulle stringhe di nome e cognome.

Dopodiché abbiamo scritto/pensato a 7 simple case test per comprendere, in maniera più approfondita, cosa facesse la funzione.

```java
@Test
public void simpleCaseTest() {
    assertEquals(
            isMaggiorennePhrase("Mario", "Rossi", 21, Date.valueOf(LocalDate.parse("2001-07-26"))),
            "Mario Rossi è maggiorenne!");
    assertEquals(
            isMaggiorennePhrase("Mario", "Rossi", 5, Date.valueOf(LocalDate.parse("2018-04-06"))),
            "Mario Rossi non è maggiorenne!");
    assertEquals(
            isMaggiorennePhrase("Mario", "Rossi", 11, Date.valueOf(LocalDate.parse("2012-05-05"))),
            "Buon compleanno Mario Rossi!");
    assertEquals(
            isMaggiorennePhrase("Mario", "Rossi", -11, Date.valueOf(LocalDate.parse("2012-05-05"))),
            "Età minore di zero");
    assertEquals(
            isMaggiorennePhrase(null, "Rossi", 11, Date.valueOf(LocalDate.parse("2012-05-05"))),
            "Nome e/o cognome null");
    assertEquals(
            isMaggiorennePhrase("Mario", "Marroni", 11, Date.valueOf(LocalDate.parse("2012-05-05"
))),         "Nome minore del cognome");
    assertEquals(
            isMaggiorennePhrase("Mario", "Marroni", 0, Date.valueOf(LocalDate.parse("2020-07-07"))),
            "Età non coincide con la data di nascita 2");
}
```

3 Possibili input e output e partizioni

Possibili input:

Parametro nome – Questo parametro può essere:

a. Null
b. Empty
c. Stringa con capolettera minuscola
d. Stringa con capolettera maiuscola
e. Stringa composta da più parole
f. Stringa con un solo carattere maiuscolo
g. Stringa con un solo carattere minuscolo

Parametro cognome – Questo parametro può essere:

a. Null
b. Empty
c. Stringa con capolettera minuscola
d. Stringa con capolettera maiuscola
e. Stringa composta da più parole
f. Stringa con un solo carattere maiuscolo
g. Stringa con un solo carattere minuscolo

Parametro eta – Questo parametro può essere:

a. Minore di zero
b. Uguale a zero
c. Maggiore di zero

Parametro dataNascita – Questo parametro può essere:

a. Minore della data odierna

b. Uguale alla data odierna
c. Maggiore della data odierna

(nome, cognome) parameters – La lunghezza di nome deve essere maggiore o uguale della lunghezza del cognome, inoltre, nome e cognome devono essere due stringhe diverse.

a. Nome ha lunghezza maggiore rispetto al cognome
b. Nome ha la medesima lunghezza di cognome
c. Nome ha lunghezza minore di cognome
d. Nome è uguale al cognome
e. Nome è diverso dal cognome

(eta, dataNascita) parameters – eta deve essere uguale alla differenza tra la data odierna e la data di nascita.

a. La differenza in anni tra la data odierna e la data di nascita è minore rispetto all'età
b. La differenza in anni tra la data odierna e la data di nascita è uguale rispetto all'età
c. La differenza in anni tra la data odierna e la data di nascita è maggiore rispetto all'età

Possibili output:

L'output è un dato di tipo String che può essere:

a. "<Nome> <Cognome> è maggiorenne!"
b. "<Nome> <Cognome> non è maggiorenne!"
c. "Buon compleanno <Nome> <Cognome>!"
d. "Nome e/o cognome null"
e. "Nome e/o cognome non valido"
f. "Nome e/o cognome vuoto"
g. "Nome e cognome sono uguali"
h. "Nome minore del cognome"
i. "Età minore di zero"
j. "dataNascita null"
k. "Data nascita maggiore della data odierna"
l. "Età non coincide con la data di nascita <differenza_anni>"

4 Analisi dei boundaries

1	Età	a. Età minore di zero – *In/On Point* b. Età uguale a zero – *Off Point* c. Età maggiore di zero – *Out/Off Point*
2	Età maggiorenne	a. Età maggiore di 18 – *In Point* b. Età uguale a 18 – *On/In Point* c. Età uguale a 17 – *Off/Out Point* d. Età minore di 18 – *Off Point*
3	Nome – cognome	a. Lunghezza del nome minore di cognome – *In/On Point* b. Lunghezza del nome uguale di cognome – *Off Point* c. Lunghezza del nome maggiore di cognome – *Out/Off Point*

4	Data di nascita	a. Data di nascita minore della data odierna – *Out/Off Point* b. Data di nascita uguale della data odierna – *Off Point* c. Data di nascita maggiore della data odierna – *In/On Point*
5	Data di nascita – età	a. La differenza in anni tra la data odierna e la data di nascita è diversa dall'età inserita – *Off Point* b. La differenza in anni tra la data odierna e la data di nascita è uguale all'età inserita – *On Point*

5 Progettazione dei casi di test

Per capire quanti casi di test dobbiamo progettare, effettuiamo un calcolo. Dal calcolo risultano che bisogna effettuare 7 * 7 * 3 * 3 * 5 * 3 = 6615 test. Essendo troppi test, analizziamo i casi che possono essere testati una sola volta.

Dopo aver trovato i vari input, output e boundaries, passiamo ora a concretizzare i vari casi di test seguendo la seguente strategia:

a. Casi eccezionali di nome e cognome:
- **T1**: Caso in cui nome e cognome sono null vengono testati una sola volta
- **T2**: Caso in cui nome e cognome sono empty vengono testati una sola volta

b. Casi boundary età:
- **T3**: Caso in cui l'età è minore di zero viene testato una sola volta
- **T4**: Caso in cui l'età è maggiore di 18 viene testato una sola volta
- **T5**: Caso in cui l'età è minore di 18 viene testato una sola volta
- **T6**: Caso in cui l'età è uguale a 17 viene testato una sola volta
- **T7**: Caso in cui l'età è uguale a 18 viene testato una sola volta

c. Casi capolettera nome e cognome:
- **T8**: Caso in cui nome e cognome hanno un capolettera minuscola viene testato una volta
- **T9**: Caso in cui nome e cognome hanno un capolettera maiuscola viene testato una volta

d. Casi con più parole nome e cognome:
- **T10**: Caso in cui il nome e cognome è composto da due o più parole viene testato una sola volta

e. Casi con lettere maiuscole e minuscole di nome e cognome:
- **T11**: Caso in cui il nome e cognome ha un solo carattere maiuscolo (tranne il capolettera) viene testato una sola volta
- **T12**: Caso in cui il nome e cognome ha un solo carattere minuscolo (tranne il capolettera) viene testato una sola volta

f. Casi con differenti lunghezze di nome e cognome:
- **T13**: Caso in cui la lunghezza del nome è minore della lunghezza del cognome viene testato una sola volta
- **T14**: Caso in cui la lunghezza del nome è uguale della lunghezza del cognome viene testato una sola volta

- **T15**: Caso in cui la lunghezza del nome è maggiore della lunghezza del cognome viene testato una sola volta

g. Casi di confronto tra nome e cognome:

- **T16**: Caso in cui nome e cognome sono la medesima stringa viene testato una sola volta
- **T17**: Caso in cui nome e cognome sono due stringhe diverse viene testato una sola volta

h. Casi di confronto di lunghezza tra nome e cognome con più stringhe:

- **T18**: Caso in cui nome e cognome sono composti da più parole con la lunghezza del nome minore della lunghezza del cognome viene testato una sola volta
- **T19**: Caso in cui nome e cognome sono composti da più parole con la lunghezza del nome uguale della lunghezza del cognome viene testato una sola volta
- **T20**: Caso in cui nome e cognome sono composti da più parole con la lunghezza del nome maggiore della lunghezza del cognome viene testato una sola volta

i. Casi di conforto tra data di nascita ed età:

- **T21**: Caso in cui la differenza in anni tra la data odierna e la data di nascita è diversa dall'età inserita viene testato una sola volta
- **T22**: Caso in cui la differenza in anni tra la data odierna e la data di nascita è uguale all'età inserita viene testato una sola volta

j. Casi tra data di nascita e età è uguale a zero:

- **T23**: Caso in cui età uguale a zero e la differenza in anni tra la data odierna e la data di nascita è minore rispetto all'età viene testato una sola volta
- **T24**: Caso in cui età uguale a zero e la differenza in anni tra la data odierna e la data di nascita è uguale rispetto all'età viene testato una sola volta
- **T.25**: Caso in cui età uguale a zero e la differenza in anni tra la data odierna e la data di nascita è maggiore rispetto all'età viene testato una sola volta

k. Casi tra data di nascita e età è maggiore di zero:

- **T26**: Caso in cui età maggiore a zero e la differenza in anni tra la data odierna e la data di nascita è minore rispetto all'età viene testato una sola volta
- **T27**: Caso in cui età maggiore a zero e la differenza in anni tra la data odierna e la data di nascita è uguale rispetto all'età viene testato una sola volta
- **T28**: Caso in cui età maggiore a zero e la differenza in anni tra la data odierna e la data di nascita è maggiore rispetto all'età viene testato una sola volta

6 Automazione dei casi di test

a. Casi eccezionali di nome e cognome:

```
class MainClassH1Test {

    // SPECIFICATION-BASED TEST
    @Test
    public void nomeCognomeNullEmpty() { // a. Casi eccezionali di nome e cognome
        assertEquals(
                isMaggiorennePhrase(null, null, 11, Date.valueOf(LocalDate.parse("2012-05-05"))),
                "Nome o cognome null"); //T1
        assertEquals(
                isMaggiorennePhrase("", "", 11, Date.valueOf(LocalDate.parse("2012-05-05"))),
                "Nome e/o cognome vuoto"); //T2
    }
```

b. Casi boundary età:

```
    @Test
    public void boundaryCasesEta() { // b. Casi boundary età
        assertEquals(
                isMaggiorennePhrase("Mario", "Rossi", -11, Date.valueOf(LocalDate.parse("2012-05-05"))),
                "Età minore di zero"); //T3
        assertEquals(
                isMaggiorennePhrase("Mario", "Rossi", 21, Date.valueOf(LocalDate.parse("2001-11-14"))),
                "Mario Rossi è maggiorenne!"); //T4
        assertEquals(
                isMaggiorennePhrase("Mario", "Rossi", 15, Date.valueOf(LocalDate.parse("2007-11-14"))),
                "Mario Rossi non è maggiorenne!"); //T5
        assertEquals(
                isMaggiorennePhrase("Mario", "Rossi", 17, Date.valueOf(LocalDate.parse("2005-11-14"))),
                "Mario Rossi non è maggiorenne!"); //T6
        assertEquals(
                isMaggiorennePhrase("Mario", "Rossi", 18, Date.valueOf(LocalDate.parse("2004-11-14"))),
                "Mario Rossi è maggiorenne!"); //T7
    }
```

c. Casi capolettera nome e cognome:

```
    @Test
    public void capoletteraNomeCognomeMinuscolaMaiuscola() { // c. Casi capolettera nome e cognome
        assertEquals(
                isMaggiorennePhrase("mario", "rossi", 21, Date.valueOf(LocalDate.parse("2001-07-26"))),
                "Mario Rossi è maggiorenne!"); //T8
        assertEquals(
                isMaggiorennePhrase("Mario", "Rossi", 21, Date.valueOf(LocalDate.parse("2001-07-26"))),
                "Mario Rossi è maggiorenne!"); //T9
    }
```

d. Casi con più parole nome e cognome:

```
    @Test
    public void nomeCognomeManyWords() { // d. Casi con più parole nome e cognome
        assertEquals(
                isMaggiorennePhrase("Mario Stefano Violi", "Rossi Verdi", 21, Date.valueOf(LocalDate.parse("2001-07-26"))),
                "Mario Stefano Violi Rossi Verdi è maggiorenne!"); //T10
    }
```

e. Casi con lettere maiuscole e minuscole di nome e cognome:

```
    @Test
    public void nomeCognomeOnlyOneLetterMinuscolaMaiuscola() { // e. Casi con lettere maiuscole e minuscole di nome e cognome
        assertEquals(
                isMaggiorennePhrase("maRio", "roSsi", 21, Date.valueOf(LocalDate.parse("2001-07-26"))),
                "Mario Rossi è maggiorenne!"); //T11
        assertEquals(
                isMaggiorennePhrase("MArIO", "ROsSI", 21, Date.valueOf(LocalDate.parse("2001-07-26"))),
                "Mario Rossi è maggiorenne!"); //T12
    }
```

f. Casi con differenti lunghezze di nome e cognome:

```
    @Test
    public void lenNomeCognome() { // f. Casi con differenti lunghezze di nome e cognome
        assertEquals(
                isMaggiorennePhrase("Mario", "Marroni", 21, Date.valueOf(LocalDate.parse("2001-07-26"))),
                "Nome minore del cognome"); //T13
        assertEquals(
                isMaggiorennePhrase("Mario", "Rossi", 21, Date.valueOf(LocalDate.parse("2001-07-26"))),
                "Mario Rossi è maggiorenne!"); //T14
        assertEquals(
                isMaggiorennePhrase("Mario", "Cosi", 21, Date.valueOf(LocalDate.parse("2001-07-26"))),
                "Mario Cosi è maggiorenne!"); //T15
    }
```

g. Casi di confronto tra nome e cognome:

```
    @Test
    public void nomeEqualOrNotEqualCognome() { // g. Casi di confronto tra nome e cognome
        assertEquals(
                isMaggiorennePhrase("Mario", "Mario", 21, Date.valueOf(LocalDate.parse("2001-07-26"))),
                "Nome e cognome sono uguali"); //T16
        assertEquals(
                isMaggiorennePhrase("Vittorio", "Rossi", 21, Date.valueOf(LocalDate.parse("2001-07-26"))),
                "Vittorio Rossi è maggiorenne!"); //T17
    }
```

h. Casi di confronto di lunghezza tra nome e cognome con più stringhe:

```
    @Test
    public void nomeCognomeLenComparisonWithManyWords() {
// h. Casi di confronto di lunghezza tra nome e cognome con più stringhe
        assertEquals(
                isMaggiorennePhrase("Mario Violi", "Rossi Marroni", 21, Date.valueOf(LocalDate.parse("2001-07-26"))),
                "Nome minore del cognome"); //T18
        assertEquals(
                isMaggiorennePhrase("Mario Violi", "Rossi Verdi", 21, Date.valueOf(LocalDate.parse("2001-07-26"))),
                "Mario Violi Rossi Verdi è maggiorenne!"); //T19
        assertEquals(
                isMaggiorennePhrase("Mario Stefano", "Rossi Verdi", 21, Date.valueOf(LocalDate.parse("2001-07-26"))),
                "Mario Stefano Rossi Verdi è maggiorenne!"); //T20
    }
```

i. Casi di conforto tra data di nascita ed età:

```
    @Test
    public void comparisonDataNascitaAndEta() { // i. Casi di conforto tra data di nascita ed età
        assertEquals(
                isMaggiorennePhrase("Mario", "Rossi", 15, Date.valueOf(LocalDate.parse("1975-02-07"))),
                "Età non coincide con la data di nascita 48"); //T21
        assertEquals(
                isMaggiorennePhrase("Mario", "Rossi", 48, Date.valueOf(LocalDate.parse("1975-02-07"))),
                "Mario Rossi è maggiorenne!"); //T22
    }
```

j. Casi tra data di nascita e età è uguale a zero:

```
    @Test
    public void etaEqualToZeroAndPeriod() { // j. Casi tra data di nascita e età è uguale a zero
        assertEquals(
                isMaggiorennePhrase("Mario", "Rossi", 0, Date.valueOf(LocalDate.parse("2030-07-07"))),
                "Data nascita maggiore della data odierna"); //T23
        assertEquals(
                isMaggiorennePhrase("Mario", "Rossi", 0, Date.valueOf(LocalDate.now())),
                "Buon compleanno Mario Rossi!"); //T24
        assertEquals(
                isMaggiorennePhrase("Mario", "Rossi", 0, Date.valueOf(LocalDate.parse("2020-07-07"))),
                "Età non coincide con la data di nascita 2"); //T25
    }
```

k. Casi tra data di nascita e età è maggiore di zero:

```
    @Test
    public void etaGreaterThenZeroAndPeriod() { // k. Casi tra data di nascita e età è maggiore di zero
        assertEquals(
                isMaggiorennePhrase("Mario", "Rossi", 10, Date.valueOf(LocalDate.parse("2018-02-07"))),
                "Età non coincide con la data di nascita 5"); //T26
        assertEquals(
                isMaggiorennePhrase("Mario", "Rossi", 10, Date.valueOf(LocalDate.parse("2013-02-07"))),
                "Mario Rossi non è maggiorenne!"); //T27
        assertEquals(
                isMaggiorennePhrase("Mario", "Rossi", 10, Date.valueOf(LocalDate.parse("2010-02-07"))),
                "Età non coincide con la data di nascita 13"); //T28
    }

}
```

7 Casi di test aggiuntivi

Dopo aver automatizzato i casi test trovati dall'analisi degli input, output e boundaries, abbiamo pesato di introdurre altri due test, che nell'analisi precedenti non erano stati affrontati.

- **P.1**: `nomeCognomeWithNumber()` → testa il caso in cui nel nome e nel cognome ci siano dei numeri
- **P.2**: `nomeCognomeWithSymbols()` → testa il caso in cui nel nome e nel cognome ci siano dei simboli

```
@Test
public void nomeCognomeWithNumber() {
    assertEquals(
            isMaggiorennePhrase("Mar10", "Ross1", 11, Date.valueOf(LocalDate.parse("2012-05-05"))),
            "Nome o cognome non valido");
}

@Test
public void nomeCognomeWithSymbols() {
    assertEquals(
            isMaggiorennePhrase("M@rio", "R0$$i", 11, Date.valueOf(LocalDate.parse("2012-05-05"))),
            "Nome o cognome non valido");
}
```

Homework 2

Task 1

Completare la suite di test black-box (specification-based testing) analizzata per l'homework 1 con gli structural testing (white-box) e con l'analisi di copertura del codice (code coverage).

Introduzione

Dopo aver sviluppato la suite di test black-box (assieme ai test aggiuntivi), abbiamo analizzato la code coverage della funzione `isMaggiorennePhrase()` (metodo analizzato nell'homework 1). Da questa analisi abbiamo ottenuto che le seguenti parti di codice non parzialmente testate o non testate affatto:

```
32          // Verifica che nome e/o cognome non siano null
33          if (nome == null || cognome == null)
34              return "Nome o cognome null";
35
```

```
39
40          // Verifica che nome e/o cognome non sia una stringa vuota
41          if (nome.isBlank() || cognome.isBlank())
42              return "Nome e/o cognome vuoto";
43
```

```
56          // Verifica che dataNascita non sia null
57          if (dataNascita == null)
58              return "dataNascita null";
59
```

Inoltre, i risultati di coverage (mediante l'utilizzo di JaCoCo) sono i seguenti:

- Class % = 100%;
- Method % = 100%;
- Line % = 97% (46/47);
- Branch % = 92% (39/42).

Risultati analisi JaCoCo

Visualizzando il codice, sono presenti sia righe di codice parzialmente testate negli specification-based test e una riga di codice che non è stata trattata negli specification-based test:
La riga di codice che risulta non testata nasce dal fatto che questa non era segnalata all'interno della JavaDoc, quindi eseguendo i test black-box era difficile rilevarla.

Criterio di code coverage

Partendo ora da tutte le righe parzialmente o non testate, studiamo ora un corretto criterio per raggiungere il 100% di code coverage.

In generale scegliere un criterio di coverage è particolarmente importante al fine di testare in modo efficiente il codice.

```
if (nome == null || cognome == null)
```

In questa condizione abbiamo deciso di adoperare il criterio di **MC/DC**. Pertanto, questa istruzione può essere testata con 3 test. I test sono:

Test Case	`nome == null`	`cognome == null`	Decisione
T29	True	True	True
T30	True	False	True
T31	False	True	True
T32	False	False	False

nome == null → **(T30, T32)**
cognome == null → **(T31, T32)**
Pertanto, i test necessari per raggiungere il 100% di coverage per questa istruzione sono:

l. **Casi in cui nome o cognome sono uguali a null:**
   - **T30:** Caso in cui nome è uguale a null mentre cognome no
   - **T31**: Caso in cui cognome è uguale a null mentre nome no
   - **T32**: Caso in cui sia nome che cognome sono diversi da null

```java
@Test
public void nomeCognomeNull() { // l. Casi in cui nome o cognome sono uguali a null:
    assertEquals(
            isMaggiorennePhrase(null, "Rossi", 21, Date.valueOf(LocalDate.parse("2001-11-14"))),
            "Nome o cognome null"); // T30
    assertEquals(
            isMaggiorennePhrase("Mario", null, 21, Date.valueOf(LocalDate.parse("2001-11-14"))),
            "Nome o cognome null"); // T31
    assertEquals(
            isMaggiorennePhrase(null, null, 21, Date.valueOf(LocalDate.parse("2001-11-14"))),
            "Nome o cognome null"); // T32
}
```

```
if (nome.isBlank() || cognome.isBlank())
```

In questa condizione abbiamo deciso di adoperare il criterio di **MC/DC**. Pertanto, questa istruzione può essere testata con 3 test. I test sono:

Test Case	`nome.isBlank()`	`cognome.isBlank()`	Decisione
T33	True	True	True
T34	True	False	True
T35	False	True	True
T36	False	False	False

nome.isBlank() → **(T34, T36)**
cognome.isBlank() → **(T35, T36)**
Pertanto, i test necessari per raggiungere il 100% di coverage per questa istruzione sono:

m. **Casi in cui nome o cognome sono vuoti:**

- **T34:** Caso in cui nome.isBlank() è verificata mentre cognome.isBlank() non è verificata
- **T35**: Caso in cui cognome.isBlank() è verificata mentre nome.isBlank() non è verificata
- **T36**: Caso in cui sia nome.isBlank() che cognome.isBlank() non sono verificati

```java
@Test
public void nomeCognomeIsBlank() { // m. Casi in cui nome o cognome sono vuoti:
    assertEquals(
            isMaggiorennePhrase("", "Rossi", 21, Date.valueOf(LocalDate.parse("2001-11-14"))),
            "Nome e/o cognome vuoto"); // T34
    assertEquals(
            isMaggiorennePhrase("Mario", "", 21, Date.valueOf(LocalDate.parse("2001-11-14"))),
            "Nome e/o cognome vuoto"); // T35
    assertEquals(
            isMaggiorennePhrase("", "", 21, Date.valueOf(LocalDate.parse("2001-11-14"))),
            "Nome e/o cognome vuoto"); // T36
}
```

```
if (dataNascita == null)
```

In questa condizione abbiamo deciso di adoperare il criterio di **Branch Coverage**. Pertanto, questa istruzione può essere testata con 2 test. I test sono

n. **Casi in cui data di nascita è nulle oppure diversa da null:**
   - **T37:** Caso in cui data di nascita è uguale a null
   - **T38**: Caso in cui data di nascita è diversa da null

```java
@Test
public void dataNascitaNull() { // l. Casi in cui data di nascita è nulle oppure diversa da null
    assertEquals(
            isMaggiorennePhrase("Mario", "Rossi", 11, null),
            "dataNascita null"); //T29
    assertEquals(
            isMaggiorennePhrase("Mario", "Rossi", 11, Date.valueOf(LocalDate.parse("2012-05-05"))),
            "Mario Rossi non è maggiorenne!"); //T30
}
```

```
if(dataNascita.toLocalDate().getDayOfMonth() == LocalDate.now().
      getDayOfMonth() && dataNascita.toLocalDate().getMonthValue() ==
      LocalDate.now().getMonthValue())
```

In questa condizione abbiamo deciso di adoperare il criterio di **MC/DC**. Pertanto, questa istruzione può essere testata con 3 test.
Assumiamo, per semplicità, i seguenti nomi:

A = `dataNascita.toLocalDate().getDayOfMonth() == LocalDate.now().getDayOfMonth()`
B = `dataNascita.toLocalDate().getMonthValue() == LocalDate.now().getMonthValue()`

Basandoci quindi sulla condizione if(A && B), i test sono:

Test Case	A	B	Decisione
T39	True	True	True
T40	True	False	False
T41	False	True	False
T42	False	False	False

nome.isBlank() → **(T39, T41)**
cognome.isBlank() → **(T39, T40)**
Pertanto, i test necessari per raggiungere il 100% di coverage per questa istruzione sono:

o. **Casi in cui data e mese inseriti sono uguali a quelli odierni:**
   - **T39:** Caso in cui sia il mese che il giorno coincidono con la data odierna
   - **T40**: Caso in cui solo il giorno coincide con la data odierna mentre il mese no
   - **T41**: Caso in cui sia solo il mese coincide con la data odierna mentre il giorno no

```
@Test
    public void meseGiornoEqualActualDate() { // o. Casi in cui data e mese inseriti sono uguali a quelli odierni:
        int giorno = LocalDate.now().getDayOfMonth();
        int mese = LocalDate.now().getMonthValue();
        String giornoStr, meseStr;
        if (giorno < 10)
            giornoStr = "0" + giorno;
        else
            giornoStr = String.valueOf(giorno);
        if (mese < 10)
            meseStr = "0" + mese;
        else
            meseStr = String.valueOf(mese);

        assertEquals(
                isMaggiorennePhrase("Mario", "Rossi", 22, Date.valueOf(LocalDate.parse("2001-"+ meseStr + "-" +
giornoStr))),
                "Buon compleanno Mario Rossi!"); // T39
        assertEquals(
                isMaggiorennePhrase("Mario", "Rossi", 21, Date.valueOf(LocalDate.parse("2001-11-" + giornoStr))),
                "Mario Rossi è maggiorenne!"); // T40
        assertEquals(
                isMaggiorennePhrase("Mario", "Rossi", 21, Date.valueOf(LocalDate.parse("2001-" + meseStr + "-30"))),
                "Mario Rossi è maggiorenne!"); // T41
    }
```

Homework 2

Task 2

Progettare una suite di test black-box e progettare e implementare una suite di casi di test minimi per raggiungere il 100% di coverage

Introduzione

Descrizione

La funzione `squareCubeList()` è un metodo implementato con Java 19. Questo metodo prende in input quattro parametri e restituisce un oggetto di tipo ArrayList: n, m, nomi, numeri. I primi due parametri sono dei valori di tipo int, mentre gli ultimi due sono rispettivamente un array di stringhe e un array di Integer. Presi questi quattro parametri, per la lista dei numeri viene calcolato il quadrato o il cubo del numero a seconda che questo sia rispettivamente pari o dispari. Inoltre, si verifica che il quadrato non superi 99 e il cubo 299. Dopo aver effettuato i calcoli la funzione restituisce un oggetto di tipo ArrayList contenente i risultati calcolati.

Tuttavia, vi sono una serie di condizioni affinché la funzione restituisca una risposta corretta. Tali condizioni sono:

a. n e m che indicano rispettivamente la lunghezza delle liste dei nomi e dei numeri devono avere lo stesso valore ed essere maggiori di zero;
b. n e m devono avere il medesimo valore rispettivamente di nomi.lenght e numeri.lenght;
c. nella lista nomi non vi devono essere duplicati;
d. nella lista numeri vi devono essere numeri compresi tra 1 e 99;
e. Il quadrato del numero non deve superare 99 e il cubo 299;
f. Nella lista dei nomi non vi devono essere quelli uguali a null, blank o che contengano simboli.

Codice

```java
public static ArrayList<String> squareCubeList (int n, int m, String[] nomi, Integer[] numeri) throws
            CubeSquareLimitsExceededException, NumberEqualZeroException, DifferentLenghtNomiNumeriException,
            LenResultEqualZeroException, NegativeNumberException, NumberOffpointException, NomeNullException,
            NomeContainsNumbersException, DuplicatesNomiException {

        if (n == 0 || m == 0)
            throw new LenResultEqualZeroException();
        if (n != m || n != nomi.length)
            throw new DifferentLenghtNomiNumeriException();

        for (int i = 0; i < nomi.length - 1; i++)
            for (int j = i + 1; j < nomi.length; j++)
                if (nomi[i] == nomi[j] && i != j)
                    throw new DuplicatesNomiException();
```

```java
        for (int i = 0; i < n; i++) {
            if (numeri[i] < 0)
                throw new NegativeNumberException();
            else if (numeri[i] > 100)
                throw new NumberOffpointException();
            else if (numeri[i] == 0)
                throw new NumberEqualZeroException();
            else if (((numeri[i] % 2) == 0 && numeri[i] >= 10) || ((numeri[i] % 2) != 0 && numeri[i] >= 7))
                throw new CubeSquareLimitsExceededException();
            else if (nomi[i] == "" || nomi[i] == null)
                throw new NomeNullException();
            else if (nomi[i].matches(".*\\d.*"))
                throw new NomeContainsNumbersException();
        }

        ArrayList<String> result = new ArrayList<>();
        for (int i = 0; i < n; i++) {
            if (numeri[i] % 2 == 0)
                result.add(nomi[i] + " " + numeri[i] * numeri[i]);
            else
                result.add(nomi[i] + " " + numeri[i] * numeri[i] * numeri[i]);
        }

        return result;
    }
```

Specification-Based Test

1 Comprensione dei requisiti, degli input e degli output

```
public static ArrayList<String> squareCubeList()
```

Obbiettivo:

La funzione squareCubeList() prende in input n. Presi questi quattro parametri, per la lista dei numeri viene calcolato il quadrato o il cubo del numero a seconda che questo sia rispettivamente pari o dispari. Inoltre, si verifica che il quadrato non superi 99 e il cubo 299. La funzione restituisce un oggetto di tipo ArrayList contenente i risultati calcolati. Inoltre, se un qualche dato di input è errato viene restituita una specifica eccezione in base al problema.

Input:

- `n`: Intero che rappresenta la lunghezza la lunghezza che dovrebbe avere la lista dei nomi
- `m`: Intero che rappresenta la lunghezza la lunghezza che dovrebbe avere la lista dei numeri
- `nomi`: Array di stringhe che rappresenta il nome di una persona
- `numeri`: Array di Integer che contiene un numero su cui calcolare il quadrato o il cubo

Return: ArrayList

- `ArrayList`: contenente la lista dei risultati dei quadrati e dei cubi

Throw:

- `CubeSquareLimitsExceededException`: se il risultato supera i limiti predisposti;
- `DifferentLenghtNomiNumeriException`: se la lista dei numeri e quella dei nomi hanno lunghezze differenti;
- `LenResultLessEqualZeroException`: se il valore della lunghezza della lista dei nomi e/o numeri è uguale a zero;
- `NumberOffpointException`: se il numero presente nella lista è maggiore o uguale a 100;

- `NomeNullException`: se il nome null o blank;
- `NomeContainsNumbersException`: se il nome contiene dei caratteri non validi;
- `DuplicatesNomiException`: se nella lista nomi sono presenti due o più nomi uguali.

2 Analisi dell'output del programma al variare dell'input

Partiamo prima con il capire cosa fa il metodo, leggendo le specifiche e la documentazione associata alla funzione.

Il metodo `squareCubeList()` restituisce una lista di quadrati e cubi per ciascun nome inserito. Inoltre, i dati che vengono passati alla funzione devono rispettare determinate condizioni:

1. n e m che indicano rispettivamente la lunghezza delle liste dei nomi (nomi.lenght)e dei numeri (numeri.lenght)devono avere lo stesso valore ed essere diversi da zero
2. Nella lista nomi non vi devono essere duplicati
3. Nella lista numeri vi devono essere numeri compresi tra 1 e 99
4. Il quadrato del numero non deve superare 99 e il cubo 299
5. Nella lista dei nomi non vi devono essere quelli uguali a null, blank o che contengano simboli

Dopodiché abbiamo scritto/pensato a 7 simple case test per comprendere, in maniera più approfondita, cosa facesse la funzione.

3 Possibili input e output e partizioni

Possibili input:

Parametro n – Questo parametro può essere:

a. Minore di zero
b. Uguale a zero
c. Maggiore di zero

Parametro m – Questo parametro può essere:

a. Minore di zero
b. Uguale a zero
c. Maggiore di zero

Parametro nomi – Questo parametro può essere:

a. Lunghezza maggiore di zero
b. Lunghezza uguale a zero

Inoltre:

c. Uno o più valori sono stringhe nulle
d. Uno o più valori sono stringhe che contengono simboli o numeri
e. Due o più valori sono uguali

Parametro numeri – Questo parametro può essere:

a. Lunghezza maggiore di zero
b. Lunghezza uguale a zero

Inoltre:

a. Numero minore di zero

b. Numero uguale a zero
c. Numero compreso tra 1 e 100
d. Numero uguale a 101
e. Numero maggiore di 101

(n, m) parameters – Sia che n che m devono avere lo stesso valore, maggiore o uguale a 0

a. n è minore di m
b. n è uguale a m
c. n è maggiore di m

(n, nomi) parameters – Il valore di n deve essere uguale alla lunghezza dell'array nomi (nomi.lenght)

a. n è minore di nomi.lenght
b. n è uguale a nomi.lenght
c. n è maggiore di nomi.lenght

(m, numeri) parameters – Il valore di m deve essere uguale alla lunghezza dell'array numeri (numeri.lenght)

a. m è minore di numeri.lenght
b. m è uguale a numeri.lenght
c. m è maggiore di numeri.lenght

Possibili output:

L'output è un dato di tipo ArrayList<String> che può essere:

a. Lunghezza maggiore di zero
b. Lunghezza uguale a zero

Ogni valore individuale può essere:

f. Stringa "<nome> <Risultato_Quadrato_Cubo>"

Inoltre, la funzione può generale le seguenti eccezioni:

- CubeSquareLimitsExceededException;
- NumberEqualZeroException;
- DifferentLenghtNomiNumeriException;
- LenResultLessEqualZeroException;
- NegativeNumberException;
- NumberOffpointException;
- NomeNullException;
- NomeContainsNumbersException;
- DuplicatesNomiException.

4 Analisi dei boundaries

1	n	a. n minore di zero – *In/On Point* b. n uguale a zero – *On Point* c. n maggiore di zero – *Out/Off Point*
2	m	e. m minore di zero – *In/On Point* f. m uguale a zero – *On Point* g. m maggiore di zero – *Out/Off Point*
3	nomi	a. Lunghezza maggiore di zero – *On Point* b. Lunghezza uguale a zero – *In/On Point*
4	numeri	a. Lunghezza maggiore di zero – *On Point* b. Lunghezza uguale a zero – *In/On Point*
5	n – m	a. n è minore di m – *In/On Point* b. n è uguale a m – *Off Point* c. n è maggiore di m – *In/On Point*
6	n – nomi	d. n è minore di nomi.lenght – *In/On Point* e. n è uguale a nomi.lenght – *Off Point* f. n è maggiore di nomi.lenght – *In/On Point*
7	m - numeri	g. m è minore di numeri.lenght – *In/On Point* h. m è uguale a numeri.lenght – *Off Point* i. m è maggiore di numeri.lenght – *In/On Point*

5 Progettazione dei casi di test

Per capire quanti casi di test dobbiamo progettare, effettuiamo un calcolo. Dal calcolo risultano che bisogna effettuare 3 * 3 * 4 * 7 * 3 * 3 * 3 = 6804 test. Essendo troppi test, analizziamo i casi che possono essere testati una sola volta.

Dopo aver trovato i vari input, output e boundaries, passiamo ora a concretizzare i vari casi di test seguendo la seguente strategia:

a. Casi in cui n è minore di m:
1. Caso in cui n e m sono minore di zero viene testato una volta con n minore di m;
T2. Caso in cui n e m sono maggiore di zero viene testato una volta con n minore di m;

b. Casi in cui n è maggiore di m:
T3. Caso in cui n e m sono minore di zero viene testato una volta con n maggiore di m;
T4. Caso in cui n e m sono maggiore di zero viene testato una volta n maggiore di m;

c. Casi in cui n è uguale a m:
T5. Caso in cui n e m sono minore di zero viene testato una volta con n uguale a m;
T6. Caso in cui n e m sono uguale a zero viene testato una volta n uguale a m;
T7. Caso in cui n e m sono maggiore di zero viene testato una volta n uguale a m;

d. Caso in cui nomi è un array vuoto:
T8. Caso in cui nomi.lenght è uguale a zero;

e. **Caso in cui nomi è un array vuoto:**
   T9. Caso in cui numeri.lenght è uguale a zero;

f. **Casi in cui la lunghezza di numeri sia maggiore o uguale a uno:**
   T10. Caso in cui numeri.lenght è maggiore o uguale a uno, uno o più valori sono minori di zero;
   T11. Caso in cui numeri.lenght è maggiore o uguale a uno, uno o più valori sono uguali a zero;
   T12. Caso in cui numeri.lenght è maggiore o uguale a uno, uno o più valori sono compresi tra 1 e 100;
   T13. Caso in cui numeri.lenght è maggiore o uguale a uno, uno o più valori sono uguali a 101;
   T14. Caso in cui numeri.lenght è maggiore o uguale a uno, uno o più valori sono maggiori di 101;

g. **Casi un cui la lunghezza di nomi è maggiore o uguale a 1:**
   T15. Caso in cui nomi.lenght è maggiore o uguale a uno e n e minore della lunghezza di nomi;
   T16. Caso in cui nomi.lenght è maggiore o uguale a uno e n e uguale alla lunghezza di nomi;
   T17. Caso in cui nomi.lenght è maggiore o uguale a uno e n e maggiore della lunghezza di nomi;
   T18. Caso in cui nomi.lenght è maggiore o uguale a uno, uno o più valori sono stringhe nulle;
   T19. Caso in cui nomi.lenght è maggiore o uguale a uno, uno o più valori sono stringhe che contengono simboli o numeri;
   T20. Caso in cui nomi.lenght è maggiore o uguale a uno, due o più valori sono uguali tra loro;

h. **Casi un cui la lunghezza di nomi è maggiore o uguale a 1:**
   T21. Caso in cui numeri.lenght è maggiore o uguale a uno e m e minore della lunghezza di numeri;
   T22. Caso in cui numeri.lenght è maggiore o uguale a uno e m e uguale alla lunghezza di numeri;
   T23. Caso in cui numeri.lenght è maggiore o uguale a uno e m e maggiore della lunghezza di numeri;

6 Automazione dei casi di test

In questa parte dell'homework non è stata richiesta alcuna implementazione dei test descritti nella fase precedente.

7 Casi di test aggiuntivi

Non abbiamo trovato casi di test aggiuntivi.

Structural Test

Introduzione

Dopo aver progettato la suite di test black-box (assieme ai test aggiuntivi), progettiamo e sviluppiamo una suite di test white-box con lo Structural Testing.

Creazione dei casi di test

In generale scegliere un criterio di coverage è particolarmente importante al fine di testare in modo efficiente il codice.

Nel nostro codice sono presenti molte diramazioni nelle istruzioni, quali if, if-else e for. In particolare, abbiamo 4 condizioni if, 4 condizioni if-else e 4 cicli for.

Come vedremo nel corso dei paragrafi, sono stati utilizzati i seguenti criteri di coverage con le loro formule per calcolarlo:

- **Branch + Condition Coverage**:

$$condition + branch\ coverage = \frac{branches\ covered + conditions\ covered}{number\ of\ branches\ + number\ of\ conditions} * 100\%$$

- **Branch Coverage**:

$$branch\ coverage = \frac{branches\ covered}{total\ number\ of\ branches} * 100\%$$

- **Path Coverage**

Condizione if (1)

```
if (n <= 0 || m <= 0)
      throw new LenResultLessEqualZeroException();
```

In questa condizione abbiamo deciso di adottare il **Condition + Branch Coverage** in quanto è necessario testare sia le singole condizioni quando sono True e quando sono False sia quando il branch è False, in quanto la condizione di branch True è testata nella condition coverage.

Casi di Test	`n <= 0`	`m <= 0`	Decisione
T1	T	F	T
T2	F	T	T
T3	F	F	F

$$condition + branch\ coverage = \frac{2+4}{2+4} * 100\% = \mathbf{100}\%$$

Di seguito la sua implementazione:

```java
@Test
public void lenResultLessEqualZero() throws LenResultLessEqualZeroException, NomeContainsNumbersException,
        NumberOffpointException, CubeSquareLimitsExceededException, DifferentLenghtNomiNumeriException,
        NomeNullException, DuplicatesNomiException {

    Integer[] numeri = {3, 2};
    String[] nomi = {"Benedetto", "Giovanni"};
    ArrayList<String> result = new ArrayList<>();
    result.add("Benedetto 27");
    result.add("Giovanni 4");

    //T01
    assertThrows(LenResultLessEqualZeroException.class, () -> {
        squareCubeList(2, 0, nomi, numeri);
    });

    //T02
    assertThrows(LenResultLessEqualZeroException.class, () -> {
        squareCubeList(-2, 2, nomi, numeri);
    });

    //T03
    assertEquals(squareCubeList(2, 2, nomi, numeri), result);
}
```

Condizione if (2)

```java
if (n != m || n != nomi.length)
      throw new DifferentLenghtNomiNumeriException();
```

In questa condizione abbiamo deciso di adottare il **Condition + Branch Coverage** in quanto è necessario testare sia le singole condizioni quando sono True e quando sono False sia quando il branch è False, in quanto la condizione di branch True è testata nella condition coverage.

Casi di Test	`n != m`	`n != nomi.lenght`	Decisione
T4	T	F	T
T5	F	T	T
T6	F	F	F

$$condition + branch\ coverage = \frac{2+4}{2+4} * 100\% = \mathbf{100}\%$$

Di seguito la sua implementazione:

```java
@Test
    public void lenResultLessEqualZero() throws LenResultLessEqualZeroException,
NomeContainsNumbersException,
            NumberOffpointException, CubeSquareLimitsExceededException,
DifferentLenghtNomiNumeriException,
            NomeNullException, DuplicatesNomiException {

        Integer[] numeri = {3, 2};
        String[] nomi = {"Benedetto", "Giovanni"};
        ArrayList<String> result = new ArrayList<>();
        result.add("Benedetto 27");
        result.add("Giovanni 4");
```

```java
        //T01
        assertThrows(LenResultLessEqualZeroException.class, () -> {
            squareCubeList(2, 0, nomi, numeri);
        });

        //T02
        assertThrows(LenResultLessEqualZeroException.class, () -> {
            squareCubeList(-2, 2, nomi, numeri);
        });

        //T03
        assertEquals(squareCubeList(2, 2, nomi, numeri), result);
    }
```

Condizione if (3)

```java
if (nomi[i] == nomi[j] && i != j)
      throw new DuplicatesNomiException();
```

In questa condizione abbiamo deciso di adottare il **Condition + Branch Coverage** in quanto è necessario testare sia le singole condizioni quando sono True e quando sono False sia quando il branch è True, in quanto la condizione di branch False è testata nella condition coverage.

Casi di Test	`nomi[i] == nomi[j]`	`i != j`	Decisione
T7	T	T	T
T8	F	T	F

$$condition + branch\ coverage = \frac{2+4}{2+4} * 100\% = \mathbf{100}\%$$

Di seguito la sua implementazione:

```java
@Test
public void duplicatesNomi() throws LenResultLessEqualZeroException,
NomeContainsNumbersException,
        NumberOffpointException, CubeSquareLimitsExceededException,
DifferentLenghtNomiNumeriException,
        NomeNullException, DuplicatesNomiException {

    Integer[] numeri = {3, 2, 1};
    String[] nomi = {"Benedetto", "Giovanni", "Mario"};
    String[] nomiDupl = {"Benedetto", "Giovanni", "Giovanni"};
    ArrayList<String> result = new ArrayList<>();
    result.add("Benedetto 27");
    result.add("Giovanni 4");
    result.add("Mario 1");

    //T07
    assertThrows(DuplicatesNomiException.class, () -> {
        squareCubeList(3, 3, nomiDupl, numeri);
    });
```

```
    //T08
    assertEquals(squareCubeList(3, 3, nomi, numeri), result);
}
```

Condizione if (4)

```
if (nomi[i].matches(".*\\d.*"))
      throw new NomeContainsNumbersException();
```

In questa condizione abbiamo deciso di adottare il **Branch Coverage** nella quale si neccessita di testare solo quando la singola condizione è True e quando la singola condizione è False

Casi di Test	`nomi[i].matches(".*\\d.*")`	Decisione
T9	T	T
T10	F	F

$$branch\ coverage = \frac{2}{2} * 100\% = \mathbf{100}\%$$

Di seguito la sua implementazione:

```
@Test
public void nomeContainsNumbers() throws LenResultLessEqualZeroException,
NomeContainsNumbersException,
        NumberOffpointException, CubeSquareLimitsExceededException,
DifferentLenghtNomiNumeriException,
        NomeNullException, DuplicatesNomiException {

    Integer[] numeri = {3, 2};
    String[] nomi = {"Benedetto", "Giovanni"};
    String[] nomiNum = {"Benedetto2", "Giovanni"};
    ArrayList<String> result = new ArrayList<>();
    result.add("Benedetto 27");
    result.add("Giovanni 4");

    //T07
    assertThrows(NomeContainsNumbersException.class, () -> {
        squareCubeList(2, 2, nomiNum, numeri);
    });

    //T08
    assertEquals(squareCubeList(2, 2, nomi, numeri), result);
}
```

Condizione if-else (1)

```
if (numeri[i] <= 0 || numeri[i] >= 101)
      throw new NumberOffpointException();
else if …
```

In questa condizione abbiamo deciso di adottare il **Condition + Branch Coverage** in quanto è necessario testare sia le singole condizioni quando sono True e quando sono False sia quando il branch è False, in quanto la condizione di branch True è testata nella condition coverage.

Casi di Test	numeri[i] <= 0	numeri[i] >= 101	Decisione
T11	T	F	T
T12	F	T	T
T13	F	F	F

$$condition + branch\ coverage = \frac{2+4}{2+4} * 100\% = \mathbf{100}\%$$

Di seguito la sua implementazione:

```java
@Test
public void numberOffpoint() throws LenResultLessEqualZeroException,
NomeContainsNumbersException,
        NumberOffpointException, CubeSquareLimitsExceededException,
DifferentLenghtNomiNumeriException,
        NomeNullException, DuplicatesNomiException {

    Integer[] numeri = {3, 2};
    Integer[] numeri101 = {3, 105};
    Integer[] numeri0 = {3, 0};
    String[] nomi = {"Benedetto", "Giovanni"};
    ArrayList<String> result = new ArrayList<>();
    result.add("Benedetto 27");
    result.add("Giovanni 4");

    //T11
    assertThrows(NumberOffpointException.class, () -> {
        squareCubeList(2, 2, nomi, numeri0);
    });

    //T12
    assertThrows(NumberOffpointException.class, () -> {
        squareCubeList(2, 2, nomi, numeri101);
    });

    //T13
    assertEquals(squareCubeList(2, 2, nomi, numeri), result);
}
```

Condizione if-else (2)

```
if (((numeri[i] % 2) == 0 && numeri[i] >= 10) ||
      ((numeri[i] % 2) != 0 && numeri[i] >= 7))
      throw new CubeSquareLimitsExceededException();
else if …
```

In questa condizione abbiamo deciso di adottare il **Path Coverage**. Tale scelta è derivante dal fatto che abbiamo provato a effettuare MC/DC, tuttavia dai N + 1 test che dovrebbero uscire con tale criterio ne sono usciti circa 8. Per questo motivo abbiamo decidi di adottare il Path Coverage.

Assumiamo:

A = `(numeri[i] % 2) == 0`
B = `(numeri[i] >= 10)`
C = `(numeri[i] % 2) != 0`
D = `(numeri[i] >= 7)`

Successivamente, abbiamo creato una tabella contenente tutte le possibili combinazioni della condizione iniziale ottenendo quindi 16 possibili casi test.

Casi di Test	A	B	C	D	Decisione
1	T	T	T	T	T
2	T	T	T	F	T
3	T	T	F	T	T
4	T	T	F	F	T
5	T	F	T	T	T
6	T	F	T	F	F
7	T	F	F	T	F
8	T	F	F	F	F
9	F	T	T	T	T
10	F	T	T	F	F
11	F	T	F	T	F
12	F	T	F	F	F
13	F	F	T	T	T
14	F	F	T	F	F
15	F	F	F	T	F
16	F	F	F	F	F

Tuttavia, durante l'analisi dei casi di test, abbiamo notato **che alcuni test non possono essere effettuati**.

Ad esempio, consideriamo il primo caso di test:

Casi di Test	A	B	C	D	Decisione
1	T	T	T	T	T

Questo test non può essere eseguito poiché non è possibile che sia vera sia la condizione A sia la condizione C contemporaneamente. Se la condizione A è True (ovvero il numero è pari), allora la condizione C deve essere False (numero dispari). Pertanto, questo test viene scartato.

Stessa cosa vale per il seguente test:

Casi di Test	A	B	C	D	Decisione
4	T	T	T	T	T

In questo caso, le condizioni per i numeri pari e dispari sono corrette (numero pari, quindi A è True e C è False). Tuttavia, anche questo test non può essere effettuato poiché, se la condizione B è True (numero >= 10), la condizione C non può essere Falsa (se il numero è >= 10, allora è anche > 7). Pertanto, anche questo test viene scartato.

Applicando questo ragionamento a tutti gli altri test presenti nella tabella, osserviamo che solo alcuni di essi possono essere effettuati (sono quelli evidenziati in blu):

Casi di Test	A	B	C	D	Decisione
1	T	T	T	T	T
2	T	T	T	F	T
T14	T	T	F	T	T
4	T	T	F	F	T
5	T	F	T	T	T
6	T	F	T	F	F
7	T	F	F	T	F
T15	T	F	F	F	F
T16	F	T	T	T	T
10	F	T	T	F	F
11	F	T	F	T	F
12	F	T	F	F	F
T17	F	F	T	T	T
T18	F	F	T	F	F
15	F	F	F	T	F
16	F	F	F	F	F

Tra i 16 casi di test, **solo 5 possono essere eseguiti**:

- 3 test (**T14, T16, T17**) generano una decisione True (pertanto possiamo verificare se viene effettivamente lanciata l'eccezione CubeSquareLimitsExceededException).
- 2 test (**T15, T18**) generano una decisione False (ovvero il ramo else viene eseguito correttamente e il programma prosegue senza lanciare eccezioni).

Di seguito la sua implementazione:

```
@Test
    public void cubeSquareLimitsExceeded() throws LenResultLessEqualZeroException,
NomeContainsNumbersException,
            NumberOffpointException, CubeSquareLimitsExceededException,
DifferentLenghtNomiNumeriException,
            NomeNullException, DuplicatesNomiException {

        Integer[] numeri = {3, 2};
        Integer[] numeriPariLimits = {10, 6};
        Integer[] numeriDispariLimits10 = {11, 5};
        Integer[] numeriDispariLimits7 = {7, 5};
        String[] nomi = {"Benedetto", "Giovanni"};
        ArrayList<String> result = new ArrayList<>();
        result.add("Benedetto 27");
        result.add("Giovanni 4");
```

```java
        //T14
        assertThrows(CubeSquareLimitsExceededException.class, () -> {
            squareCubeList(2, 2, nomi, numeriPariLimits);
        });

        //T16
        assertThrows(CubeSquareLimitsExceededException.class, () -> {
            squareCubeList(2, 2, nomi, numeriDispariLimits10);
        });

        //T17
        assertThrows(CubeSquareLimitsExceededException.class, () -> {
            squareCubeList(2, 2, nomi, numeriDispariLimits7);
        });

        //T15 = T18
        assertEquals(squareCubeList(2, 2, nomi, numeri), result);
    }
```

Condizione if-else (3)

```
if (nomi[i] == "" || nomi[i] == null)
      throw new NomeNullException();
else if …
```

In questa condizione abbiamo deciso di adottare il **Condition + Branch Coverage** in quanto è necessario testare sia le singole condizioni quando sono True e quando sono False sia quando il branch è False, in quanto la condizione di branch True è testata nella condition coverage.

Casi di Test	`nomi[i] == ""`	`nomi[i] == null`	Decisione
T19	T	F	T
T20	F	T	T
T21	F	F	F

$$condition + branch\ coverage = \frac{2+4}{2+4} * 100\% = \mathbf{100}\%$$

Di seguito la sua implementazione:

```java
@Test
public void nomeNullOrEmpty() throws LenResultLessEqualZeroException,
NomeContainsNumbersException,
        NumberOffpointException, CubeSquareLimitsExceededException,
DifferentLenghtNomiNumeriException,
        NomeNullException, DuplicatesNomiException {

    Integer[] numeri = {3, 2};
    String[] nomi = {"Benedetto", "Giovanni"};
    String[] nomiNull = {null, "Giovanni"};
    String[] nomiEmpty = {"", "Giovanni"};
    ArrayList<String> result = new ArrayList<>();
    result.add("Benedetto 27");
    result.add("Giovanni 4");

    //T19
    assertThrows(NomeNullException.class, () -> {
        squareCubeList(2, 2, nomiEmpty, numeri);
    });

    //T20
    assertThrows(NomeNullException.class, () -> {
        squareCubeList(2, 2, nomiNull, numeri);
    });

    //T21
    assertEquals(squareCubeList(2, 2, nomi, numeri), result);
}
```

Risultati

Inoltre, i risultati di coverage (mediante l'utilizzo di JaCoCo) sono i seguenti:

- Class % = 100%;
- Method % = 100%;
- Line % = 100% (23/23);
- Branch % = 97% (39/40)

La motivazione del 97% di branch coverage sta nel fatto che non viene verificata per via del ciclo for; quindi, secondo noi non è possibile testarla.

Homework 3

Progettare e implementare una suite di Property-based test e generare il report delle statistiche opportuno per quei PBT.

Introduzione

Descrizione

IvaProdotto è una classe implementata con Java 19. Questa classe possiede un costruttore nella quale si passano tre valori: String nome, TipoProdotto tipo, float prezzoNetto. Il primo è una stringa che rappresenta il nome di un prodotto, il secondo invece è un tipo di dato enum che contiene le varie tipologie di prodotti dove si applica e dove non si applica l'IVA e infine un float che rappresenta il prezzo del prodotto.

La classe possiede tre metodi che sono:

- `getIva()` che restituisce l'IVA per i prodotti che non sono esenti da iva altrimenti restituisce 0.
  NB: per semplicità abbiamo assunto che l'IVA è pari al 22%.
- `getPrezzoNetto()` che restituisce il prezzo netto inserito nel costruttore quando si è istanziato l'oggetto
- `setPrezzoNetto(float prezzoNetto)` che imposta il prezzo netto se questo è maggiore di zero. Se il prezzo netto dato in input è minore o uguale a zero viene gerata l'eccezione `IllegalArgumentException`.

Codice

```java
public class IvaProdotto {

    private String nome;
    private TipoProdotto tipo;
    private float prezzoNetto;

    public IvaProdotto(String nome, TipoProdotto tipo, float prezzoNetto) {
        this.nome = nome;
        this.tipo = tipo;
        this.prezzoNetto = setPrezzoNetto(prezzoNetto);
    }

    public float getIva() {
        if (tipo.equals(TipoProdotto.PANE) || tipo.equals(TipoProdotto.PASTA) || tipo.equals(TipoProdotto.RISO) ||
                tipo.equals(TipoProdotto.FRUTTA) || tipo.equals(TipoProdotto.VERDURA) || tipo.equals(TipoProdotto.CARNE) ||
                tipo.equals(TipoProdotto.PESCE) || tipo.equals(TipoProdotto.UOVA) || tipo.equals(TipoProdotto.FARMACO) ||
                tipo.equals(TipoProdotto.LIBRO) || tipo.equals(TipoProdotto.GIORNALE) || tipo.equals(TipoProdotto.RIVISTA) ||
                tipo.equals(TipoProdotto.SANITA) || tipo.equals(TipoProdotto.SERVIZI_EDUCATIVI))
            return 0.0f;
        else
            return prezzoNetto * 0.22f;
    }

    public float getPrezzoNetto() {
        return prezzoNetto;
    }

    public float setPrezzoNetto(float prezzoNetto) {
        if (prezzoNetto <= 0)
            throw new IllegalArgumentException();
        return prezzoNetto;
    }

}
```

Implementazione Property-Based Test

Passiamo ora a realizzare i Property-Based Test con jqwik.

Property-Based Test di setPrezzoNetto()

Il primo test che andiamo a realizzare è quello dove il metodo setPrezzo() non è valido, ossia genera eccezione in quanto l'input inserto (float prezzoNetto) è minore o uguale a zero.

Abbiamo deciso di generare 40 prove, nella quale viene generato un numero randomico compreso tra -100.00 e 0.00 (incluso). Tale valore rappresenta il prezzo netto che sarà utilizzato durante l'istanza della classe IvaProdotto. Ciò che dovrebbe restituire questa situazione è una eccezione di IllegalArgumentException.

Di seguito la sua implementazione:

```
@Property(tries = 40)
@Report(Reporting.GENERATED)
@StatisticsReport(format = Histogram.class)
void invalidSetPrezzoNettoTest(
        @ForAll
        @FloatRange(min = -100.0f, max = 0, maxIncluded = true)
        float prezzoNetto) {

    assertThrows(IllegalArgumentException.class, () -> {
        new IvaProdotto("Product", TipoProdotto.ALTRO, prezzoNetto);
    });
    Statistics.label("Count prezzi netti non valido").collect(prezzoNetto);
}
```

Il secondo test che andiamo a realizzare è quello dove il metodo setPrezzo() assegna il prezzo netto in modo corretto, in quanto il suo valore è maggiore di zero.

Anche in questo caso abbiamo deciso di generare 40 prove, nella quale viene generato un numero randomico compreso tra 0.00 e 100.00 (incluso) che rappresenta il prezzo netto del prodotto che sarà utilizzato durante l'istanza della classe IvaProdotto. Ciò che dovrebbe restituire questa situazione è il prezzo effettivo dato in input alla funzione.

Di seguito la sua implementazione:

```
@Property(tries = 40)
@Report(Reporting.GENERATED)
@StatisticsReport(format = Histogram.class)
void passSetPrezzoNettoTest(
        @ForAll RoundingMode mode,
        @ForAll
        @FloatRange(min = 0, max = 100.0f, maxIncluded = true, minIncluded = false)
        float prezzoNetto) {

    IvaProdotto ivaProdotto = new IvaProdotto("Product", TipoProdotto.ALTRO, prezzoNetto);
    assertEquals(ivaProdotto.getPrezzoNetto(), prezzoNetto);

    float result = ivaProdotto.getPrezzoNetto();
    Statistics.label("Prezzi netti accettati").collect(result);
}
```

Property-Based Test di getIva()

Il terzo test che andiamo a realizzare è quello dove il metodo getIva() restituisce un valore pari a zero. Questo perché il TipoProdotto è un valore pari a uno dei seguenti:

PANE, PASTA, RISO, FRUTTA, VERDURA, CARNE, PESCE, UOVA, FARMACO, LIBRO, GIORNALE, RIVISTA, SANITÀ o SERVIZI_EDUCATIVI.

In questo test abbiamo definito tre costati:

- LEN_TIPO_PRODOTTO_ENUM: intero che indica tutti i valori contenuti nella enum TipoProfilo;
- LEN_TIPO_PRODOTTO_NO_IVA_ENUM: intero che indica tutti i valori contenuti nella enum TipoProfilo che non sono sottoposti a iva;
- LEN_TIPO_PRODOTTO_IVA_ENUM: intero che indica tutti i valori contenuti nella enum TipoProfilo che sono sottoposti a iva.

In questo caso abbiamo deciso di generare LEN_TIPO_PRODOTTO_NO_IVA_ENUM test per ossia uno per ciascun valore di TipoProfilo. Inoltre, generiamo due numeri randomici:

1. randomTipoProdotto: numero int randomico compreso tra 0 e LEN_TIPO_PRODOTTO_NO_IVA_ENUM che rappresenta uno specifico TipoProdotto a cui non viene applicata l'iva.
2. randomPrezzoNetto : numero float randomico compreso tra 0.00 e 100.00 (incluso) che rappresenta il prezzo netto.

Entrambi i valori saranno utilizzati durante l'istanza della classe IvaProdotto. In questa situazione si richiede che il valore dell'iva è pari a zero.

Di seguito la sua implementazione:

```
private static final int LEN_TIPO_PRODOTTO_ENUM = 28;
private static final int LEN_TIPO_PRODOTTO_NO_IVA_ENUM = 14;
private static final int LEN_TIPO_PRODOTTO_IVA_ENUM = LEN_TIPO_PRODOTTO_ENUM - LEN_TIPO_PRODOTTO_NO_IVA_ENUM;

@Property(tries = LEN_TIPO_PRODOTTO_NO_IVA_ENUM)
@Report(Reporting.GENERATED)
@StatisticsReport(format = Histogram.class)
void nullGetIvaTest(
        @ForAll
        @IntRange(min = 0, max = LEN_TIPO_PRODOTTO_NO_IVA_ENUM - 1)
        int randomTipoProdotto,
        @ForAll
        @FloatRange(min = 0, max = 100.0f, maxIncluded = true, minIncluded = false)
        float randomPrezzoNetto) {

    TipoProdotto tipoProdotto= TipoProdotto.class.getEnumConstants()[randomTipoProdotto];
    IvaProdotto ivaProdotto = new IvaProdotto("Product", tipoProdotto, randomPrezzoNetto);
    assertEquals(ivaProdotto.getIva(), 0);

    Statistics.label("Tipo prodotti senza iva").collect(tipoProdotto);
}
```

Il quarto e ultimo test che andiamo a realizzare è quello dove il metodo getIva() restituisce un valore pari al 22% del prezzo netto del prodotto. Questo perché il TipoProdotto è un valore pari a uno dei seguenti:

ABBIGLIAMENTO, ELETTRONICA, BELLEZZA, GIOCATTOLI, DOLCI, SNACK, BEVANDE_ANALCOLICHE, BEVANDE_ALCOLICHE, SERVIZI_PROFESSIONALI, TRASPORTI, RISTORAZIONE, INTRATTENIMENTO, LUSSO o ALTRO.

NB: per semplicità abbiamo assunto che l'IVA è pari al 22%.

In questo caso abbiamo deciso di generare LEN_TIPO_PRODOTTO_IVA_ENUM test ossia uno per ciascun valore di TipoProfilo. Inoltre, generiamo due numeri randomici:

3. randomTipoProdotto: numero int randomico compreso tra LEN_TIPO_PRODOTTO_NO_IVA_ENUM e LEN_TIPO_PRODOTTO_IVA_ENUM che rappresenta uno specifico TipoProdotto a cui non viene applicata l'iva.
4. randomPrezzoNetto : numero float randomico compreso tra 0.00 e 100.00 (incluso) che rappresenta il prezzo netto.

Entrambi i valori saranno utilizzati durante l'istanza della classe IvaProdotto. In questa situazione si richiede che il valore dell'iva sia pari al prezzo moltiplicato per 0.22.

Di seguito la sua implementazione:

```
private static final int LEN_TIPO_PRODOTTO_ENUM = 28;
private static final int LEN_TIPO_PRODOTTO_NO_IVA_ENUM = 14;
private static final int LEN_TIPO_PRODOTTO_IVA_ENUM = LEN_TIPO_PRODOTTO_ENUM - LEN_TIPO_PRODOTTO_NO_IVA_ENUM;

@Property(tries = LEN_TIPO_PRODOTTO_IVA_ENUM)
@Report(Reporting.GENERATED)
@StatisticsReport(format = Histogram.class)
void notNullGetIvaTest(
        @ForAll
        @IntRange(min = LEN_TIPO_PRODOTTO_NO_IVA_ENUM, max = LEN_TIPO_PRODOTTO_ENUM - 1)
        int randomTipoProdotto,
        @ForAll
        @FloatRange(min = 0, max = 100.0f, maxIncluded = true, minIncluded = false)
        float randomPrezzoNetto) {

    TipoProdotto tipoProdotto= TipoProdotto.class.getEnumConstants()[randomTipoProdotto];
    IvaProdotto ivaProdotto = new IvaProdotto("Product",  tipoProdotto, randomPrezzoNetto);
    assertEquals(ivaProdotto.getIva(), randomPrezzoNetto * 0.22f);

    float result = ivaProdotto.getPrezzoNetto();
    Statistics.label("Tipo prodotti con iva").collect(tipoProdotto);
}
```

Report Statistiche Property-Based Test

invalidSetPrezzoNettoTest()

```
timestamp = 2023-06-28T22:35:36.217769400, [IvaProdottoTest:invalidSetPrezzoNettoTest] (40) Count prezzi netti non valido =
    # |  label | count |
  ----|--------|-------|-------------------------------------------------------------------------------
    0 | -100.0 |     2 | ▬
    1 | -96.39 |     1 | ■
    2 |  -81.2 |     1 | ■
    3 | -71.81 |     1 | ■
    4 | -46.98 |     1 | ■
    5 |  -33.2 |     1 | ■
    6 | -30.12 |     1 | ■
    7 | -25.69 |     1 | ■
    8 | -23.73 |     1 | ■
    9 | -22.99 |     1 | ■
   10 | -22.96 |     1 | ■
   11 | -22.39 |     1 | ■
   12 | -22.34 |     1 | ■
   13 | -17.25 |     1 | ■
   14 | -10.21 |     1 | ■
   15 |  -6.99 |     1 | ■
   16 |  -6.63 |     1 | ■
   17 |  -6.27 |     1 | ■
   18 |  -3.62 |     1 | ■
   19 |  -3.05 |     1 | ■
   20 |  -2.67 |     1 | ■
   21 |  -2.12 |     1 | ■
   22 |  -1.16 |     1 | ■
   23 |   -1.0 |     2 | ▬
   24 |  -0.75 |     1 | ■
   25 |  -0.73 |     1 | ■
   26 |  -0.72 |     1 | ■
   27 |  -0.66 |     1 | ■
   28 |  -0.31 |     1 | ■
   29 |  -0.12 |     1 | ■
   30 |  -0.11 |     1 | ■
   31 |  -0.09 |     1 | ■
   32 |  -0.06 |     1 | ■
   33 |  -0.02 |     1 | ■
   34 |  -0.01 |     1 | ■
   35 |   0.0 |     3 | ▬▬
```

```
                                  |--------------------jqwik--------------------
tries = 40                        | # of calls to property
checks = 40                       | # of not rejected calls
generation = RANDOMIZED           | parameters are randomly generated
after-failure = PREVIOUS_SEED     | use the previous seed
when-fixed-seed = ALLOW           | fixing the random seed is allowed
edge-cases#mode = MIXIN           | edge cases are mixed in
edge-cases#total = 5              | # of all combined edge cases
edge-cases#tried = 5              | # of edge cases tried in current run
seed = 1574561755791868409        | random seed to reproduce generated values
```

passSetPrezzoNettoTest()

```
timestamp = 2023-06-28T22:38:21.609828700, [IvaProdottoTest:passSetPrezzoNettoTest] (40) Prezzi netti accettati =
      # | label | count |
    -----|-------|-------|-------------------------------------------------------------------------------
      0 |  0.01 |     4 | ■■■■
      1 |  0.02 |     1 | ■
      2 |  0.05 |     1 | ■
      3 |  0.07 |     1 | ■
      4 |  0.13 |     2 | ■■
      5 |  0.17 |     1 | ■
      6 |  0.18 |     1 | ■
      7 |  0.26 |     1 | ■
      8 |   0.3 |     1 | ■
      9 |  0.32 |     1 | ■
     10 |  0.62 |     1 | ■
     11 |  0.88 |     1 | ■
     12 |   1.0 |     2 | ■■
     13 |   1.4 |     1 | ■
     14 |  1.44 |     1 | ■
     15 |   1.6 |     1 | ■
     16 |  1.91 |     1 | ■
     17 |  2.48 |     1 | ■
     18 |  2.55 |     1 | ■
     19 |  3.11 |     1 | ■
     20 |  4.64 |     1 | ■
     21 |  5.62 |     1 | ■
     22 |  6.98 |     1 | ■
     23 |  9.58 |     1 | ■
     24 | 21.36 |     1 | ■
     25 | 24.27 |     1 | ■
     26 | 26.51 |     1 | ■
     27 | 31.52 |     1 | ■
     28 | 32.71 |     1 | ■
     29 | 41.15 |     1 | ■
     30 | 49.48 |     1 | ■
     31 |  64.1 |     1 | ■
     32 |  84.9 |     1 | ■
     33 | 100.0 |     2 | ■■
```

```
                                  |--------------------jqwik--------------------
tries = 40                        | # of calls to property
checks = 40                       | # of not rejected calls
generation = RANDOMIZED           | parameters are randomly generated
after-failure = PREVIOUS_SEED     | use the previous seed
when-fixed-seed = ALLOW           | fixing the random seed is allowed
edge-cases#mode = MIXIN           | edge cases are mixed in
edge-cases#total = 8              | # of all combined edge cases
edge-cases#tried = 6              | # of edge cases tried in current run
seed = -8695581451126997113       | random seed to reproduce generated values
```

nullGetIvaTest()

```
timestamp = 2023-06-28T22:38:55.809263800, [IvaProdottoTest:nullGetIvaTest] (14) Tipo prodotti senza iva =
      # |             label | count |
    -----|-------------------|-------|-------------------------------------------------------------------------------
      0 |              PANE |     5 | ■■■■■
      1 |             PASTA |     1 | ■
      2 |             CARNE |     2 | ■■
      3 |             PESCE |     1 | ■
      4 |           FARMACO |     1 | ■
      5 |          GIORNALE |     1 | ■
      6 | SERVIZI_EDUCATIVI |     3 | ■■■
```

```
                                  |--------------------jqwik--------------------
tries = 40                        | # of calls to property
checks = 40                       | # of not rejected calls
generation = RANDOMIZED           | parameters are randomly generated
after-failure = PREVIOUS_SEED     | use the previous seed
when-fixed-seed = ALLOW           | fixing the random seed is allowed
edge-cases#mode = MIXIN           | edge cases are mixed in
edge-cases#total = 8              | # of all combined edge cases
edge-cases#tried = 7              | # of edge cases tried in current run
seed = -4451281504230513901       | random seed to reproduce generated values
```

notNullGetIvaTest()

```
timestamp = 2023-06-28T22:39:22.096737300, [IvaProdottoTest:notNullGetIvaTest] (14) Tipo prodotti con iva =
      # |                 label | count |
  -----|----------------------|-------|--------------------------------------------------------------------------------
      0 |          ABBIGLIAMENTO |     4 | ■■■■
      1 |            ELETTRONICA |     1 | ■
      2 |               BELLEZZA |     1 | ■
      3 |             GIOCATTOLI |     1 | ■
      4 |                  SNACK |     1 | ■
      5 |      BEVANDE_ALCOLICHE |     1 | ■
      6 | SERVIZI_PROFESSIONALI |     1 | ■
      7 |           RISTORAZIONE |     1 | ■
      8 |        INTRATTENIMENTO |     2 | ■■
      9 |                  ALTRO |     1 | ■
```

```
                                  |--------------------jqwik--------------------
tries = 14                        | # of calls to property
checks = 14                       | # of not rejected calls
generation = RANDOMIZED           | parameters are randomly generated
after-failure = PREVIOUS_SEED     | use the previous seed
when-fixed-seed = ALLOW           | fixing the random seed is allowed
edge-cases#mode = MIXIN           | edge cases are mixed in
edge-cases#total = 15             | # of all combined edge cases
edge-cases#tried = 2              | # of edge cases tried in current run
seed = 1196137510827497668        | random seed to reproduce generated values
```