UNIVERSITA'
DEGLI STUDI
DI BARI

Integrazione e Test di Sistemi Software

Homework 2 - Property-Based Testing

Student

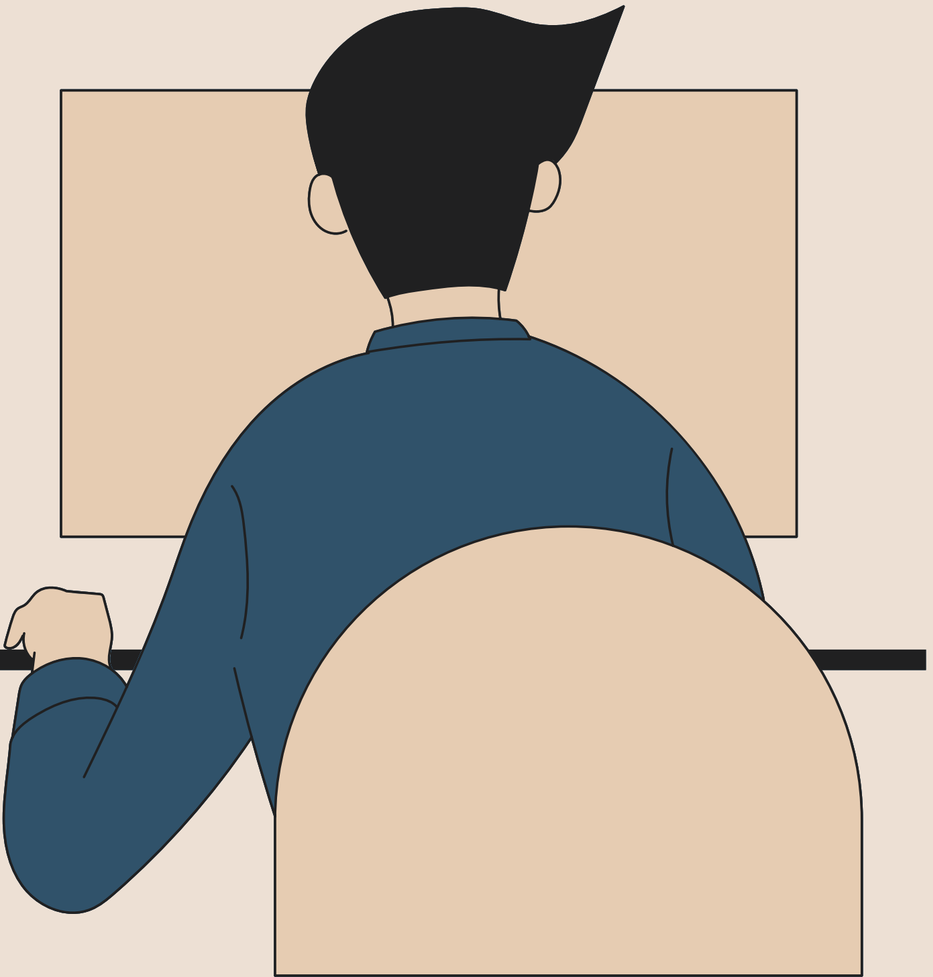
Marco Porro 717061

Student

Stefano Mansi 717448

Property-Based Testing

Definiamo **proprietà** generali che il nostro programma dovrebbe soddisfare, e un framework di testing genererà automaticamente una serie di casi di test per verificarle.



Analisi del codice

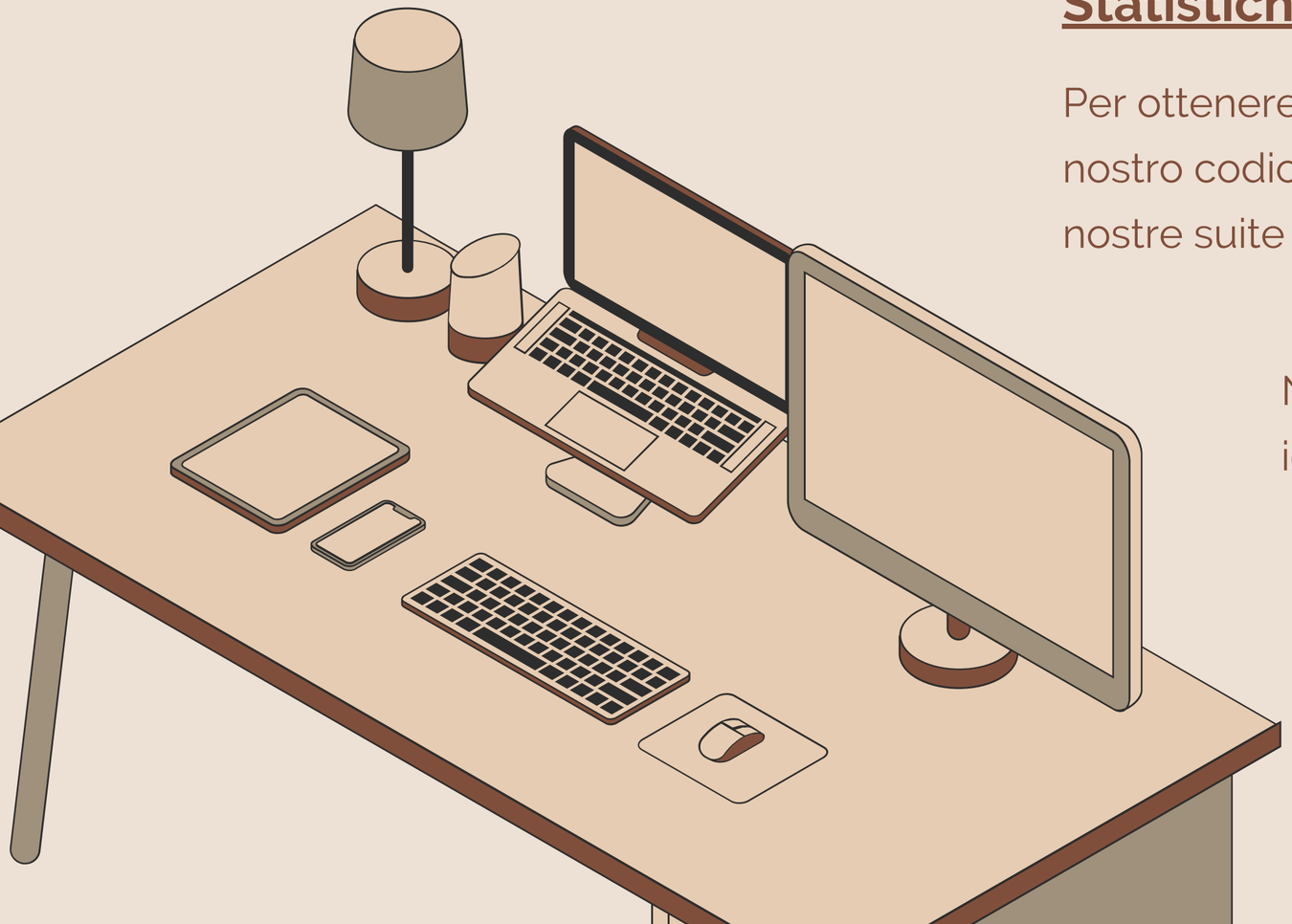
Per questo homework abbiamo deciso di utlizzare lo stesso codice ideato nell'homework precedente. Ricapitolando,Il codice è strutturato in due classi principali: **Main** e **OperazioniMath**. La prima gestisce l'interazione con l'utente attraverso l'input da console e le stampe dell'output, mentre la seconda contiene i metodi per la conversione di numeri e il calcolo delle soluzioni di un'equazione di secondo grado.

Metodi

- public String **convertiBase**(int numeroDecimale, int baseDestinazione);
- public double[] **calcolaSoluzioniEquazioneSecondoGrado**(double a, double b, double c).

Testing

Abbiamo applicato il **Property-based testing** (**PBT**) attraverso la libreria **JQwik** per coprire una vasta gamma di casi. Ogni proprietà è stata progettata per verificare specifiche condizioni e garantire il corretto funzionamento del codice.



Statistiche

Per ottenere una visione approfondita delle prestazioni del nostro codice, abbiamo incorporato l'uso di statistiche nelle nostre suite di test.

Nelle slide seguenti riportiamo tutti i test da noi ideati per coprire la maggior parte dei casi.

1 public String convertiBase(int numeroDecimale, int baseDestinazione)

testConvertiBase()

Questo caso di test verifica che il metodo **convertiBase()** restituisca una stringa valida per le basi supportate (**2, 8, 16**).
Il numero decimale da convertire è anch'esso generato casualmente nell'intervallo da **1** a **Integer.MAX_VALUE**.
Si avvale del Property-based testing approfondito per esplorare un ampio spettro di casi, generando input casuali e verificando che le conversioni rispettino le regole della base specificata.
Verifica che la stringa risultante contenga solo caratteri validi per la base di destinazione, utilizzando il metodo ausiliario **verificaCaratteriValidi().**

```
@Provide
Arbitrary<Integer> baseDestinazione() { return Arbitraries.of( ...values: 2, 8, 16); }

    Lookma96 *
    @Property
    @Report(Reporting.GENERATED)
    @StatisticsReport(format = Histogram.class)
    void testConvertiBase(@ForAll @IntRange(min=1, max=Integer.MAX_VALUE) int numeroDecimale,
                          @ForAll @From ("baseDestinazione") int baseDestinazione) {

        String risultatoMetodo = operazioniMath.convertiBase(numeroDecimale, baseDestinazione);

        assertTrue(verificaCaratteriValidi(risultatoMetodo, baseDestinazione));

        Statistics.collect(baseDestinazione == 2 ? "Binario" : baseDestinazione == 8 ? "Ottale" :
                baseDestinazione == 16 ? "Esadecimale" : "Base Sbagliata" );
    }

1 usage   Lookma96
boolean verificaCaratteriValidi(String risultato, int baseDestinazione) {
    switch (baseDestinazione) {
        case 2:
            return risultato.matches( regex: "[01]+");
        case 8:
            return risultato.matches( regex: "[0-7]+");
        case 16:
            return risultato.matches( regex: "[0-9A-Fa-f]+");
        default:
            return false;
    }
}
```

Statistica:

Il test utilizza una statistica di tipo istogramma (**Histogram**). L'intento è tenere traccia del numero di volte che il metodo è stato testato con successo per una specifica base di destinazione. La statistica categorizza i risultati in quattro categorie: "**Binario**", "**Ottale**", "**Esadecimale**", e "**Base Sbagliata**".

```
timestamp = 2024-02-05T18:23:35.771011, [PropertyProvaTest:testConvertiBase] (1000) statistics =
       # |       label | count |
    -----|-------------|-------|-----------------------------------------------------------------------------
       0 |     Binario |   356 | ##############################################################################
       1 | Esadecimale |   345 | ########################################################################
       2 |      Ottale |   299 | ############################################################
```

La statistica offre una visione dettagliata delle prestazioni del metodo in base alla base di destinazione. Per esempio, se il test ha successo quando la base di destinazione è **2**, la statistica "**Binario**" sarà incrementata. Allo stesso modo, se la base di destinazione è **8**, **16**, o è **errata**, verranno incrementate le **statistiche appropriate.**

La statistica mostra che il test è stato eseguito un totale di **1000** volte e ciascuna delle basi di destinazione è stata testata un numero significativo di volte (**323** volte "**Binario**", **353** volte "**Esadecimale**" e **324** volte "**Ottale**"), indicando un'adeguata copertura dei casi nel test.

1 **public String convertiBase(int numeroDecimale, int baseDestinazione)**

conversioneBaseGenericaRestituisceStringa()

Questo caso di test assicura che il metodo **convertiBase()** restituisca sempre una stringa non nulla, indipendentemente dai valori specifici della conversione. Il test è parametrizzato, generando casualmente un numero decimale nell'intervallo da **1** a **Integer.MAX_VALUE** e una base di destinazione tra **2**, **8**, e **16**.

Si avvale del Property-based testing per testare questa proprietà su un insieme casuale di input. Verifica che la stringa risultante non sia nulla (**assertNotNull**).

```
@Property
@Report(Reporting.GENERATED)
@StatisticsReport(format = Histogram.class)
void conversioneBaseGenericaRestituisceStringa(@ForAll @IntRange(min=1, max=Integer.MAX_VALUE)int numeroDecimale,
                                               @ForAll @From("baseDestinazione") int baseDestinazione) {

    String risultato = operazioniMath.convertiBase(numeroDecimale, baseDestinazione);
    assertNotNull(risultato);

    Statistics.collect(risultato != null ? "Converte a stringa non nulla" : "Converte a stringa nulla");
}
```

✓ conversioneBaseGenericaRestituisceStringa 180 ms

Statistica:

Il test utilizza una statistica di tipo istogramma (**Histogram**). La statistica tiene traccia del numero di volte che la conversione ha restituito una **stringa non nulla**. Le categorie della statistica sono "**Converte a stringa non nulla**" e "**Converte a stringa nulla**"

```
timestamp = 2024-02-05T18:23:35.451368800, [PropertyProvaTest:conversioneBaseGenericaRestituisceStringa] (1000) statistics =
       # |                         label | count |
    -----|-------------------------------|-------|----------------------------------------------------------------------------------------
       0 | Converte a stringa non nulla |  1000 | ########################################################################################
```

La statistica offre un'indicazione del comportamento del metodo rispetto alla generazione di stringhe non nulle. Se il test ha successo, verrà incrementata la statistica "**Converte a stringa non nulla**"; in caso contrario, sarà incrementata la statistica "**Converte a stringa nulla**".

La statistica riportata mostra che su un totale di **1000** esecuzioni del test, in tutte le occasioni il test ha avuto successo.

1 **public String convertiBase(int numeroDecimale, int baseDestinazione)**

conversioneBaseRestituisceStringaVuotaPerNumeroNegativo()

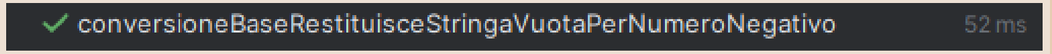
Questo caso di test verifica che la conversione restituisca una stringa vuota quando il numero decimale minore o uguale a zero. Il test è parametrizzato, generando casualmente un numero decimale nell'intervallo da **Integer.MIN_VALUE** a **0** e una base di destinazione tra **2**, **8**, e **16**. Utilizza il Property-based testing per esplorare input negativi, controllando che la conversione sia gestita correttamente.
Verifica che la stringa risultante sia vuota (**assertEquals**("", **risultato**)).

```
@Property
@Report(Reporting.GENERATED)
@StatisticsReport(format = Histogram.class)
void conversioneBaseRestituisceStringaVuotaPerNumeroNegativo(@ForAll @IntRange(min=Integer.MIN_VALUE, max=0)int numeroDecimale,
                                                             @ForAll @From("baseDestinazione") int baseDestinazione) {

    String risultato = operazioniMath.convertiBase(numeroDecimale, baseDestinazione);
    assertEquals( expected: "",risultato);

    Statistics.collect(risultato.isEmpty() ? "Converte a stringa vuota" : "Non converte a stringa vuota");

}
```



Statistica:

Ill test utilizza una statistica di tipo istogramma (**Histogram**). La statistica tiene traccia del numero di volte che la conversione ha restituito una stringa vuota. Le categorie della statistica sono **"Converte a stringa vuota**" e "**Non converte a stringa vuota**"

```
timestamp = 2024-02-05T18:23:35.504222400, [PropertyProvaTest:conversioneBaseRestituisceStringaVuotaPerNumeroNegativo] (1000) statistics =
      # |                    label | count |
    ----|--------------------------|-------|----------------------------------------------------------------------------------------
      0 | Converte a stringa vuota |  1000 | ########################################################################################
```

La statistica offre un'indicazione del comportamento del metodo rispetto alla gestione di numeri decimali negativi. Se il test ha successo, verrà incrementata la statistica "**Converte a stringa vuota**"; in caso contrario, sarà incrementata la statistica "**Non converte a stringa vuota”.**

La statistica riportata mostra che su un totale di **1000** esecuzioni del test, in tutte le occasioni il test ha avuto successo.

1 public String convertiBase(int numeroDecimale, int baseDestinazione)

conversioneBaseNonSupportata()

Questo caso di test assicura che venga restituito il messaggio di errore corretto quando si tenta di convertire in una base non supportata. Il test è parametrizzato, generando casualmente un numero decimale e una base di destinazione, assicurandosi che la base di destinazione **non** sia né **2**, né **8**, né **16.**

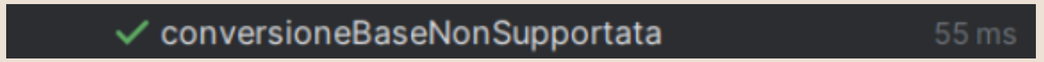
Il Property-based testing è utilizzato per testare il comportamento del metodo in presenza di basi non valide.

Verifica che il risultato sia uguale alla stringa "**Base di destinazione non supportata**" (**assertEquals("Base di destinazione non supportata", risultato)**)

```
MarcoPrr00 *
@Property
@Report(Reporting.GENERATED)
@StatisticsReport(format = Histogram.class)
void conversioneBaseNonSupportata(@ForAll int numeroDecimale,@ForAll int baseDestinazione) {

    Assume.that( condition: baseDestinazione != 2 && baseDestinazione != 8 && baseDestinazione != 16);
    String risultato = operazioniMath.convertiBase(numeroDecimale, baseDestinazione);
    assertEquals( expected: "Base di destinazione non supportata",risultato);

    Statistics.collect(risultato.equals("Base di destinazione non supportata") ?
            "Conversione non effettuata per basi errate":"Conversione effettuata per basi errate");
}
```



Statistica:

Il test utilizza una statistica di tipo istogramma (**Histogram**). La statistica tiene traccia del numero di volte che la conversione ha restituito il messaggio di errore "**Base di destinazione non supportata**". Le categorie della statistica sono "**Conversione non effettuata per basi errate**" e "**Conversione effettuata per basi errate**".

```
timestamp = 2024-02-05T18:23:35.549233800, [PropertyProvaTest:conversioneBaseNonSupportata] (978) statistics =
      # |                                    label | count |
    ----|------------------------------------------|-------|-----------------------------------------------------------------------------------
      0 | Conversione non effettuata per basi errate |   978 | ###################################################################################
```

La statistica fornisce un'indicazione sulla frequenza con cui il metodo gestisce correttamente i casi in cui viene fornita una base di destinazione non supportata. Se il test ha successo, sarà incrementata la statistica "**Conversione non effettuata per basi errate**"; in caso contrario, sarà incrementata la statistica "**Conversione effettuata per basi errate**".

La statistica riportata mostra che su un totale di 1000 esecuzioni del test, 978 hanno soddisfatto le condizioni, in quanto viene applicata la restrizione "**Assume.that(baseDestinazione!=2 && baseDestinazione!=8 && baseDestinazione!=16)**", che limita ulteriormente il numero di input validi.

2 public double[] calcolaSoluzioniEquazioneSecondoGrado(double a, double b, double c).

soluzioniEquazioneSecondoGradoConAZeroRestituisconoNull()

Verifica che il metodo calcolaSoluzioniEquazioneSecondoGrado() restituisca null quando il coefficiente '**a**' è zero.

Il test è parametrizzato, generando casualmente i coefficienti '**b**' e '**c**' attraverso l'uso di una sottoclasse "**bNonZeroECNonZero**" di "Arbitrary". Si avvale di Property-based testing per generare input che soddisfino questa condizione particolare.

Verifica che il risultato (**soluzioni**) sia nullo **(assertNull(soluzioni)).**

```java
@Property
@Report(Reporting.GENERATED)
@StatisticsReport(format = Histogram.class)
void soluzioniEquazioneSecondoGradoConAZeroRestituisconoNull(
        @ForAll("bNonZeroECNonZero") double b,
        @ForAll("bNonZeroECNonZero") double c
) {

    OperazioniMath operazioniMath = new OperazioniMath();
    double[] soluzioni = operazioniMath.calcolaSoluzioniEquazioneSecondoGrado( a: 0, b, c);
    assertNull(soluzioni);

    Statistics.collect(soluzioni == null ? "Calcolo delle soluzioni restituito null" :
            "Calcolo delle soluzioni non restituito null");
}

no usages  MarcoPrr00
@Provide
Arbitrary<Double> bNonZeroECNonZero() { return Arbitraries.doubles().filter(d -> d != 0); }
```

✓ soluzioniEquazioneSecondoGradoConAZeroRestituisconoNull 131 ms

Statistica:

Il test utilizza una statistica di tipo istogramma (**Histogram**). La statistica tiene traccia del numero di volte che il calcolo delle soluzioni ha restituito il valore **nullo**. Le categorie della statistica sono "**Calcolo delle soluzioni restituito null**" e "**Calcolo delle soluzioni non restituito null**".

```
timestamp = 2024-02-05T18:23:35.857318800, [PropertyProvaTest:soluzioniEquazioneSecondoGradoConAZeroRestituisconoNull] (1000) statistics =
      # |                                 label | count |
    ----|---------------------------------------|-------|-----------------------------------------------------------------------------------
      0 | Calcolo delle soluzioni restituito null |  1000 | ###################################################################################
```

La statistica fornisce un'indicazione sulla frequenza con cui il metodo gestisce correttamente i casi in cui il coefficiente '**a**' è zero. Se il test ha successo, sarà incrementata la statistica "**Calcolo delle soluzioni restituito null**"; in caso contrario, sarà incrementata la statistica "**Calcolo delle soluzioni non restituito null**".

La statistica riportata mostra che su un totale di **1000** esecuzioni del test, in tutte le occasioni il test ha avuto successo.

soluzioniEquazioneSecondoGradoCoefficientiOltreLeSoglie()

Questo caso di test assicura che venga lanciata **un'eccezione** quando i coefficienti superano le soglie minime o massime consentite.

Il test è parametrizzato, generando casualmente i coefficienti '**a**', '**b**' e '**c**'.

Utilizza il Property-based testing per esplorare input con coefficienti al di fuori delle soglie.

Verifica che, se '**a**' o '**b**' o '**c**' superano le soglie minime o massime consentite, il metodo lancia un'eccezione di tipo **ArithmeticException** attraverso **assertThrows.**

```
@Property
@Report(Reporting.GENERATED)
@StatisticsReport(format = Histogram.class)
void soluzioniEquazioneSecondoGradoCoefficientiOltreLeSoglie(@ForAll double a, @ForAll double b, @ForAll double c) {
    double sogliaMax = 1000000000;
    double sogliaMin = -1000000000;
    Assume.that( condition: a>sogliaMax || a<sogliaMin || b>sogliaMax || b<sogliaMin || c>sogliaMax || c<sogliaMin);
    Assume.that( condition: a != 0);

    assertThrows(ArithmeticException.class,() ->{
        operazioniMath.calcolaSoluzioniEquazioneSecondoGrado(a,b,c);
    });

    Statistics.collect(a>sogliaMax ?  "A Molto Grande" : b>sogliaMax ?  "B Molto Grande" :c>sogliaMax ?  "C Molto Grande"
            :a<sogliaMin-1 ?  "A Molto Piccolo" :b<sogliaMin ?  "B Molto Piccolo" :c<sogliaMin ?
            "C Molto Piccolo" :"Coefficienti con buoni valori");
}
```

✓ soluzioniEquazioneSecondoGradoCoefficientiOltreLeSoglie 223 ms

Statistica:

Il test utilizza una statistica di tipo istogramma (**Histogram**). La statistica tiene traccia del numero di volte che l'eccezione è stata lanciata con coefficienti oltre le soglie. Le categorie della statistica sono "**A Molto Grande**", "**B Molto Grande**", "**C Molto Grande**", "**A Molto Piccolo**", "**B Molto Piccolo**", "**C Molto Piccolo**" e "**Coefficienti con buoni valori**".

```
timestamp = 2024-02-05T18:23:35.727446400, [PropertyProvaTest:soluzioniEquazioneSecondoGradoCoefficientiOltreLeSoglie] (835) statistics =
    # |           label | count |
-----|-----------------|-------|-------------------------------------------------------------------------------
    0 |  A Molto Grande |   253 | ##########################################################################
    1 | A Molto Piccolo |   116 | ################################
    2 |  B Molto Grande |   186 | ####################################################
    3 | B Molto Piccolo |    87 | ########################
    4 |  C Molto Grande |   143 | ########################################
    5 | C Molto Piccolo |    50 | ##############
```

La statistica fornisce un'indicazione sulla frequenza con cui il metodo lancia correttamente **un'eccezione** quando i coefficienti superano le soglie. Le categorie specificano quali coefficienti superano le soglie: "**A Molto Grande**", "**B Molto Grande**", "**C Molto Grande**" per valori **superiori** alle **soglie massime**; "**A Molto Piccolo**", **"B Molto Piccolo**", "**C Molto Piccolo**" per valori **inferiori** alle **soglie minime**. La categoria "**Coefficienti con buoni valori**" indica quando tutti i coefficienti sono **all'interno** delle **soglie**.

La statistica riporta il numero di volte che ciascun coefficiente supera i limiti di soglia indicando la distribuzione delle condizioni e offrendo una loro panoramica. La statistica mostra che su un totale di **1000** esecuzioni del test, **835** hanno soddisfatto le condizioni, in quanto viene applicata la restrizione "**Assume.that(a!=0 && a>sogliaMax || a<sogliaMin || b>sogliaMax || b<sogliaMin || c>sogliaMax || c<sogliaMin)**", che limita ulteriormente il numero di input validi.

soluzioniEquazioneSecondoGradoDeltaNegativoRestituisconoNaN()

Questo caso di test verifica che le soluzioni siano **NaN** quando il delta è negativo. Il test è parametrizzato, generando casualmente i coefficienti '**a**', '**b**' e '**c**'. Applica il Property-based testing per testare questo comportamento in presenza di delta negativo. Verifica che, se il delta è negativo (**b * b - 4 * a * c < 0**), il metodo restituisca un **array** di **soluzioni** contenente **NaN** per entrambe le soluzioni. Utilizza **assertArrayEquals** per confrontare l'array di soluzioni con l'array atteso [**Double.NaN, Double.NaN**].

```
MarcoPrr00 +1 *
@Property
@Report(Reporting.GENERATED)
@StatisticsReport(format = Histogram.class)
void soluzioniEquazioneSecondoGradoDeltaNegativoRestituisconoNaN(
        @ForAll @DoubleRange(min = -1000000000, max = 1000000000) double a,
        @ForAll @DoubleRange(min = -1000000000, max = 1000000000) double b,
        @ForAll @DoubleRange(min = -1000000000, max = 1000000000) double c
) {
    Assume.that( condition: a != 0);
    Assume.that( condition: b * b - 4 * a * c < 0);
    OperazioniMath operazioniMath = new OperazioniMath();
    double[] soluzioni = operazioniMath.calcolaSoluzioniEquazioneSecondoGrado(a, b, c);
    assertArrayEquals(new double[]{Double.NaN, Double.NaN}, soluzioni);
    Statistics.collect(Double.isNaN(soluzioni[0]) && Double.isNaN(soluzioni[1]) ?
            "Soluzioni NaN con delta negativo" :
            "Soluzioni reali con delta negativo"
            );
}
```

✓ soluzioniEquazioneSecondoGradoDeltaNegativoRestituisconoNaN 105 ms

Statistica:

Il test utilizza una statistica di tipo istogramma (**Histogram**). La statistica tiene traccia del numero di volte che il test ha avuto successo quando il delta è negativo. Le categorie della statistica sono "**Soluzioni NaN con delta negativo**" e "**Soluzioni reali con delta negativo**".

```
timestamp = 2024-02-05T18:23:35.618357900, [PropertyProvaTest:soluzioniEquazioneSecondoGradoDeltaNegativoRestituisconoNaN] (248) statistics =
      # |                            label | count |
    ----|----------------------------------|-------|----------------------------------------------------------------------------------------
      0 | Soluzioni NaN con delta negativo |   248 | ########################################################################################
```

La statistica fornisce un'indicazione sulla frequenza con cui il metodo restituisce correttamente soluzioni **NaN** quando il delta è negativo. La categoria "**Soluzioni NaN con delta negativo**" indica che il test ha avuto successo quando entrambe le **soluzioni** sono **NaN**. La categoria "**Soluzioni reali con delta negativo**" indica che il test ha avuto successo quando il metodo ha restituito **soluzioni reali** invece di NaN.

La statistica riportata mostra che su un totale di 1000 esecuzioni del test, **248** hanno soddisfatto le condizioni, in quanto viene applicata la restrizione “**Assume.that(b * b - 4 * a * c < 0)**”, che limita ulteriormente il numero di input validi.

soluzioniEquazioneSecondoGradoDeltaPositivo()

Questo caso di test assicura che le soluzioni siano **diverse** quando il delta è positivo, in modo da esser certi di aver avuto due soluzioni distinte. Il test è parametrizzato, generando casualmente i coefficienti '**a**', '**b**' e '**c**'.
Si avvale del Property-based testing per generare input con delta positivo. Verifica che, se il delta è positivo (**b * b - 4 * a * c > 0**), il metodo restituisca un **array** di **soluzioni** con due **valori distinti**.
Utilizza **assertThat** per verificare che l'array di **soluzioni** non contenga duplicati (**doesNotHaveDuplicates**).

```
@Property
@Report(Reporting.GENERATED)
@StatisticsReport(format = Histogram.class)
void soluzioniEquazioneSecondoGradoDeltaPositivo( @ForAll @DoubleRange(min = -1000000000, max = 1000000000) double a,
                                                  @ForAll @DoubleRange(min = -1000000000, max = 1000000000) double b,
                                                  @ForAll @DoubleRange(min = -1000000000, max = 1000000000) double c) {
    Assume.that( condition: a != 0);
    Assume.that( condition: b * b - 4 * a * c > 0);
    double[] soluzioni = operazioniMath.calcolaSoluzioniEquazioneSecondoGrado(a, b, c);
    assertThat(soluzioni).doesNotHaveDuplicates();
    Statistics.collect(
            soluzioni.length == 2 ? "Test con delta positivo eseguito correttamente" :
                    "Test con delta positivo fallito"
    );
}
```

✓ soluzioniEquazioneSecondoGradoDeltaPositivo 686 ms

Statistica:

Il test utilizza una statistica di tipo istogramma (**Histogram**). La statistica tiene traccia del numero di volte che il test ha avuto successo o fallito in base al delta. Le categorie della statistica sono "**Test con delta positivo eseguito correttamente**" e "**Test con delta positivo fallito**".

```
timestamp = 2024-02-05T18:23:35.215804700, [PropertyProvaTest:soluzioniEquazioneSecondoGradoDeltaPositivo] (696) statistics =
      # |                                          label | count |
   -----|------------------------------------------------|-------|-----------------------------------------------------------------------------------------
      0 | Test con delta positivo eseguito correttamente |   696 | ########################################################################################
```

La statistica fornisce un'indicazione sulla frequenza con cui il metodo restituisce correttamente due **soluzioni distinte** quando il delta è positivo. La categoria "**Test con delta positivo eseguito correttamente**" indica che il test ha successo quando il metodo restituisce due **soluzioni distinte**. La categoria "**Test con delta positivo fallito**" indica che il test fallisce quando il metodo **non** restituisce due **soluzioni distinte**.

La statistica riportata mostra che su un totale di 1000 esecuzioni del test, 696 hanno soddisfatto le condizioni, in quanto viene applicata la restrizione “**Assume.that(b * b - 4 * a * c > 0)**”, che limita ulteriormente il numero di input validi.

soluzioniEquazioneSecondoGradoDeltaUgualeAZero()

Questo caso di test verifica che le soluzioni siano **uguali** quando il delta è uguale a zero. Il test è parametrizzato, generando casualmente i coefficienti '**a**', '**b**' e '**c**'. Utilizza il metodo **isValidInput()** per verificare se il delta è uguale a zero. Se il delta è diverso da zero, il test viene interrotto.
Utilizza il Property-based testing per testare il calcolo delle soluzioni in presenza di delta uguale a zero.
Verifica che, se il delta è uguale a zero (**b * b - 4 * a * c == 0**) il metodo restituisca un array di soluzioni contenente due **valori uguali**. Utilizza **assertThat** per verificare che le soluzioni sono uguali(**soluzioni[0]).isEqualTo(soluzioni[1]**).

```java
@Property
@Report(Reporting.GENERATED)
@StatisticsReport(format = Histogram.class)
void soluzioniEquazioneSecondoGradoDeltaUgualeAZero(
        @ForAll @DoubleRange(min = -1000000000, max = 1000000000) double a,
        @ForAll @DoubleRange(min = -1000000000, max = 1000000000) double b,
        @ForAll @DoubleRange(min = -1000000000, max = 1000000000) double c
) {
    Assume.that( condition: a != 0);

    double[] soluzioni = new double[0];
    if (isValidInput(a, b, c)) {
        soluzioni = operazioniMath.calcolaSoluzioniEquazioneSecondoGrado(a, b, c);
        assertThat(soluzioni[0]).isEqualTo(soluzioni[1]);
    }

    Statistics.collect(soluzioni.length == 2 && soluzioni[0] == soluzioni[1] ?
            "Test con delta uguale a zero eseguito correttamente" :
            "Test con delta uguale a zero fallito"
    );
}

1 usage  MarcoPrr00
private boolean isValidInput(double a, double b, double c) {
    double delta = b * b - 4 * a * c;
    return delta == 0;
}
```

✓ soluzioniEquazioneSecondoGradoDeltaUgualeAZero 207 ms

Statistica:

Il test utilizza una statistica di tipo istogramma (**Histogram**). La statistica tiene traccia del numero di volte che il test ha avuto successo quando il delta è uguale a zero. Le categorie della statistica sono "**Test con delta uguale a zero eseguito correttamente**" e "**Test con delta uguale a zero fallito**".

```
timestamp = 2024-02-05T18:23:35.353238200, [PropertyProvaTest:soluzioniEquazioneSecondoGradoDeltaUgualeAZero] (943) statistics =
      # |                                                label | count |
    ----|------------------------------------------------------|-------|-----------------------------------------------------------------------------------------
      0 | Test con delta uguale a zero eseguito correttamente |     3 |
      1 |               Test con delta uguale a zero fallito |   940 | ########################################################################################
```

La statistica fornisce un'indicazione sulla frequenza con cui il metodo restituisce correttamente due **soluzioni uguali** quando il delta è uguale a zero. La categoria "**Test con delta uguale a zero eseguito correttamente**" indica che il test ha successo quando il metodo restituisce due **soluzioni uguali** in caso di delta uguale a zero. La categoria "**Test con delta uguale a zero fallito**" indica che il test fallisce quando il metodo restituisce **soluzioni diverse** in caso di delta uguale a zero.

La statistica riportata mostra che su un totale di **943 esecuzioni** del test, solo in **3 casi** il test ha avuto **successo**. Questo significa che su 943 input generati casualmente, solo in 3 casi il delta dell'equazione è stato uguale a zero e le soluzioni calcolate sono risultate uguali. Questo perché la **probabilità** che il **delta** sia esattamente **zero** su un intervallo così ampio di numeri casuali è **estremamente bassa** e anche se il delta fosse zero, il calcolo potrebbe essere influenzato da errori di arrotondamento o approssimazioni numeriche, portando a risultati leggermente diversi.

Considerazioni Finali

Attraverso l'implementazione dei test **Property Based** per la classe "**OperazioniMath**", abbiamo potuto verificare la correttezza della conversione di numeri decimali in diverse basi (binaria, ottale ed esadecimale) e la correttezza del calcolo delle soluzioni di un'equazione di secondo grado.

L'utilizzo del **PBT**, insieme a generazione casuale dei dati di input, ha consentito di **coprire** una vasta **gamma di casi**, inclusi **casi limite**.

Inoltre, l'utilizzo di **statistiche** all'interno dei test ci ha consentito di valutare la robustezza del nostro programma rispetto a varie situazioni e input.