Esercizio A

La classe *StreamingService* ha un metodo *calculateSubscription* che viene utilizzato per calcolare il prezzo mensile da offrire ad un cliente per accedere a un servizio di streaming. Il metodo prende in input i seguenti parametri:

- `Int maxDevice:` indica il numero massimo di dispositivi che possono essere utilizzati contemporaneamente.
- `String quality`: indica la qualità dello streaming richiesta dal cliente ("SD","HD","UHD").
- `Boolean familyPlan:` indica se il pacchetto include un piano famiglia, che permette di creare profili separati per ogni membro della famiglia.

Se i parametri non sono validi, il metodo solleva un IllegalArgumentException. In caso contrario, ritorna il prezzo mensile da proporre al cliente. Il prezzo mensile di base è calcolato moltiplicando il numero di dispositivi per il costo mensile per dispositivo determinato in base alla qualità dello streaming richiesta, come da tabella seguente. Inoltre, se è incluso il piano famiglia, la somma di 10.00 € va aggiunta al totale.

Qualità dello streaming	**Costo per dispositivo (€)**
SD	5
HD	10
UHD	15

- Indicare, per ciascuno dei parametri del metodo *calculateSubscription*, le classi di equivalenza individuate.
- Scrivere quattro test JUnit con strategia Black Box per il metodo *calculateSubscription*, indicando per ciascuno di essi quali classi di equivalenza copre. Si richiede inoltre che un test corrisponda a scenari in cui i parametri non sono validi, e che i restanti tre corrispondano a scenari in cui i parametri sono validi.
- Quanti test sono necessari per testare il metodo con strategia R-WECT? Motivare la risposta.

Esercizio B

Un numero naturale positivo si dice strobogrammatico se è simmetrico rispetto a una rotazione di 180°. Per esempio, i numeri `181`, `1961`, e `160091` sono strobogrammatici perché restano "identici" quando capovolti dopo una rotazione di 180°. Il metodo `isStrobogrammatic` della classe `Utils`, la cui implementazione è riportata di seguito, viene utilizzato per calcolare se un numero intero è strobogrammatico.

```
   public static boolean isStrobogrammatic(int n) {
1     if(n<0)
2         throw new IllegalArgumentException("Non ammissibili interi negativi");
3     String original = Integer.toString(n);
4     for(String c : Arrays.asList("2", "3", "4", "5", "7"))
5        if(original.contains(c))
6             return false; //non può essere stenogrammatico se contiene 2, 3, 4, 5, o 7
7     StringBuilder sb = new StringBuilder(original);
8     String reversed = sb.reverse().toString();
9     reversed = reversed.replace("6","X");
10    reversed = reversed.replace("9","Y");
11    reversed = reversed.replace("X","9");
12    reversed = reversed.replace("Y","6");
12    if(original.equals(reversed))
13        return true;
14    else
15        return false;
   }
```

i. Rappresentare il CFG del metodo ***`isStrobogrammatic`***;

ii. Scrivere **tre** test JUnit con strategia White Box per il metodo ***`isStrobogrammatic`***, indicando per ciascuno di essi quale cammino copre nel CFG. Si richiede che almeno un test copra uno scenario di errore (input non valido) e che, ove possibile, i test JUnit coprano cammini distinti nel CFG;

iii. Qual è la Test Effectiveness Ratio (TER), relativamente alla copertura di nodi del CFG, della suite di tre test sviluppata al punto (ii)? Motivare la risposta.

Esercizio C

Il metodo `generateRandomPets(int n)` della classe `GameUtils` ritorna una `List<Pet>` contenente `n` oggetti di tipo `Pet` (il codice della classe `Pet` è riportato di seguito).

```
public class Pet {
    private String name;
    private int age;
    private String gender;
    /* Costruttori, getter e setter omessi per brevità */
}
```

Sono ammissibili soltanto valori di `n` compresi tra 2 e 100, estremi inclusi.

1. Definire un partizionamento in classi di equivalenza per il parametro n del metodo `generateRandomPet`.
2. Quanti test sono necessari a testare il metodo con strategia SECT? Quanti con strategia R-WECT?
3. Si scriva un test JUnit distinto per verificare ciascuno dei seguenti requisiti aggiuntivi:
   a. Gli animali domestici della lista non devono essere avere tutti lo stesso genere;
   b. Tutti gli animali domestici della lista devono avere un'età compresa tra 1 e 7 anni.
   c. Almeno un animale domestico della lista deve avere un'età pari a 7 anni.
   d. Se il metodo viene invocato con parametro non valido, viene lanciata una IllegalArgumentException.

Esercizio D

In un videogioco di ruolo a turni, il metodo statico `dodgeProbability` della classe `MissileAttacksUtils` viene utilizzato per calcolare la probabilità che il personaggio controllato dal giocatore riesca a schivare un proiettile magico con determinate caratteristiche. Il metodo prende in input i seguenti parametri:

- `double speed`: indica la velocità del proiettile, e può assumere valori nell'intervallo]0,100[;
- `boolean isTargetTracking`: indica se il proiettile possiede la proprietà di guida autonoma verso il bersaglio;
- `int dex`: indica il livello della statistica "dexterity" del giocatore, e può assumere valori interi non negativi;

Se i parametri non sono validi, il metodo solleva una `IllegalArgumentException`. In caso contrario, ritorna la probabilità del giocatore di schivare l'attacco. Tale proprietà è calcolata in maniera pseudo-casuale all'interno di intervalli determinati come mostrato nella tabella seguente. Gli intervalli sono da considerarsi con estremi inclusi.

	$0 < speed \leq 30$		**$30 < speed \leq 80$**		**$80 < speed < 100$**	
	Tracking	***Non Tracking***	***Tracking***	***Non Tracking***	***Tracking***	***Non Tracking***
$dex < 50$	0.2 – 0.5	0.5 – 0.8	0.3 – 0.4	0.4 – 0.5	0	0
$50 \leq dex < 80$	0.5 – 0.8	1	0.4 – 0.6	0.5 – 0.7	0	0.2 – 0.5
$dex \geq 80$	1	1	0.7 – 0.8	0.8 – 1	0.2 – 0.5	0.3 – 0.6

Per esempio, qualsiasi probabilità di schivata compresa tra 0.2 e 0.5 sarebbe valida per un proiettile con velocità di 99, con la proprietà TargetTracking attiva, lanciato contro un giocatore con livello di dexterity di 81. Allo stesso modo, un proiettile con velocità compresa tra 80 e 100 (estremi esclusi) non può mai essere schivato da un giocatore con dexterity inferiore a 50 (probabilità di schivata è fissa a 0).

a) Indicare, per ciascuno dei parametri del metodo `dodgeProbability`, le classi di equivalenza individuate.
b) Scrivere quattro test JUnit con strategia Black Box per il metodo `dodgeProbability`, indicando per ciascuno di essi quali classi di equivalenza copre. Si richiede inoltre che un test corrisponda a scenari in cui i parametri non sono validi, e che i restanti tre corrispondano a scenari in cui i parametri sono validi.
c) Quanti test sono necessari per testare il metodo con strategia N-WECT? Motivare la risposta.