Esercizio A

In un videogioco di ruolo, il metodo statico `computePrice` della classe `ShopUtils` viene utilizzato per calcolare il prezzo (espresso in Septim, la valuta di gioco) di un item in vendita in un negozio gestito da un NPC. Il metodo prende in input i seguenti parametri:

- `int itemLevel`: indica il livello dell'oggetto e può assumere valori compresi tra 1 e 99;
- `String rarity`: indica il livello di rarità dell'item ("RARE", "EPIC", oppure "LEGENDARY");
- `boolean isPlayerMerchant`: indica se è il giocatore ha correntemente attivo il tratto "Merchant", che gli permette di avere prezzi più vantaggiosi quando acquista da NPC.

Se i parametri non sono validi, il metodo solleva una `IllegalArgumentException`. In caso contrario, ritorna il prezzo da mostrare al giocatore. Il prezzo è determinato in maniera pseudocasuale all'interno di intervalli determinati come mostrato nella tabella seguente. Gli intervalli sono da considerarsi con estremi inclusi.

	Rare		**Epic**		**Legendary**	
	Merchant	**Non Merchant**	**Merchant**	**Non Merchant**	**Merchant**	**Non Merchant**
Level < 50	5-10	10-15	100-150	200-250	500-700	800-1000
50 <= Level < 80	100-150	200-250	500-700	800-1000	2500-5000	3000-7000
80 <= Level < 100	500-700	800-1000	2500-5000	3000-7000	10000-20000	15000-30000

Per esempio, qualsiasi valore compreso tra 2500 e 5000 Septim sarebbe valido per un item Legendary di livello 51 da vendere a un giocatore con il tratto Merchant attivo.

a) Indicare, per ciascuno dei parametri del metodo `getQuote`, le classi di equivalenza individuate.
b) Scrivere quattro test JUnit con strategia Black Box per il metodo `computePrice`, 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.

Esercizio B

```java
public boolean isPalindromo(String str){
    boolean palindromo = true; // indica se la stringa è palindroma
    int idxFw = 0; // indice per scorrere in avanti
    int idxBw = str.length()-1; // indice per scorrere indietro
    int estremo = str.length()/2; // numero confronti
    while(idxFw<estremo && palindromo){
        palindromo = str.charAt(idxFw) == str.charAt(idxBw);
        idxFw++;
        idxBw--;
    }
    if(palindromo){
        System.out.println("La stringa è palindroma");
    }else{
        System.out.println("La stringa non è palindroma");
    }
    return palindromo;
}
```

Una stringa palindroma è una sequenza di caratteri che rimane la stessa se letta da sinistra a destra e da destra a sinistra, un esempio “radar”, “level”. Il metodo `isPalindromo` la cui implementazione è riportata di seguito, viene utilizzato per calcolare se una stringa è palindroma.

i. Rappresentare il CFG del metodo `isPalindromo`;

ii. Scrivere **tre** test JUnit con strategia White Box per il metodo `isPalindromo`, 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.