Incontro 3

Parte 1

Si vuole realizzare un software per gestire una sistema di carte collezionabili. All'interno della cartella `to_do` troverete i file da cui iniziare.

1. La classe `Card` rappresenta una carta da collezione con i seguenti attributi:

   - `code (int)`: il codice univoco che rappresenta la carta
   - `name (str)`: il nome della carta

   Vi viene chiesto di completare il metodo `__str__` ritornando una stringa del tipo `NAME - CODE`

2. La classe `User` rappresenta un singolo utente con la relativa collezione di carte, rappresentato dai seguenti attributi:

   - `name (str)`: il nome dell'utente
   - `deck (set[Card])`: la collezione di carte
   - `duplicates (set[Card])`: i doppioni dell'utente
   - `dup_counter (Dict[int, int])`: dizionario per tenere conto del numero dei doppioni (qual è la chiave e qual è il valore?)

   La classe inoltre richiede di completare i seguenti metodi:

   - `add_card(self, card: Card)`: l'utente riceve una carta. Se **non** possiede già la carta, allora viene aggiunta al `deck` e impostato il contatore dei doppioni (`dup_counter`) a 0. Altrimenti, viene aggiunta ai `duplicates` e incrementato il rispettivo contatore
   - `is_collection_complete(self) -> bool`: ritorna `True` se la collezione contiene tutte e **10** le carte da collezione, `False` altrimenti
   - `discard_card(self, card: Card)`: l'utente deve scartare **una volta** la carta passata come parametro. Ad esempio, supponiamo che venga richiesto di scartare la carta `Tigre`:
     - Se l'utente ne possiede 3 (una nel deck e due doppioni): allora andrà scartato un doppione, diminuendo il contatore
     - Se l'utente ne possiede 2 (una nel deck e un doppione): allora va rimossa dai doppioni e il contatore decrementato
     - Se l'utente ne possiede una: allora va rimossa dal deck
   - `count_all_duplicates(self) -> int`: ritorna il numero di tutti i doppioni dell'utente (**Hint**: non è la lunghezza di `duplicates`)

   Test: vengono forniti dei test di collaudo sotto la classe User (**N.B.** superare tutti i test non garantisce l'assenza di errori)

Parte 2: main.py

Per aiutare gli utenti a completare la collezione, il sistema vuole permettere agli utenti di effettuare scambi.

- `get_interesting_cards(user1, user2) -> Set[Card]`: dati due utenti, user1 e user2, si vogliono trovare tutte le carte di user2 che interessano a user1. Una carta di user2 si dice interessante per user1 se:
  - è un doppione per user2 (altrimenti vorrà tenerla nella sua collezione)
  - manca nella collezione di user1

  Hint: ricorda che stiamo usando i set
- `count_possible_trades(user1, user2) -> int`: ritorna il numero di scambi di carte possibili tra i 2 utenti. Supponiamo che user1 sia interessato a 4 carte di user2 e che user2 sia interessato a 3 carte di user1. Gli scambi possibili sono **3**
- `make_random_trade(user1: User, user2: User)`: viene presa **casualmente** una carta che interessa a user1 e una che interessa a user2 e viene effettuato uno scambio

  Hint: utilizza la funzione già implementata `pick_random_card` che dato un set, ritorna una carta a caso
- `trade_all_interesting_duplicates(user1: User, user2: User)`: gli utenti effettuano scambi casuali (come sopra) finché non finiscono gli scambi possibili

main(): viene fornito un main di collaudo per testare il codice (**N.B.** superare tutti i test non garantisce l'assenza di errori)