Sommario

Class MyShelfie

La classe MyShelfie ha due variabili di istanza private, un'ArrayList di giocatori (players) e un'ArrayList di giochi (games).
Il costruttore pubblico MyShelfie() inizializza l'ArrayList dei giocatori (players) con un nuovo ArrayList vuoto.
La classe ha anche un metodo pubblico void joinGame(int intPlayerNumber, Player p) che può essere utilizzato per aggiungere un giocatore ad un gioco specifico. Il primo parametro, intPlayerNumber, rappresenta il numero del giocatore all'interno del gioco e il secondo parametro, p, è l'oggetto Player da aggiungere.
Attualmente il metodo joinGame() è vuoto e non implementa alcuna logica, quindi dovrebbe essere implementato per aggiungere il giocatore p alla lista dei giocatori del gioco corrispondente. Inoltre, potrebbe essere necessario controllare se il gioco esiste già nell'ArrayList dei giochi (games) e, se non esiste, crearlo e aggiungerlo all'ArrayList.

Class Game

La classe Game rappresenta una partita del gioco e contiene un riferimento ad un'istanza della classe Board (plancia), un'ArrayList di oggetti Player (giocatori), un oggetto Player che rappresenta il mazziere (dealer), il numero di giocatori nella partita e lo stato attuale della partita (StatoPartita).
Il costruttore Game(int playerNumber, Player mazziere) crea una nuova istanza della partita con il numero di giocatori specificato (playerNumber) e l'oggetto Player che rappresenta il mazziere. Inoltre, crea un nuovo oggetto ArrayList di Player (players), aggiunge il mazziere all'ArrayList dei giocatori e crea una nuova istanza della plancia (Board).
I metodi pubblici getBoard(), getPlayers(), getPlayerNumber() e getDealer() restituiscono rispettivamente una copia dell'oggetto Board, una copia dell'ArrayList dei giocatori, il numero di giocatori nella partita e l'oggetto Player che rappresenta il mazziere.
Il metodo pubblico void addPlayer(Player p) viene utilizzato per aggiungere un nuovo giocatore all'ArrayList dei giocatori. Se lo stato attuale della partita non è StatoPartita.IN_ATTESA, viene sollevata un'eccezione. Se il numero di giocatori nella partita raggiunge il valore specificato nel costruttore (playerNumber), lo stato della partita viene impostato su StatoPartita.IN_CORSO.
Il metodo pubblico StatoPartita getStato() restituisce lo stato attuale della partita.
Il metodo pubblico void end() viene utilizzato per terminare la partita e calcolare il punteggio di ogni giocatore e il ranking dei giocatori. Il metodo imposta lo stato della partita su StatoPartita.FINITA e calcola il punteggio di ogni giocatore in base alle schede presenti nelle loro plance. Il metodo crea un'ArrayList di oggetti Ranking che contengono un riferimento al giocatore e il suo punteggio. Successivamente, il metodo ordina l'ArrayList di Ranking in base ai punteggi dei giocatori e imposta l'ordine dei giocatori nel ranking.
Il metodo pubblico ArrayList<Ranking> getRanking() restituisce l'ArrayList di Ranking dei giocatori ordinati in base al loro punteggio. Se lo stato attuale della partita non è StatoPartita.FINITA, viene sollevata un'eccezione.

Class Board

La classe "Board" contiene metodi e attributi per gestire la plancia di gioco del gioco da tavolo. La plancia è rappresentata come un insieme di posizioni, dove ogni posizione può contenere una tessera di gioco.

Attributi:
"placement": ArrayList di TilePositionBoard,
rappresenta le posizioni della plancia.
"bag": TileObjBag, rappresenta la borsa di tessere di gioco.
"commonDeck": CommonDeck, rappresenta il mazzo di carte comuni di gioco.

"personalDeck": PersonalDeck, rappresenta il mazzo di carte personali di gioco.

Metodi:

"Board(ArrayList<TilePositionBoard> placement, TileObjBag bag, CommonDeck commonDeck)":
costruttore che inizializza la plancia, la borsa di tessere e il mazzo comune, ricevendo come parametro una lista di posizioni, una borsa di tessere e un mazzo comune.
"Board(Board board)":
costruttore che inizializza la plancia, la borsa di tessere e il mazzo comune, ricevendo come parametro un oggetto di tipo Board.
"Board()": costruttore vuoto che inizializza la plancia, la borsa di tessere e il mazzo comune.

"getPlacement()": metodo che restituisce una copia dell'ArrayList "placement".

"showBoard()": metodo che restituisce una copia dell'ArrayList "placement".

"getBag()": metodo che restituisce una copia dell'oggetto "bag".

"isTileRemovible(int x, int y)":
metodo che controlla se la tessera in una determinata posizione può essere rimossa, ricevendo come parametro le coordinate (x,y) della posizione. Restituisce un booleano.

"tileIsRemovable(TilePositionBoard position)":
metodo che controlla se la tessera in una determinata posizione può essere rimossa, ricevendo come parametro un oggetto di tipo TilePositionBoard. Restituisce un booleano.

"remove1Tile(TilePositionBoard position)":
metodo che rimuove una tessera dalla posizione specificata, ricevendo come parametro un oggetto di tipo TilePositionBoard. Restituisce un oggetto di tipo TileObj.

"removeTile(ArrayList<TilePositionBoard> tile_to_remove)":
metodo che rimuove una o più tessere dalla plancia, ricevendo come parametro un ArrayList di oggetti di tipo TilePositionBoard. Restituisce un ArrayList di oggetti di tipo TileObj. Se una delle posizioni specificate non può essere svuotata, viene lanciata un'eccezione di tipo Exception.

CLASS TileObj
Classe che descrive le singole tessere oggetto:

Ha un attributo type di tipo TileType che è un'enumerazione dei 6 tipi di tessera possibili.
Ha un attributo variant di tipo TileVariant che è un'enumerazione delle tre varianti che ciascun tipo di tessera ha.
Due costruttori. Il primo riceve in ingresso un TileType e un TileVariant e inizializza gli attributi della classe associandogli i parametri passati. Il secondo inizializza la tessera con i valori degli attributi di un'altra tessera passata al costruttore come parametro.
Seguono due metodi getter.

CLASS TileObjBag
Descrive il sacchetto delle tessere oggetto:

Un attributo "tiles" lista di tutte le tessere oggetto contenute nel sacchetto.
Un costruttore che inizializza le 132 tessere oggetto.
Un metodo "shuffleT()" che mischia le tessere.
Un metodo "extractFromBag()" che estrae una tessera dal sacchetto e la restituisce al chiamante.

CLASS TilePositionBoard
Classe che descrive una posizione sulla board di gioco:

Un attributo "x" per l'ascissa,
un attributo "y" per l'ordinata,
un attributo "tileInSlot" per la tessera contenuta nell'oggetto TilePositionBoard,
un attributo "occupied" per indicare se la posizione contiene una tessera oppure è vuota.

Un costruttore che riceve in ingresso solo le coordinate x,y e inizializza la posizione a vuota.
Un costruttore che riceve in ingresso le coordinate x,y e un TileObj (una tessera) e inizializza la posizione a "piena", contenente la tessera passata come parametro.

isOccupied() è il getter dell'attributo "occupied".
getX(), getY(), getTile() rispettivamente sono i metodi getter degli attriburi "x", "y", "tileInSlot".

setTile(TileObj tile) metodo che quando invocato in un dato istante durante la partita associa la tessera tile alla posizione su cui ho chiamato il metodo.

removeTile() metodo che quando invocato su una TilePositionShelves la "svuota" dell'oggetto TileObj

CLASS TilePositionShelves
Classe che descrive una posizione sulla libreria del giocatore:

Un attributo "x" per l'ascissa,
un attributo "y" per l'ordinata,
un attributo "tileInSlot" per la tessera contenuta nell'oggetto TilePositionShelves,
un attributo "occupied" per indicare se la posizione contiene una tessera oppure è vuota.

Un costruttore che riceve in ingresso solo le coordinate x,y e inizializza la posizione a vuota.

Un costruttore che riceve in ingresso le coordinate x,y e un TileObj (una tessera) e inizializza la posizione a “piena”, contenente la tessera passata come parametro.

isOccupied() è il getter dell’attributo “occupied”.
getX(), getY(), getTile() rispettivamente sono i metodi getter degli attriburi “x”, ”y”, “tileInSlot”.

setTile(TileObj tile) metodo che quando invocato in un dato istante durante la partita associa la tessera tile alla posizione su cui ho chiamato il metodo.

“CommonCard gestite tramite un pattern strategy, utilizzati solo 2 oggetti per le carte obbiettivo comuni. -CommonCardShape definisce la forma da rispettare, il numero di volte che si deve presentare. -CommonCardPosition definisce l'algoritmo per identificare le posizioni da rispettare. -CommonCardColumn3Types definisce l'algoritmo per le colonne o righe con max 3 tipi, un numero definito di volte. -CommonCardAllTypes algoritmo per colonne o righe con tutti i tipi. -CommonCardLowDiagonal definisce l'algoritmo per la diagonale inferiore di tessere. \nLe descrizioni di PersonalCard e CommonCard vengono parsate con json, la pescata della carta tramite un randomizing di indici."}],"