Antichi Commerci

Progetto di Programmazione dell'Appello di Gennaio 2017

Introduzione

In questo esame vi chiederemo di scrivere un insieme di classi per la simulazione dell'evoluzione di civilità.

Classi

Descriviamo ora nei dettagli come realizzare le classi necessarie al progetto. Se lo ritenete, potete apportare cambiamenti alle segnature dei metodi proposti e anche – se lo ritenete – alla struttura delle classi, restando ovviamente all'interno dei requisiti esposti nel paragrafo introduttivo.

N.B. (1): Oltre ai metodi qui indicati, potete aggiungere ad ogni classe un valido metodo `toString()` e, dove lo ritenete, anche un metodo `equals` e un metodo `hashCode` (con le segnature appropriate e che soddisfino i contratti previsti).

N.B. (2): Nel seguito, useremo il termine *lista di X* per indicare un qualunque tipo che consenta di conservare delle sequenze di oggetti di tipo `X`: potete implementare una lista con un array (`X[]`) oppure usando ad esempio delle `ArrayList<X>` (pacchetto `java.util`).

Classe Risorsa

Una classe che rappresenta una singola unità di una risorsa presente nel terreno. Contiene i campi `nome` (la stringa rappresentante il nome) e `prezzo` (un numero intero che indica a quanto viene venduta un'unità di questa risorsa); per esempio, "Petrolio" potrebbe avere un prezzo di 100, mentre "Carbone" di 30. Due risorse sono considerate uguali se hanno lo stesso nome.

Classe Citta

Rappresenta una città. Una città è caratterizzata da un nome e da una `Risorsa` che la città è in grado di produrre. Il costruttore ha segnatura `Citta(String nome, Risorsa r)`. Il metodo `Risorsa produci()` fa produrre alla città la risorsa che è in grado di produrre (cioè, restituisce la risorsa prodotta).

Classe Civilta

Rappresenta una civiltà, attraverso il suo `nome` (p.es. "Impero Inglese") e una lista di `Citta`. Avrà un campo intero `tesoro` che indica la quantità di denaro attualmente posseduta da questa civiltà. Avrà anche una lista di `Risorsa` chiamato `stock`, contenente le risorse prodotte nelle città di questa civiltà.

Il costruttore chiederà solo il nome. Una civiltà appena costruita non avrà né città, né risorse, né tesoro.

Deve contenere i seguenti metodi:

- un metodo `Citta fondaCitta(String nome, Risorsa risorsa)` che crea (aggiunge alla lista delle città e restituisce) una città con nome `nome` e che è in grado di produrre la risorsa `risorsa`
- un metodo `void aggiungiRisorsa(Risorsa r)` che aggiunge `r` allo `stock`
- un metodo `void aggiungiDenaro(int d)` che aggiunge la quantità `d` di denaro (che può anche essere negativa) al `tesoro`
- un metodo `void faiProdurre()` che richiama il metodo `produci` di ciascuna città, e aggiunge allo `stock` tutte le risorse prodotte
- un metodo `boolean possiede(Risorsa r)` che determina se la risorsa è posseduta (cioè, è presente nello `stock`)
- un metodo `boolean vendiRisorseA(Civilta altra)`, che consente alla civiltà che lo richiama di vendere risorse alla civiltà specificata come argomento in questo modo (è possibile utilizzare una lista aggiuntiva di supporto, se si ritiene necessario):
    - scorre tutte le risorse dello `stock`
    - per ogni risorsa `r`, controlla se la civiltà `altra` possiede la risorsa `r`:
        - se non la possiede, la civilità vende ad `altra` la risorsa: cioè, la aggiunge allo `stock` di `altra`, e sposta denaro pari al prezzo di `r` dalla civiltà `altra` a questa; il denaro di una civiltà può anche andare in negativo
        - se invece la possiede, la risorsa `r` viene tenuta nello `stock`
    - il metodo restituisce `true` se e solo se vi è stato commercio tra le due civiltà (cioè, se almeno una risorsa è stata venduta).

Classe Storia

Ha un costruttore che accetta una lista di `Civilta`, che sarà un campo della classe.

- Ha un metodo `void commercia()` che fa quanto segue:
    - per civiltà `c`
        - per ogni civiltà `k`, se `c` è diverso da `k`, chiama il metodo `vendiRisorseA(k)`: se

vi è effettivamente un commercio da `c` a `k`, `c` non venderà risorse a nessun altro; altrimenti, proseguirà a considerare la prossima civiltà `k`.

- Ha un metodo `Civilta piuRicca()` che restituisce la civiltà con più denaro.
- Ha un metodo `Civilta evolvi(int n)` che, per `n` volte, fa quanto segue:
  - chiama il metodo `faiProdurre` di ogni civiltà
  - chiama il metodo `commercia`
  - alla fine restituisce la cività più ricca.
- Ha un metodo `void conquista()` che
  - determina la civiltà più ricca, diciamo `cr`
  - determina la civiltà più povera, diciamo `cp`
  - se `cr` e `cp` hanno lo tesso tesoro, il metodo non ha effetto
  - altrimenti, cerca fra le città di `cp` quella con la risorsa meno costosa, e sposta quella città dalla lista delle città di `cp` alla lista delle città di `cr`.