Dispensa

Progetto di Programmazione del 19 Febbraio 2018

Introduzione

In questo esame vi chiederemo di scrivere un insieme di classi per la gestione di un sistema di biciclette a noleggio.

Il progetto verrà valutato prima di tutto in base al suo corretto funzionamento rispetto ai requisiti qui descritti; suggeriamo di commentare con cura il codice, in particolare anche scrivendo i commenti `javadoc`.

Realizzazione

Classe Bicicletta

La classe `Bicicletta` rappresenta delle biciclette. Ogni bicicletta è identificata da un identificatore (che è diverso da bicicletta a bicicletta) e può essere o meno elettrica. In ogni istante la bicicletta può essere libera o in uso a un utente.

La classe ha solo un costruttore

- `public Bicicletta(String id, boolean elettrica)`: crea una bicicletta con identificatore dato (non dovete controllarlo, ma ogni volta che viene creata una bicicletta chi la crea garantisce di usare un identificatore nuovo); la bicicletta è elettrica solo se il booleano è true. La bicicletta è inizialmente libera.

Inoltre sono disponibili i metodi:

- `public boolean elettrica()`: dice se la bicicletta è elettrica.
- `public void daiInUso(Utente x)`: dà in uso la bicicletta all'utente indicato; solleva una `IllegalStateException` se la bicicletta è in uso a un altro utente.
- `public void restituisci()`: l'utente che aveva in uso la bicicletta la restituisce; solleva una `IllegalStateException` se la bicicletta non è in uso a nessun utente.
- `public Utente utente()`: se attualmente questa bicicletta è in uso a un utente, restituisce l'utente che ce l'ha in uso; altrimenti restituisce `null`.

La classe inoltre sovrascrive i metodi `toString`, `equals` e `hashCode` di `Object`.

Classe Utente

La classe `Utente` rappresenta un utente. Ogni utente ha un nome, un cognome e in ogni istante può avere in prestito al massimo una bicicletta. La classe ha un solo costruttore

- `public Utente(String nome, String cognome)`.

Inoltre sono disponibili i seguenti metodi:

- `public Bicicletta bicicletta()` : se attualmente questo utente ha in uso una bicicletta, il metodo la restituisce; altrimenti restituisce `null`.
- `public void prendiInUso(Bicicletta x)` : prende in uso la bicicletta indicata; solleva una `IllegalStateException` se l'utente ha già in uso un'altra bicicletta.
- `public void restituisci()` : restituisce la bicicletta attualmente in uso; solleva una `IllegalStateException` se l'utente non ha biciclette in uso.

La classe inoltre sovrascrive il metodo `toString`.

Classe Stazione

La classe `Stazione` rappresenta una stazione di deposito biciclette. Ogni stazione ha un nome, una certa localizzazione geografica (immaginiamo che le stazioni siano disposte sul piano cartesiano) e un certo numero di posizioni disponibili. Può essere o meno abilitata a ospitare biciclette elettriche.

- `public Stazione(String nome, double x, double y, int posizioni, boolean elettriche)`: crea una stazione con nome dato, localizzata nel punto (x,y), e avente un certo numero di posizioni (inizialmente tutte vuote); il booleano indica se la stazione può ospitare biciclette elettriche.

Sono disponibili i seguenti metodi:

- `public int nPosizioni()` : restituisce il numero di posizioni.
- `public int nBiciclette()` : restituisce il numero di posizioni occupate.
- `public boolean piena()` : dice se questa stazione è attualmente piena.
- `public boolean vuota()` : dice se questa stazione è attualmente vuota.
- `public boolean ammetteElettriche()` : dice se questa stazione è in grado di ospitare biciclette elettriche.
- `public boolean ciSonoElettriche()` : dice se questa stazione contiene al momento almeno una bicicletta elettrica.
- `public Bicicletta posizione(int i)` : restituisce la bicicletta attualmente agganciata alla posizione `i` (che deve essere un numero compreso fra 0 e `nPosizioni()-1`. Restituisce `null` se la posizione è vuota.
- `public void aggancia(Utente u, int i)` : l'utente `u` aggancia la bicicletta che ha in uso la bicicletta in posizione `i`. Solleva una `IllegalStateException` se la posizione `i` è già occupata o se l'utente non ha in uso biciclette. Si occupa di registrare sia nella bicicletta sia nell'utente coinvolti il fatto che ora l'utente non ha più biciclette in uso e che la bicicletta è libera.
- `public void prendi(Utente u, int i)` : l'utente `u` sgancia e prende la bicicletta in posizione `i`. Solleva una `IllegalStateException` se la posizione `i` è vuota o se l'utente ha già in uso una bicicletta. Si occupa di registrare sia nella bicicletta sia nell'utente coinvolti il fatto che ora l'utente ha quella bicicletta in uso e che quella bicicletta è ora in uso a quell'utente.
- `public int disponibile(boolean elettrica)` : restituisce la prima posizione che contiene una bicicletta (se `elettica` è true considera solo biciclette elettriche; altrimenti considera tutte le biciclette).

Restituisce -1 se la stazione non contiene biciclette di quel tipo.
- `public double distanzaDa(double x, double y)` : restituisce la distanza fra questa stazione e il punto di coordinate (x,y) (si tratta della distanza fra due punti nel piano cartesiano).

La classe inoltre sovrascrive il metodo `toString` .

Aggiungete inoltre alla classe il metodo statico:

- `public static Stazione cerca(Stazione[] s, boolean elettrica, double x, double y)` : cerca e restituisce, fra le stazioni dell'array `s` , quella più vicina al punto (x,y) e avente disponibilità di biciclette (se `elettica` è true considera solo biciclette elettriche; altrimenti considera tutte le biciclette). Restituisce `null` se non c'è alcuna stazione che soddisfa il criterio.