Esame di Programmazione II, 7 febbraio 2018

Si consideri un'implementazione di un sistema di prenotazione di stanze *HairBnB*. Le stanze possono essere aggiunte al sistema di prenotazione, specificando il numero massimo di ospiti e il prezzo richiesto al giorno. I clienti possono scrivere delle recensioni (*review*) per le stanze, indicando una valutazione da una a cinque stelle. Il sistema di prenotazione permette di selezionare tutte le stanze che soddisfano dei requisiti di ricerca (numero ospiti e prezzo), eventualmente ordinate rispetto a un qualche criterio di ordinamento.

Nota: in questo compito non si ridefiniscano i metodi `equals()` e `hashCode()`, anche se si aggiungessero oggetti in `HashSet`: vanno bene quelli ereditati da `Object`.

Esercizio 1 [5 punti] Si completi la seguente classe, che rappresenta una recensione di un utente:

```
public class Review {
  public enum Stars { ONE, TWO, THREE, FOUR, FIVE }
  ...

  // costruisce una recensione con autore, testo e valutazione da 1 a 5 stelle
  public Review(String author, String text, Stars stars) { ... }

  // ritorna una stringa di asterischi, lunga da 1 a 5, sulla base delle stelle della recensione
  private String stars() { ... }

  // ritorna l'autore seguito dal risultato di stars() seguito dal testo della recensione
  @Override public String toString() { ... }

  // ritorna il numero di stelle della recensione, da 1 a 5
  public int howManyStars() { ... }
}
```

Esercizio 2 [6 punti] Si completi la seguente classe, che rappresenta una stanza:

```
public class Room {
  ...

  // costruisce una stanza con una descrizione, un numero massimo di occupanti e un prezzo giornaliero
  public Room(String description, int people, int pricePerDay) { .... }

  // aggiunge la review a quelle di questa stanza
  public void addReview(Review review) { ... }

  // restituisce la descrizione della stanza, seguita dal numero medio di stelle, seguito
  // dal numero massimo di persone, seguito dal prezzo giornaliero richiesto,
  // seguito dal toString() di ciascuna recensione ricevuta da questa stanza
  @Override public String toString() { ... }

  // ritorna il numero massimo di persone che possono occupare la stanza
  public int getPeople() { ... }

  // ritorna il prezzo giornaliero richiesto
  public int getPriceForDay() { ... }

  // ritorna la media delle stelle delle recensioni ricevute; ritorna 0.0 in caso di nessuna recensione
  public double averageStars() { ... }
}
```

Esercizio 3 [6 punti] Si completi la seguente classe, che rappresenta il sistema di prenotazione di un insieme di stanze. È possibile aggiungere stanze al sistema ed è possibile ottenere l'insieme delle stanze compatibili con il criterio di selezione prescelto (numero minimo di persone e prezzo giornaliero massimo richiesto):

```
public class HairBnB {
  ...

  // aggiunge a questo sistema le stanze fornite come argomenti
  public void addRooms(Room... rooms) { ... }
```

```
  // restituisce l'insieme delle stanze che abbiano spazio per almeno minPeople
  // e che costino al massimo maxPrice al giorno; questo metodo deve lanciare
  // una NoRoomAvailableException se nessuna stanza soddisfacesse tali richieste
  public Set<Room> availableFor(int minPeople, int maxPrice) { ... }
}
```

Esercizio 4 [1 punto] Si definisca l'eccezione `NoRoomAvailableException` per l'esercizio 3.

Esercizio 5 [7 punti] Si aggiunga il seguente metodo alla classe `HairBnB` dell'esercizio 3:

```
public SortedSet<Room> sortedAvailableFor(int minPeople, int maxPrice, Comparator<Room> comparator) {
  ...
}
```

che si comporta esattamente come `availableFor()`, incluso il lancio dell'eccezione, ma si differenzia solo per il fatto che restituisce un insieme ordinato di stanze. L'ordinamento è specificato dal parametro `comparator`, che implementa l'interfaccia della libreria standard:

```
public interface Comparator<T> {
  // restituisce:
  // un numero negativo se o1 viene prima di o2;
  // un numero positivo se o2 viene prima di o1;
  // 0 se o1 e o2 sono uguali
  int compare(T o1, T o2);
}
```

Suggerimento: normalmente la classe di libreria `TreeSet<T>` realizza un insieme ordinato di `T` e tale tipo generico deve implementare `Comparable<T>`. È però possibile definire dei `TreeSet<T>` anche per tipi `T` che non implementino `Comparable<T>`, purché il criterio di confronto venga fornito al momento della costruzione dell'insieme, tramite un `Comparator<T>`, così: `new TreeSet<T>(comparator)`.

Esercizio 6 [7 punti] Si completi la seguente classe di prova, inizializzando le variabili `queryByPrice` e `queryByStars`:

```
public class Main {
  public static void main(String[] args) {
    Room room1 = new Room("Room with view in Borgo Roma", 2, 30);
    Room room2 = new Room("Nice flat in Chievo", 4, 70);
    Room room3 = new Room("Historic room in Verona", 1, 80);

    room1.addReview(new Review("Albert Einstein",
      "I loved the room, large and with view on the hospital", Review.Stars.FOUR));
    room1.addReview(new Review("Liz Taylor", "Ugly place, full of hair", Review.Stars.ONE));
    room2.addReview(new Review("Vasco Rossi", "Really lovely", Review.Stars.FIVE));

    HairBnB bnb = new HairBnB();
    bnb.addRooms(room1, room2, room3);

    // queryByPrice devono essere le stanze per almeno una persona, che costano al massimo 70 al giorno,
    // ordinate per prezzo crescente (cioe' prima quelle che costano meno)
    SortedSet<Room> queryByPrice = ...

    // come queryByPrice, ma ordinate per valutazione decrescente (cioe' prima quelle con valutazione maggiore)
    SortedSet<Room> queryByStars = ...
  }
}
```

È possibile definire campi, metodi, costruttori e classi aggiuntive, ma solo `private`.