Labor 3XHIF

RandomAcessFile

1) Eine Datei hat folgenden Aufbau:
   Zunächst steht ein Byte mit Wert 0 oder 1. Hat das Byte den Wert 0, so folgt ein (binär geschriebener) Integer, hat das Byte den Wert 1, so folgt ein (binär geschriebener) Double.
   Diese Sequenz (Byte - Integer oder Double) wiederholt sich bis zum Dateiende.

   a) Schreiben Sie eine Klassenmethode, welcher der Dateiname als `String` übergeben wird und die eine Liste vom Typ `List<Number>` liefert, in der alle in der Datei gespeicherten Werte als Wrapperobjekte vom Typ `java.lang.Integer` bzw. `java.lang.Double` gespeichert sind.

   b) Schreiben Sie eine Methode
   `Map<String,Integer> analyze(List<? extends Number> l) { ... }`
   welche die in a) erzeugte Liste auf die Anzahl der gespeicherten Integer- und Doublewerte analysiert. Die retournierte Map sollte (bei 17 Integerwerten und 36 Doublewerten) also folgenden Aufbau haben:
   `{int=17, double=36}`
   Was bedeutet der Compilertyp `List<? extends Number> l`?

2) Die beiliegende Datei `db-1x2.db` hat den folgenden Aufbau:

   - Start der Datei:
     → 4 Byte numerisch, Kennzahl zur Identifizierung der Datei
     → 4 Byte numerisch, Offset des ersten Datensatzes
     → 2 Byte numerisch, Anzahl der Datenfelder in jedem Record

   - Abschnitt zur Beschreibung des Datenschemas
     Wiederholt sich für jedes Datenfeld
     → 2 Byte numerisch, Länge des Feldnamens in Byte
     → n Byte Feldname (definiert durch den vorigen Eintrag)
     → 2 Byte numerisch, Feldlänge in Byte
     Ende des Wiederholungsblocks

   - Datenabschnitt
     Wiederholt sich bis zum Dateiende
     → 2 Byte Flag, 0x0000 bedeutet einen gültigen Datensatz, 0x8000 bedeutet einen gelöschten Datensatz.
     → Datensatz. Dieser enthält Felder in der Reihenfolge wie im Schemaabschnitt beschrieben, keine Separatoren zwischen den Feldern, jedes Feld mit fixer Länge wie in der Schemabeschreibung angegeben.

Öffnen Sie die Datei (als RandomAccessFile) zum Lesen und lesen Sie die Inhalte des Startabschnittes und der Schemabeschreibung aus. Geben Sie diese Inhalte in geeigneter Form auf der Konsole aus. Bestimmen Sie außerdem die Anzahl der gültigen bzw. als gelöscht markierten Datensätze im Datenabschnitt und geben Sie diese Informationen aus.

Schreiben Sie eine Klasse Hotel, welche die in Aufgabe 1 bestimmten Datensätze in Objekten verwaltet. Die Feldbeschreibung ist über eine innere `enum Datenfeld` zu verwalten. Ihre Klasse hat also folgendes Grundgerüst:

```
public class Hotel {
  public enum Datenfeld {
  // ...
  private int len;
  private String bezeichnung;
}
  //...
  public Hotel(byte []data) { / * .... * / }
  public byte[] getBytes() { / * .... * / }
}
```

Geben Sie mit Hilfe ihrer Klasse die in der gegebenen Datei gespeicherten Objekte aufsteigend sortiert nach dem Ort und bei gleichem Ort aufsteigend sortiert nach dem Hotelnamen auf `stdout` aus.

3) Entwickeln Sie zu ihrer Klasse Hotel wenigstens 5 Testfälle, in denen Sie prüfen, ob die Verwandlung einer Hotel-Instanz in ein Bytearray und umgekehrt richtig arbeiten und schreiben Sie geeignete JUnit-Tests für diese Testfälle. Die Testfälle sind im Kommentar der entsprechenden Testmethode ausführlich zu beschreiben.