Programação Orientada pelos Objectos

Exame (época normal)
2007/06/19

Atenção: A fraude numa prova de avaliação, mesmo quando detectada após a prova, é punida com a reprovação na cadeira.

Cada uma das questões do primeiro grupo vale 5 pontos. Cada uma das questões 1 a 8 vale 10 pontos. A questão 9 vale 20 pontos. A questão 10 vale 50 pontos.

Primeiro grupo

Por favor, responda imediatamente ao questionário anexo. Note bem, o questionário será recolhido 15 minutos após o início da prova.

Segundo grupo

Nesta nova época balnear, o concessionário da praia da sereia, na nossa Costa da Caparica, instalou um sofisticado sistema informático para gerir o aluguer diário das palhotas. As palhotas ocupam um espaço rectangular e estão arrumadas em filas paralelas ao mar, formando colunas de palhotas, perpendiculares ao mar. As palhotas são numeradas a partir de 1, de norte para sul, e depois da borda de água para as dunas.

O sistema baseia-se na classe Beach, a qual tem campos privados para o número de filas e para número de colunas de palhotas. Há também um vector com os nomes das famílias que ocupam cada palhota (na posição *x* do vector está o nome da família que ocupa a palhota *x*+1) e uma tabela de dispersão em que a chave é o nome da família e o valor é o número da palhota que a família ocupa. Nesta primeira versão, o sistema não aceita duas famílias com o mesmo nome ☹

A classe tem métodos para verificar se uma palhota está livre, para alugar uma palhota a uma família, para verificar se uma família está presente na praia (porque alugou uma palhota), para calcular a fila e a coluna de uma dada palhota, para calcular a distância entre duas famílias e para listar por ordem alfabética as famílias presentes juntamente com o número das respectivas palhotas. A distância é a "distância de Manhattan", medida ao longo das filas e das colunas. Veja a função de teste unitário mais abaixo. Na listagem, vem uma família em cada linha, separada do número da palhota por um espaço. Eis a classe, em esquema:

```
public class Beach
{
  private int rows;
  private int columns;
  private String[] families;
  private HashMap<String, Integer> rental;

  public Beach(int rows, int columns)
  {
    this.rows = rows;
    this.columns = columns;
    families = new String[this.rows * this.columns];
    rental = new HashMap<…>(…);
  }

  public boolean isFree(int hut)
  {
    return families[hut - 1] == null;
  }

  public void rent(int hut, String family)
  {
    assert isFree(hut);
    assert !isPresent(family);
    // …
  }

  public boolean isPresent(String family)
  {
    return rental.containsKey(family);
  }

  public Pair<Integer, Integer> position(int hut)
  {
    int r = (hut - 1) / columns + 1;
    int c = (hut - 1) % columns + 1;
    return new Pair<Integer, Integer>(r, c);
  }

  public int distance(String f1, String f2)
  {
    assert isPresent(f1);
    assert isPresent(f2);
    // …
  }

  public void printFamilies(PrintWriter out)
  {
    // …
  }

}
```

A classe para os testes unitários contém a seguinte função para testar a função distance:

```
@Test
public void testDistance()
{
  Beach b = new Beach(3, 10);
  b.rent(4, "silva");
```

```
    b.rent(20, "sousa");
    assertEquals(7, b.distance("silva", "sousa"));
  }
```

1. Por favor, complete a função `rent`.
2. Por favor, complete a função `distance`.
3. Por favor, complete a função `printFamilies`.
4. Por favor, programe a função `testPosition`, para realizar o teste unitário da função `position`. Teste três casos interessantes.

Terceiro grupo

Um dos muitos estudos que estão a ser feitos por causa do novo aeroporto é sobre o tempo entre o momento em que um avião aterra e o momento em que a primeira mala aparece no tapete rolante. Trata-se de um importante indicador sobre a qualidade do serviço do aeroporto, como todos nós, que desembarcamos na Portela frequentemente, infelizmente sabemos muito bem ☹

Pois bem: queremos calcular o *tempo médio de espera pela primeira mala, por passageiro*. Por exemplo, suponha que há dois voos: no primeiro, com 64 passageiros, a primeira mala demorou 20 minutos; no segundo, com 36 passageiros, demorou 30 minutos. Então o tempo médio é (20 * 64 + 30 * 36) / (64 + 36), ou seja, 23.6.

Para calcular este indicador precisamos de duas classes: a classe `Flight`, que representa um voo, com informações sobre a origem, a companhia, o número de passageiros, a hora da aterragem e a hora de saída da primeira mala, e a classe `Airport`, que processa os registos de tipo `Flight` para calcular o indicador. As horas são representadas por números inteiros. Por exemplo, 1245 representa um quarto para a uma da tarde.

5. Por favor, programe em esquema a classe `Flight`, indicando apenas os campos.
6. Por favor, programe um método estático `parseFlight` que dada uma cadeia representando os valores dos campos de um objecto de tipo `Flight` devolve um objecto novo, devidamente inicializado. Na cadeia, os valores estão separados por vírgulas. Por exemplo, “paris,tap,124,2012,2133” representa o voo da Tap, proveniente de Paris, com 124 passageiros, que aterrou às 20:12 e cuja primeira mala saiu às 21:33. Se o último número for menor do que o penúltimo, então a primeira mala saiu já depois da meia-noite.
7. Por favor, programe um método para calcular o número de minutos que passaram desde que o avião aterrou até que a primeira mala saiu.
8. Por favor, programe uma classe `Airport` com um campo de tipo `ArrayList<Flight>` e com um método para ler para esse vector um ficheiro de dados passado num argumento de tipo `Scanner`. Cada linha do ficheiro vem no formato explicado na questão 6.
9. Por favor, programe na classe `Airport` um método para calcular o tempo médio de espera pela primeira mala, por passageiro. Presuma que todos os *getters* da classe `Flight` estão disponíveis. Requisito técnico: use um ciclo for-each.

Quarto grupo

Você lembra-se do João, aquele que morava numa casa que tinha 10 degraus? Pois bem, a fachada da casa tem várias janelas e a Maria, que é a mulher do João, decidiu mudar as cortinas. Por razões que só o espírito feminino consegue entender, a Maria quer alternar as cores das cortinas das sucessivas janelas (sucessivas no sentido da esquerda para a direita, quando se olha para a casa a partir da rua): uma azul, outra amarela, etc. Acontece que se trata de uma casa com uma arquitectura moderna em que nem todas as janelas são iguais. Acresce que o tecido azul é mais caro do que o amarelo. O problema é calcular quanto tecido azul e quanto tecido amarelo é preciso comprar, dadas as dimensões de cada janela, de maneira a gastar menos. Note que a primeira janela pode ficar azul ou amarela, conforme ficar mais barato. O tecido para as cortinas é vendido ao centímetro quadrado.

A primeira linha do ficheiro de dados indica o número de janelas, *N*. Em cada uma das *N* linhas seguintes vêm dois números inteiros, representado a largura e altura de cada janela expressas em centímetros, da esquerda para a direita. O ficheiro de resultados contém apenas uma linha, com dois números inteiros separados por um espaço: a quantidade de tecido azul e a quantidade de tecido amarelo expressas em centímetros quadrados.

Exemplo de ficheiro de dados:

```
5
120 80
100 70
90 70
150 90
40 40
```

Respectivo ficheiro de resultados:

```
17500 20500
```

10. Por favor, escreva um programa para resolver o problema das cortinas da Maria.