Programação Orientada pelos Objectos

Exame (época normal)
2009/06/16

Atenção: A fraude, mesmo quando detectada após a prova, é punida com a reprovação na cadeira.
Sugestão: Resolva o exame utilizando **lápis e borracha**. Evite rasuras!

Grupo 1

A) Considere a classe IterableUtilities, da qual não se representa o corpo dos métodos, que são estáticos e genéricos. Esta classe permite manipular colecções iteráveis.

```
public class IterableUtilities
{
   // the size of the iterable collection
   public static <E> int size(Iterable<E> a) {…}
   // the number of times that the object occurs in the iterable collection
   public static <E> int counts(E object, Iterable<E> a) {…}
   // true if the object occurs in the iterable collection
   public static <E> boolean exists(E object, Iterable<E> a) {…}
   // true if iterable collection a1 contains all elements of a2 (a2 is subset of a1)
   public static <E> boolean contains(Iterable<E> a1, Iterable<E> a2) {…}
   // removes the duplicates from the iterable collection
   public static <E> void removeDuplicates(Iterable<E> a) {…}
   // the intersection of iterable collections a1 and a2 (without duplicates)
   public static <E> Iterable<E> intersection(Iterable<E> a1, Iterable<E> a2) {…}
   // the union of iterable collections a1 and a2 (without duplicates)
   public static <E> Iterable<E> union(Iterable<E> a1, Iterable<E> a2) {…}
   // converts from an iterator to an iterable collection of type E elements
   public static <E> Iterable<E> toIterable(Iterator<E> it) {…}
}
```

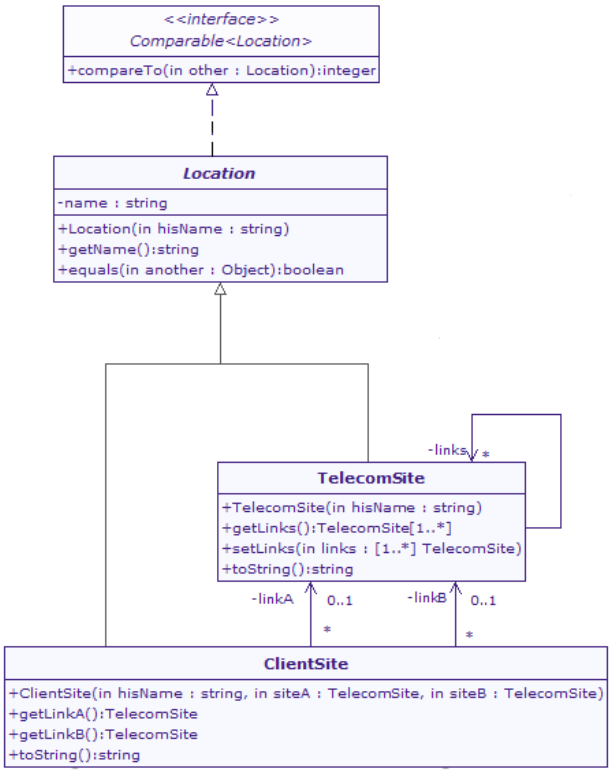
A) Implemente a classe de testes unitários (JUnit) cujo esqueleto se fornece seguidamente. Construa apenas os cenários de teste com os valores descritos em comentário.

```
public class IterableUtilitiesTest
{
   private static ArrayList<Integer> it1, it2;
   // initialize it1 from Integer[] in1 = {1, 2, 3, 4, 5, 2, 3, 2};
   // initialize it2 from Integer[] in2 = {2, 5, 9, 11};

   @Test // it1 has 8 elements and it2 has 4
   public final void testSize() {…}

   @Test // number 2 occurs three times in it1; number 5 occurs once in it2
   public final void testCounts() {…}

   @Test // number 2 appears in it1 and number 11 appears in it2
         // number 0 is neither in it1, nor in it2
   public final void testExists() {…}

   @Test // it2 is not a subset of it1
         // the intersection of it1 and it2 is a subset of it1
   public final void testIncludes() {…}

   @Test // if we remove the duplicates from clone1, it will keep only 5 numbers
   public final void testRemoveDuplicates()
   {
         Iterable<Integer> clone1 = new ArrayList<Integer>(it1);
         …
   }

   @Test // the intersection of it1 and it2 has 2 elements
         // the intersection of it1 and it2 contains the numbers 2 and 5
   public final void testIntersection() {…}

   @Test // the union of it1 and it2 has 7 elements
         // numbers 3 and 11 belong to the union of it1 and it2, but not number 10
   public final void testUnion() {…}
}
```

B) Implemente todos os métodos da classe IterableUtilities.

Grupo 2

O operador de comunicações *NovaTel* está a desenvolver um novo pacote de software para suportar o planeamento das suas redes de fibra óptica. Os nós destas redes são chamados localizações (classe abstracta Location). As localizações podem ser ou estações próprias da *NovaTel* (classe TelecomSite) ou nas instalações do cliente (classe ClientSite). Estas últimas estão sempre ligadas, bidireccionalmente, a duas estações da *NovaTel*, por uma questão de segurança. Cada estação da *NovaTel* pode estar ligada a um qualquer número de outras estações, para permitir qualquer configuração da rede. Cada uma das ligações (links) é unidireccional, partindo do nó actual. Implemente as classes Location, TelecomSite e ClientSite, de acordo com o diagrama UML junto. Note que a classe TelecomSite deve ter o atributo *links* do tipo *TelecomSite[]* e a classe ClientSite deve ter os atributos *linkA* e *linkB* do tipo *TelecomSite.* As operações toString() devem permitir serializar toda a informação relevante sobre o estado dos objectos respectivos.



Grupo 3

A *UNL*, pretendendo melhorar a velocidade de acesso à internet dos seus 7 campus, fez um contrato com a *NovaTel* para os ligar à rede de fibra óptica que aquela empresa montou na região da Grande Lisboa. As estruturas de dados que representam os nós relativos a todas as localizações envolvidas são as seguintes (CLIENTS=7 e SITE=20 são constantes):

*ClientSite[] c_sites = **new** ClientSite[CLIENTS];*
*TelecomSite[] t_sites = **new** TelecomSite[SITES];*

As localizações do cliente correspondem neste caso a cada um dos campus. Essa informação está guardada num ficheiro com um formato a 3 colunas separadas com vírgulas, que são referentes, respectivamente, ao nome do site do cliente e ao número das duas estações da *NovaTel* a que o site do cliente está ligado.

Oeiras (ITQB), 15, 18
Campolide (Reitoria FE FD ISEGI), 1, 4
Mártires (FCM), 6, 8
Berna (FCSH), 4, 5
Junqueira (IHMT), 15, 11;
Lumiar (ENSP), 9, 10
Caparica (FCT), 0, 1

***Nota**: os números referem-se a posições em t_sites.*

A) Implemente uma operação que preenche c_sites lendo a informação deste ficheiro (chame-lhe initializeCampusLocations()). Essa operação deverá tratar a excepção FileNotFoundException.

A rede de fibra óptica, que a *NovaTel* montou na região da Grande Lisboa, é em forma de anel em que cada nó se liga ao anterior e ao seguinte. Claro, que para ser um anel, o último nó liga ao primeiro e vice-versa.

B) Implemente initializeTelecomLocations(), a operação que preenche t_sites.

Grupo 4

Para fazer o planeamento dos encaminhamentos na rede da *NovaTel* é necessário representá-la em grafo. Desta forma podemos usar algoritmos que o atravessem.

A) Implemente a operação buildGraph() que, a partir de t_sites e c_sites, constrói o grafo network representando toda a rede da *NovaTel* com os campus da *UNL* lá ligados.

***Nota**: use a variável IGraph<Location> network.*

B) Implemente a seguinte operação que calcula o caminho mais curto sobre a network, que passa por todos os campus e começa em start e termina em stop.

Iterable<Location>
shortestWithAllCampus(Location start, Location stop)

***Sugestão**: use a classe IterableUtilities do grupo 1 deste enunciado.*

C) Construa o esqueleto da classe NovaTel contendo todos os atributos, constantes e operações desenvolvidas nos grupos desta página (não reproduza o corpo das operações mas apenas o seu cabeçalho). Nesta classe escreva o programa principal que calcula o caminho mais curto entre Oeiras (c_sites[0]) e Caparica (c_sites[6]), que passa por todos os campus da UNL.