Curso 2018-2019. Cuatrimestre de Otoño. 10 de Enero de 2019

Programación Orientada a Objetos (POO). Examen final. Duración del examen: 3 horas

Publicación de notas provisionales: 22/1/2019.

Revisión de examen: 24/1/2019. Hora: 11:00h. Lugar: D6-221

El diagrama de clases UML (incompleto, ya que no se muestran las excepciones ni los métodos de las clases) que se muestra en la figura DE LA ÚLTIMA PÁGINA pretende mostrar el núcleo de un simulador de red de transporte de datos tipo Internet.

El programa deberá simular una red formada por un número indeterminado de **encaminadores** (dispositivos capaces de recibir y reenviar datos), **"routers"** en terminología inglesa (que usaremos de ahora en adelante). Los "routers" están interconectados por **enlaces** de transmisión que consideraremos unidireccionales. Cada "router" está interconectado con uno o más "routers". Entre dos "routers" solo puede haber UN enlace como mucho en un sentido.

Todo "router" dispone de un identificador único (valor entero) y de un grupo de enlaces hacia diferentes "routers". El contenedor elegido para almacenar dichos enlaces deberá proporcionar una búsqueda eficiente a partir del identificador del "router" destino (vecino) al que conecta éste.

Un enlace siempre conecta dos "routers" de la red (origen y destino). Cada enlace tendrá además tres características definitorias: su longitud, su capacidad total y nivel de ocupación (por ejemplo, un enlace puede tener una capacidad de 100 y tener un nivel de ocupación de 70 en un instante dado –esto es, en ese instante todavía podría admitir una nueva transmisión que requiriera una capacidad de red igual a 30, como máximo).

El simulador se utiliza para estudiar el comportamiento de la red simulada ante la llegada de peticiones de servicio. Cada petición de servicio dispone de un identificador único. Cada petición solicita el envío de información de un "router" origen a un "router" destino de la red. Toda petición incluye además el valor de la capacidad de enlace solicitada (CES), esto es, la capacidad que la información a transmitir ocupará en los enlaces atravesados desde el "router" origen al "router" destino.

Para cada petición el simulador debe intentar encontrar, utilizando un cierto algoritmo de encaminamiento, una ruta (entendida como una secuencia de enlaces que la información debe atravesar) entre dichos "routers", asegurándose de que todos los enlaces de dicha ruta tengan capacidad libre mayor o igual a la CES antes mencionada.

REGLA 1: Para resolver el resto de las preguntas del examen podéis suponer implementados los métodos inmediatos getXXX() y setXXX() que acceden a los atributos de las clases. **No es necesario que los implementéis a no ser que se indique lo contrario**. Si durante la resolución del resto de preguntas necesitáis de métodos más complejos que éstos, **DEBERÉIS IMPLEMENTARLOS.**

REGLA 2: Si consideráis que para implementar un método necesitáis invocar a alguno de los métodos cuya implementación se pide en otras preguntas (digamos metodo1(….)), **SOIS LIBRES DE INVOCAR A ESTE SEGUNDO MÉTODO (metodo1(….)) INDEPENDIENTEMENTE DE QUE HAYÁIS SABIDO IMPLEMENTARLO O NO**.

Problema 1 (1 punto).A partir del diagrama de clases UML de la figura, escribid para todos los componentes presentes en él, la parte de código de su definición, que va desde el inicio de dicha definición e incluye la declaración de los todos sus atributos, incluyendo aquellos que aparecen al implementar las relaciones mostradas en el diagrama. **NO debéis añadir nada más** en vuestras respuestas a esta pregunta.

Problema 2 (1 punto). Proponed e implementad un constructor para las siguientes clases:

- **Router**. Suponed que cuando se invoque se conoce solo su identificador. Proponed e implementad un constructor de la clase
- **Peticion**. Suponed que cuando se invoque se conocen solamente la CES, el "router" origen y el "router" destino de la transmisión. Haced que el constructor calcule un identificador único.

Problema 3 (1 punto). Implementad en la clase **Router**:

```
public void anotaEnlace(Enlace enlace)throws EnlaceException
```

Si el enlace pasado **NO** tiene a este "router" (el objeto "router" que contiene al método cuando se ejecuta) como "router" origen, genera y lanza una excepción `EnlaceException`. En caso contrario encuentra con que otro "router" el enlace pasado como argumento conecta a este "router" y realiza la correspondiente anotación en el contenedor de enlaces.

```
public Enlace getEnlaceHaciaRouter(int idRouter)
```

Este método devuelve el enlace que conecta este "router" con el "router" cuyo identificador se pasa como argumento o null si tal enlace no existe.

Problema 4 (1 punto). Implementad en la clase **Enlace**:

- Un constructor. Suponed que cuando se invoca se conocen su capacidad total, su longitud y los dos "routers" que interconecta. El constructor debe anotar también dicho enlace en el "router" origen del enlace.
- El método

  ```
  public boolean conectaEstosRouters(int idRouterOrigen, int idRouterDestino)
  ```

  Este método devuelve `true` si este enlace realmente conecta los routers cuyos identificadores se pasan como argumento, y `false` en caso contrario.

Problema 5 (1 punto). Implementad el siguiente método en la clase **Red**:

```
public Enlace getEnlaceQueConecta(int idRouterOrigen, int idRouterDestino)
```

Este método intenta localizar, entre todos los enlaces de la red, el enlace que conecta los "routers" cuyos identificadores se pasan como argumentos. Devuelve una referencia a dicho enlace si lo ha encontrado, o null en caso contrario.

Problema 6 (3 puntos). La clase **RedReader** se encarga de leer de un archivo de texto información que permite crear un objeto Red. Implementad en la clase **RedReader** los siguientes métodos:

```
protected Enlace creaEnlaceDeLinea(String str, Red red) throws
FormatoEnlaceException, EnlaceException
```

Este método recibe en el argumento `str` un String que contiene la información de un enlace según el patrón:

<identificador de "router" origen> <identificador de "router" destino> <capacidad de enlace> <longitud del enlace>.

Por ejemplo, el string "1 2 300 10" informaría de un enlace que interconecta al "router" 1 con el "router" 2, de que tiene una capacidad de 300 y de que tiene una longitud de 10.

Este método también recibe el argumento `red` que contiene una red parcialmente construida. Debéis suponer que en el contenedor de "routers" de dicho objeto están presentes TODOS los "routers" de la red (para crear un objeto Enlace es necesario que existan los dos objetos Router que dicho enlace interconecta).

El método debe crear el objeto Enlace descrito en el string y devolverlo si el string contiene la representación textual de 4 números enteros. Si no es así, el método debe crear y lanzar una excepción `FormatoEnlaceException`.

Se supone que, si los dos primeros valores de la línea representan dos enteros, éstos se corresponden con identificadores de nodo que efectivamente existen en la red.

```
public Red readRedFromFile(String fName) throws RedReadingException
```

Este método debe crear un objeto Red, leer los detalles de la red de un archivo de texto cuyo nombre completo se pasa como argumento, crear todos los "routers" y enlaces de la red, depositarlos convenientemente en los contenedores respectivos y devolver el objeto Red como resultado.

A continuación, se indica el formato del archivo de texto:

- La primera línea contiene un valor que indica el número de “routers” de la red.
- Las líneas restantes contienen información de cada uno de los enlaces que forman la red según el patrón detallado para el método anterior.

SUGERENCIA IMPORTANTE: Después de haber leído la primera línea cread todos los “routers” de la red. Después, pasad a leer el resto de las líneas y cread los enlaces que indiquen las líneas.

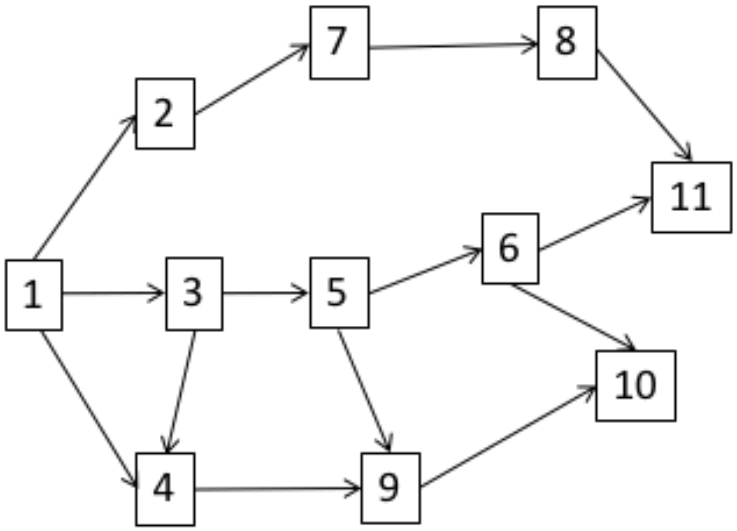
Problema 7 (2 puntos). Podréis haber observado que una red como la descrita puede modelarse con un grafo. De hecho, las clases Router y Enlace conforman una de las varias alternativas de implementar grafos (la que se presentó en clase no es la única), jugando la clase Router el papel de nodo del grafo. Por tanto, los algoritmos de gestión de grafos pueden ser también utilizados.

Implementad en la clase **BreadthFirstSearch el método que se indica a continuación que implemente el algoritmo detallado líneas abajo**:

```
private Integer[] runAlg(int idOrigen,int volumen)
```

Este método implementa el algoritmo BreadthFirstSearch que se detalla más abajo y devuelve un vector cuya dimensión coincide con el número de “routers” de la red. En ese vector se mostrarán las mejores conexiones para llegar a cada uno de los “routers” de la red a partir del “router” cuyo identificador es el pasado en el argumento `idOrigen`, entendiendo como mejores conexiones las que impliquen menos saltos. Si hay algún “router” al que no puede llegarse, en la posición correspondiente a su identificador aparecerá un null.

Por ejemplo, para la red de la figura que sigue



El vector de mejores conexiones a partir del “router” cuyo identificador es 1, será: **{null,1, 1, 1, 3, 5, 2, 7, 4, 9, 8}**. Observad: para llegar al “router” 1 no hay que hacer nada (null en posición 0 del vector); a los “routers” 2, 3, y 4 (posiciones 1, 2, y 3 del vector) se llega directamente desde el “router” 1, y en consecuencia lo tienen como “router” previo; el “router” 5 (posición 4) tiene como “router” previo el 3; el “router” 6 (posición 5) tiene como “router” previo el 5; el “router” 7 (posición 6) tiene como “router” previo el 2, el “router” 8 (posición 7) tiene como “router” previo el 7; el “router” 9 (posición 8) tiene como previo al 4 (ruta de menos saltos para llegar a él: 1-4-9, aunque hay otra que tiene más saltos); el “router” 10 (posición 9) tiene como previo al 9 (nuevamente hay otra ruta con más saltos que no se selecciona); el “router” 11 (posición 10) tiene como “router” previo al “router” 8 (la ruta de menos saltos).

Para construir el vector de mejores conexiones **debéis implementar el algoritmo que se muestra a continuación**.

1. Crear el vector de mejores conexiones con una dimensión igual al número de “routers” de la red.
2. Crear un conjunto de identificadores de “routers” visitados. Inicialmente está vacío.
3. Crear una COLA de identificadores de “router” en estudio (una lista). Inicialmente está vacía.
4. Añadir a la cola el identificador del “router” origen.
5. Añadir al conjunto de identificadores de “routers” visitados el identificador del “router” origen.

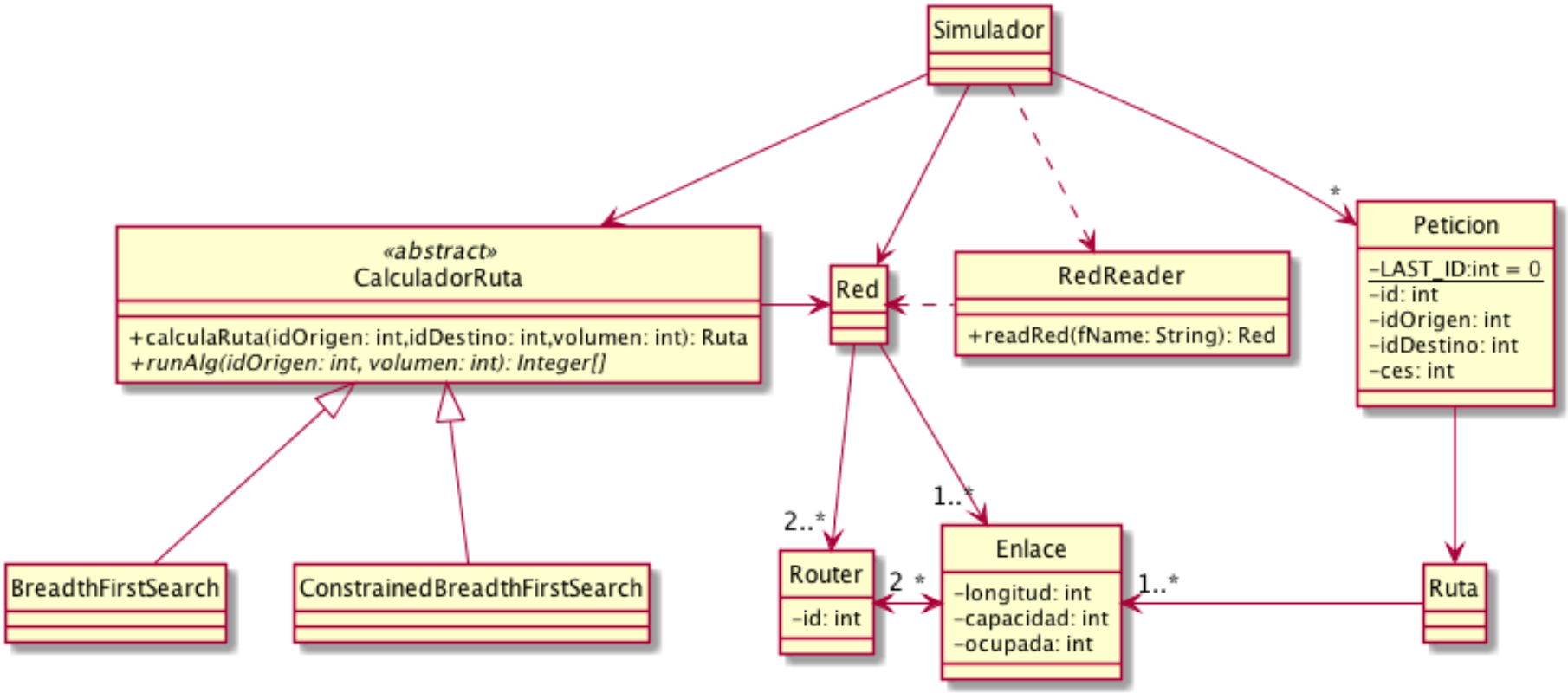
6. Mientras la COLA NO esté vacía.
    a. Hacer idRouterRef = resultado de extraer el primer identificador de “router” de la COLA
    b. Hacer routerRef = “router” cuyo identificador es el extraido de la COLA.
    c. Para cada uno de los enlaces de este router
        i. Hacer routerConectado = “router” destino del enlace.
        ii. Si el identificador de routerConectado NO está en conjunto de identificadores de “routers” visitados
            1. Hacer mejoresConexiones[idRouterConectado-1] = idRouterRef
            2. Añadir el identificador de routerConectado a conjunto de identificadores de “routers” visitados
            3. Añadir a COLA el identificador de routerConectado
7. Devolver vector mejoresConexiones.

Implementad en la clase **CalculadorRuta**, el siguiente método:

```
public Ruta calculaRuta(int idOrigen, int idDestino, int volumen) throws RutaException
```

Este método intenta calcular la ruta entre dos “routers” de la red, si existe, con el menor número de saltos, sin considerar la capacidad solicitada ni las capacidades disponibles de los enlaces. Para calcular la ruta hace uso del método `runAlg`. A continuación se describe el algoritmo que debéis implementar:

1. Crear una ruta vacía
2. Si uno de los identificadores de “routers” está fuera de rango (el menor valor para el identificador de “router” es 1; razonad cuál es el mayor), lanzar una excepción `RutaException` que dé cuenta de ellos.
3. Si los identificadores del “router” origen y destino son iguales, lanzar una excepción `RutaException` que dé cuenta de ello.
4. Encontrad el vector de mejores conexiones tomando como “router” de partida el “router” cuyo identificador es el valor del argumento `idOrigen`.
5. Hacer idRouterInspeccionado = `idDestino`.
6. Mientras idRouterInspeccionado!= `idOrigen` de mejores conexiones
    a. Encontrad el identificador del “router” previo al “router” con identificador idRouterInspeccionado en el vector de mejores conexiones (recordad: será el valor de la posición anterior) y asignad dicho valor a idRouterPrevio.
    b. Si idRouterPrevio es `null`,lanzar una excepción `RutaException` con un mensaje que informe de que el “router” con identificador idRouterInspeccionado no está conectado.
    c. Buscar enlace que conecta los “routers” con identificadores idRouterInspeccionado e idRouterPrevio.
    d. Si la búsqueda del enlace devuelve null, lanzar una excepción `RutaException` con un mensaje que informe de que el “router” no está conectado con otro “router” de la red.
    e. En caso contrario, añadir el enlace a la ruta como el primero de la ruta.
    f. Hacer idRouterInspeccionado = idRouterPrevio.
7. Devolver ruta.


Simulador
«abstract»
CalculadorRuta
+calculaRuta(idOrigen: int,idDestino: int,volumen: int): Ruta
+runAlg(idOrigen: int, volumen: int): Integer[]
Red
RedReader
+readRed(fName: String): Red
Peticion
-LAST_ID:int = 0
-id: int
-idOrigen: int
-idDestino: int
-ces: int
BreadthFirstSearch
ConstrainedBreadthFirstSearch
2..*
1..*
Router
-id: int
2
*
Enlace
-longitud: int
-capacidad: int
-ocupada: int
1..*
Ruta
*