UNED

ETS de Ingeniería Informática

Estrategias de Programación y Estructuras de Datos

Grado en Ingeniería Informática
Grado en Tecnologías de la Información

Práctica curso 2015-2016

Enunciado

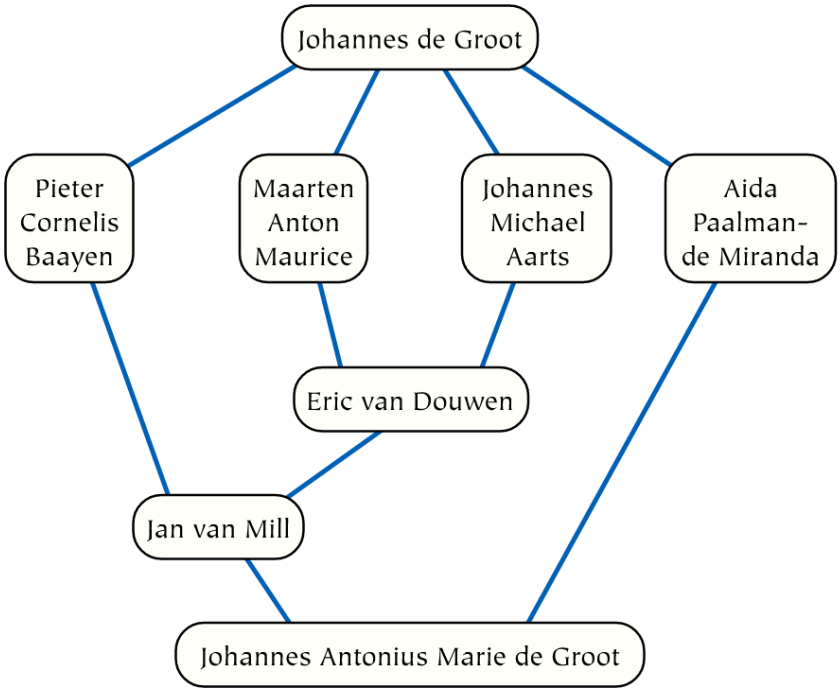
Índice

1. Presentación del problema....3
2. Diseño....4
2.1 Tipos de datos de referencia....4
2.2 Primer escenario: Academia con Árbol Genealógico Simplificado....6
Preguntas teóricas (trabajo del estudiante)....7
2.3 Segundo escenario: Academia con Árbol Genealógico Completo....8
Preguntas teóricas (trabajo del estudiante)....9
3. Implementación....9
3.1 Parámetros de entrada....10
3.2 Estructura del fichero de datos....10
3.3 Estructura del fichero de operaciones....11
3.4 Medición de tiempos....12
Estudio empírico del coste temporal (trabajo del estudiante)....12
3.5 Salida del programa....12
4. Ejecución y juegos de prueba....13
5. Documentación y plazos de entrega....13

1. Presentación del problema

Una Academia está formada por una serie de Doctores, que entran a formar parte de ella tras leer una Tesis doctoral. Éstas se realizan de forma individual bajo la supervisión (o dirección) de un director de Tesis (o varios codirectores), que también ha de ser doctor(es) de la Academia, salvo en el caso del fundador de la misma, que pertenece a ella por derecho propio (sin necesidad de haber realizado una Tesis doctoral dentro de la Academia).

En una Academia, las relaciones de dirección de Tesis doctorales conforman el llamado "Árbol Genealógico Académico".



Muestra de relaciones en un árbol genealógico académico tomada de Wikipedia (https://en.wikipedia.org/wiki/Academic_genealogy). La figura indica que *Johannes de Groot* ha supervisado cuatro tesis doctorales: las de *Pieter Baayen*, *Maarten Maurice*, *Johannes Aarts* y *Aida Paalman-de Miranda*. Se recogen también relaciones de doble supervisión, que sólo consideraremos en una segunda aproximación al problema (ver sección 2.3, "Academia con Árbol genealógico completo"). Por ejemplo, la tesis de *Jan Van Mill* fue codirigida por *Pieter Baayen* y *Eric van Douwen*.

En esta práctica gestionaremos la información de una Academia, que será construida a partir de su fundador y de los datos de dirección de Tesis doctorales por parte de Doctores de la Academia. Supondremos además, para simplificar, que un Doctor sólo podrá realizar una única Tesis doctoral (en la práctica son muy pocos los casos en los que una misma persona hace más de una).

Los objetivos principales de esta práctica son:

- Comprender la implementación de Tipos Abstractos de Datos (TAD)
- Ejercitar el uso de TAD
- Implementar nuevos TAD a partir de otros conocidos

Como objetivos adicionales, podemos destacar:

- Familiarizarse con la definición de interfaces de TAD
- Implementar la solución de un problema completo utilizando TAD en Java
- Familiarizarse con la prueba de programas mediante la técnica de Juegos de Pruebas
- Familiarizarse con el uso de entornos de programación (IDE)

2. Diseño.

En esta práctica trabajaremos con dos escenarios diferentes que condicionarán la elección (y, por tanto, el manejo) de las estructuras de datos para almacenar la información de una Academia. Siguiendo las instrucciones de este enunciado de prácticas, los estudiantes deberán (i) razonar sobre el coste de las soluciones a cada uno de los escenarios propuestos, de forma comparada; (ii) implementar ambas soluciones; y (iii) realizar un estudio empírico de su eficiencia respecto al juego de pruebas proporcionado por el equipo docente, al menos.

2.1 Tipos de datos de referencia.

En primer lugar debemos especificar la interfaz (el tipo de datos de referencia) para las posibles estructuras de datos con las que representaremos la Academia.

Las operaciones que se realizan en un TAD se pueden agrupar bajo las siglas **C.R.U.D.**:

- **CREATE - Operaciones de creación**:
  - Contemplaremos una operación de creación que creará una nueva Academia a partir de su fundador.
- **READ - Operaciones de consulta**:
  - Una operación que devuelva el Doctor que fundó la Academia.
  - Una operación que busque un Doctor dentro de la Academia a partir de un identificador unívoco.
  - Una operación que devuelva el número de Doctores que pertenecen a la Academia.
- **UPDATE - Operaciones de modificación:**
  - Una operación que añada un Doctor a la Academia, a partir de la información de la lectura de su Tesis Doctoral.
  - Una operación que añada una relación de supervisión al Árbol Genealógico de la Academia a partir de los Doctores que dicha supervisión relaciona.
- **DELETE – Operaciones de borrado:**
  - Dado que el grado de Doctor es permanente (salvo contadas excepciones) y por simplicidad a efectos de esta práctica, no se añadirá una operación que elimine información, aunque es un requisito básico de toda estructura de datos.

Las operaciones de creación se realizarán mediante constructores, cuya implementación no corresponde a una interfaz.

Una interfaz que responde a los requisitos para las operaciones de consulta y modificación es la siguiente:

```
/* Representación de una Academia formada por una colección de Doctores      */
public interface AcademiaIF extends CollectionIF<DoctorIF> {

  /* Consulta el Doctor que fundó la Academia                                 */
  /* @returns el Doctor fundador de la Academia                               */
  public DoctorIF getFounder();

  /* Busca un Doctor dentro de la Academia a partir de su identificador       */
  /* @pre     el doctor pertenece a la Academia && id > 0                     */
  /* @param   el identificador del Doctor a buscar                            */
  /* @returns el Doctor buscado                                               */
  public DoctorIF getDoctor(int id);

  /* Consulta el número de Doctores que pertenecen a la Academia              */
  /* @returns el número de Doctores pertenecientes a la Academia              */
  public int size();

  /* Añade un nuevo Doctor a la Academia a partir de la lectura de su Tesis   */
  /* @param el nuevo Doctor y su primer director de Tesis                     */
  /* @pre   el nuevo doctor no debe pertenecer a la Academia &&               */
  /*        el supervisor sí debe pertenecer a la Academia                    */
  public void addDoctor (DoctorIF newDoctor, DoctorIF supervisor);

  /* Añade una relación de dirección al Árbol Genealógico de la Academia      */
  /* @param el nuevo Doctor y uno de sus codirectores de Tesis                */
  /* @pre   ambos doctores deben pertenecer a la Academia &&                  */
  /*        no existe una relación de supervisión previa entre ambos          */
  public void addSupervision (DoctorIF student, DoctorIF supervisor);
}
```

Esta interfaz hace uso de una interfaz `DoctorIF`, que se detalla a continuación:

```
/* Representación de un Doctor perteneciente a una Academia                  */
Public interface DoctorIF {

  /* Consulta los ancestros académicos del doctor, limitándose al número de  */
  /*  generaciones indicado por el parámetro.                                 */
  /* @returns la colección de ancestros académicos del doctor limitada al    */
  /*  número de generaciones indicado o hasta llegar al fundador de la       */
  /*  Academia. No deberá contener repeticiones.                              */
  /* @param   número de generaciones a considerar                             */
  /* @pre     generations > 0                                                 */
  public CollectionIF<DoctorIF> getAncestors(int generations);

  /* Consulta los doctores a quienes el doctor ha dirigido sus Tesis.        */
  /* @returns la colección de doctores cuyo director de tesis es el doctor.  */
  public CollectionIF<DoctorIF> getStudents();

  /* Consulta los descendientes académicos del doctor, limitándose al número */
  /*  de generaciones indicado por el parámetro.                              */
  /* @returns la colección de descendientes académicos del doctor limitada   */
  /*  al número de generaciones indicado o hasta llegar a Doctores que no    */
  /*  hayan dirigido ninguna Tesis. No deberá contener repeticiones.          */
  /* @param   número de generaciones a considerar                             */
  /* @pre     generations > 0                                                 */
  public CollectionIF<DoctorIF> getDescendants(int generations);
```

```
  /* Consulta los doctores que comparten director de tesis con el doctor.    */
  /* @returns la colección de hermanos académicos del doctor. No deberá      */
  /*  contener repeticiones ni al doctor llamante                            */
  public CollectionIF<DoctorIF> getSiblings();

  /* Obtiene el identificador del Doctor                                     */
  /* @returns el identificador entero del Doctor, único en la Academia      */
  public int getId();
}
```

Todas estas operaciones de consulta devuelven colecciones de Doctores en las que no podrá haber repeticiones (es decir, cada Doctor sólo podrá aparecer una única vez). Además, dado que los métodos devuelven colecciones de objetos **`DoctorIF`**, el orden de los diferentes objetos es irrelevante.

Para poder identificar de manera unívoca a los diferentes Doctores de una Academia, cada uno de ellos tendrá asociado un número entero positivo único dentro de la Academia. Es decir, no podrán existir dos Doctores en la misma Academia que tengan el mismo identificador.

2.2 Primer escenario: Academia con Árbol Genealógico Simplificado.

Para simplificar, supongamos que una Academia se funda a partir de un **único** primer Doctor. A partir de ese fundador (que consideraremos origen del árbol genealógico de la Academia), nos interesa disponer de un TAD que recoja la información de la Academia, esto es, la colección de Doctores que la forman y la información "genealógica" de la misma, es decir, para cada Doctor deberemos saber a qué otros Doctores dirigió sus Tesis.

Simplificaremos el escenario al añadir la siguiente restricción:

> No se contempla la figura del codirector de Tesis, es decir, **cada Doctor tiene un único director de Tesis** que, necesariamente, será otro Doctor de la Academia.

Para este primer escenario se considerará una clase **`AcademiaS`** que implemente la interfaz **`AcademiaIF`** y que permita almacenar y gestionar la información de una Academia con Árbol Genealógico Simplificado. Nótese que en este escenario el método **`addSupervision()`** descrito en la interfaz **`AcademiaIF`** no tiene efecto alguno.

De igual modo, existirá una clase `DoctorS` que implemente la interfaz `DoctorIF` y que permita gestionar la información de un Doctor dentro de una Academia. A continuación se detallan los atributos y nuevos métodos que deberá tener dicha clase:

```
public class DoctorS implements DoctorIF {

  private int       id;       /* Identificador unívoco del Doctor           */
  private AcademiaS academia; /* Academia a la que pertenece el Doctor      */

  /* Consulta el director de Tesis del doctor                                */
  /* @returns el Doctor que fue su director de Tesis                         */
  /*          null en caso de que sea el fundador de la Academia             */
  public DoctorIF getSupervisor();
```

```
  /* Inclúyase aquí todo lo que haga falta para una correcta implementación. */
  ...
}
```

Para ambos atributos se implementarán los getters correspondientes. En un principio, dado que esos datos no se verán modificados tras la creación del nuevo Doctor, se puede delegar en el constructor de la clase la tarea de instanciarlos al ser creado o bien implementar setters.

Para ver un ejemplo de este escenario supongamos que los números `1…7` representan los objetos de la clase **`DoctorS`** y que la información sobre la cual se ha construido la Academia es la siguiente:

- La Academia fue fundada por 1.
- La Tesis de 2 fue dirigida por 1.
- La Tesis de 3 fue dirigida por 2.
- La Tesis de 4 fue dirigida por 2.
- La Tesis de 5 fue dirigida por 4.
- La Tesis de 6 fue dirigida por 2.
- La Tesis de 7 fue dirigida por 3.

Las siguientes llamadas producirían el resultado indicado:

- `1.getSupervisor()` → **`null`** `(es el fundador de la Academia)`
- `6.getSupervisor()` → `2`
- `7.getAncestors(2)` → `[2, 3]`
- `7.getAncestors(3)` → `[1, 2 ,3]`
- `7.getAncestors(4)` → `[1, 2 ,3] (no hay más generaciones)`
- `1.getStudents()` → `[2]`
- `2.getStudents()` → `[3, 4, 6]`
- `1.getDescendants(2)` → `[2, 3, 4, 6]`
- `3.getSiblings()` → `[4, 6]`

Preguntas teóricas (trabajo del estudiante).

Las siguientes preguntas son previas al trabajo de implementación y deberán ser respondidas por el estudiante e incluidas en la memoria de práctica para su evaluación.

1.1 La clase **`AcademiaS`** implementa la interfaz **`AcademiaIF`** que, a su vez, extiende la interfaz **`CollectionIF<DoctorIF>`**. De los Tipos Abstractos de Datos estudiados en la asignatura, ¿cuáles se pueden utilizar para representar las relaciones de dirección de tesis doctorales? ¿Cómo deberían utilizarse esos TAD?

1.2 Entre los TAD considerados en la pregunta anterior, escoja razonadamente uno y explique cómo funcionaría cada uno de los métodos detallados en la interfaz **`DoctorIF`** y el método **`getSupervisor()`** de la clase **`DoctorS`**. ¿Cuál sería su coste asintótico temporal en el caso peor? (*Sería un buen ejercicio hacerlo para cada uno de los TAD considerados en 1.1*)

1.3 Ateniéndonos a este escenario, para cada uno de los cuatro métodos descritos en la interfaz **`DoctorIF`**, ¿pueden aparecer repeticiones? Justifique su respuesta.

2.3 Segundo escenario: Academia con Árbol Genealógico Completo.

En el primer escenario asumimos que una Tesis Doctoral sólo puede ser dirigida por un único Doctor, pero en la práctica ocurre con frecuencia que haya más de un director (denominados codirectores) para una misma Tesis Doctoral.
En este segundo escenario eliminaremos esa restricción. Para lo cual consideraremos una clase `AcademiaC` que implementará la interfaz `AcademiaIF` y que permita almacenar y gestionar la información de una Academia con Árbol Genealógico Completo.

Nótese que, cuando se permiten codirectores, un académico puede asociarse a más de una generación. En la figura de la página 3 se puede ver el caso de *Jan Van Mill*, que sería Descendiente Académico de *Johannes de Groot* de segunda generación por parte de su codirector *Pieter Cornelis Baayen* y de tercera generación por parte de su codirector *Eric van Douwen*.

Análogamente, existirá una clase `DoctorC` que implemente la interfaz `DoctorIF` y que permita gestionar la información de un Doctor dentro de una Academia con Árbol Genealógico Completo. A continuación se detallan los atributos y nuevos métodos que deberá tener dicha clase:

```
public class DoctorC implements DoctorIF {

  private int                    id;       /* Identificador unívoco          */
  private AcademiaC              academia; /* Academia a la que pertenece    */
  private CollectionIF<DoctorIF> students; /* A quienes ha dirigido la tesis */

  /* Consulta los directores de Tesis del doctor.                            */
  /* @returns una lista con los Doctores que fueron sus directores de Tesis. */
  public CollectionIF<DoctorIF> getSupervisors();

  /* Inclúyase aquí todo lo que haga falta para una correcta implementación. */
  ...
}
```

Al igual que en el primer escenario, se deberán implementar los getters correspondientes a todos sus atributos e implementaremos setters o, para aquellos atributos que no se modifican durante el tiempo de vida del objeto **`DoctorC`** (identificador y Academia), delegaremos en el constructor de la clase la tarea de instanciarlos.

Veamos un ejemplo que ilustre este escenario. En esta ocasión, los números `1…7` representan los objetos de la clase **`DoctorC`** y la información sobre la cual se ha construido la Academia es la siguiente:

- La Academia fue fundada por 1.
- La Tesis de 2 fue dirigida por 1.
- La Tesis de 3 fue codirigida por 1 y 2.
- La Tesis de 4 fue dirigida por 2.
- La Tesis de 5 fue codirigida por 1 y 3.
- La Tesis de 6 fue codirigida por 4 y 5.
- La Tesis de 7 fue dirigida por 3.

Entonces, las siguientes llamadas producirían el resultado indicado (recuerde que el orden en las colecciones no es relevante):

- `1.getSupervisors()` → `[] (es el fundador de la Academia)`
- `6.getSupervisors()` → `[4, 5]`
- `7.getAncestors(2)` → `[1, 2 ,3]`
- `7.getAncestors(3)` → `[1, 2 ,3] (no hay más generaciones)`
- `1.getStudents()` → `[2, 3, 5]`
- `2.getStudents()` → `[3, 4]`
- `1.getDescendants(2)` → `[2, 3, 4, 5, 6, 7]`
- `3.getSiblings()` → `[2, 4, 5]`

Preguntas teóricas (trabajo del estudiante).

2.1 Explique cómo cambian las relaciones de supervisión en este segundo escenario. A la hora de representar el problema, ¿puede aplicarse directamente el mismo TAD utilizado en el primer escenario? ¿Por qué?

2.2 Razone, sin realizar ningún tipo de implementación, cómo funcionarían en este escenario los métodos de la interfaz **`DoctorIF`** y el método **`getSupervisors()`** de la clase **`DoctorC`** suponiendo que la clase **`AcademiaC`** se implemente utilizando una lista de objetos de la clase **`DoctorC`**.

2.3 Basándose en la respuesta a la pregunta anterior, supongamos que se añade a la clase **`DoctorC`** un nuevo atributo **`supervisors`** con la colección de los doctores que dirigieron su tesis doctoral (y, en consecuencia, el método **`getSupervisors()`** se convierte en un getter de dicho atributo). ¿Cómo afectaría esto al espacio necesario para almacenar el Árbol Genealógico de la Academia en memoria? ¿Y cómo afectaría a la programación y al tiempo de ejecución de los métodos?

2.4 Supongamos que la lista de objetos de la clase **`DoctorC`** referida en la pregunta 2.2 se encuentra siempre ordenada según el atributo **`id`**. ¿Cómo se podría aprovechar en la implementación de los métodos para mejorar el coste temporal de las operaciones de consulta? ¿Cómo afectaría a la operación de modificación?

2.5 Supongamos que levantamos la restricción de que en una Academia sólo exista un Doctor sin supervisores dentro de la Academia y permitimos que haya varios Doctores en esa situación. ¿Habría que hacer cambios en la implementación? ¿Cuáles y por qué?

2.6 Ateniéndonos a este escenario, para cada uno de los cuatro métodos descritos en la interfaz **`DoctorIF`** y para el método **`getSupervisors()`** de la clase **`DoctorC`**, ¿pueden aparecer repeticiones? Justifique su respuesta.

3. Implementación.

Se deberá realizar un diseño en Java que contenga las siguientes clases:

- Clases **`AcademiaS`** y **`DoctorS`** que implementarán, respectivamente, las interfaces **`AcademiaIF`** y **`DoctorIF`**, permitiendo así gestionar una Academia con Árbol Genealógico Simplificado (ver apartado 2.2).
- Clases **`AcademiaC`** y **`DoctorC`**, que implementarán, respectivamente, las interfaces **`AcademiaIF`** y **`DoctorIF`**, permitiendo así gestionar una Academia con Árbol Genealógico Completo (ver apartado 2.3 incluyendo las preguntas 2.2 a 2.6).

Estas cuatro clases deberán implementarse en un único paquete llamado:

es.uned.lsi.eped.pract2016

Para la implementación de las estructuras de datos **se deberán utilizar las interfaces proporcionadas por el Equipo Docente** de la asignatura. La implementación de los mecanismos de Entrada/Salida y el Estudio Empírico del Coste será proporcionada por el Equipo Docente a través del Curso Virtual.

3.1 Parámetros de entrada.

El programa recibirá tres parámetros de entrada que determinarán su comportamiento. El orden y significado de los parámetros será el siguiente:

1. Selección del escenario que se desea utilizar, que tendrá únicamente dos valores válidos:
   - S para el escenario 1 (Academia con Árbol Genealógico Simplificado).
   - C para el escenario 2 (Academia con Árbol Genealógico Completo).
2. Fichero de datos, que contendrá el nombre del fichero de datos con el que se construirá la Academia. Nótese que (debido a la definición de ambos escenarios) los ficheros de datos para el escenario 2 no son compatibles con el escenario 1 (pero sí a la inversa).
3. Fichero de operaciones, que contendrá el nombre del fichero de operaciones que se desean realizar sobre los datos contenidos en la Academia.

3.2 Estructura del fichero de datos.

El fichero de datos contendrá todos los datos necesarios para construir la Academia, es decir, quién la fundó y el histórico de dirección de tesis doctorales. Con respecto a este histórico, cuando se afirme que A ha dirigido a B se entenderá que el doctor A ya pertenecía a la Academia..

Toda la información referente a la dirección de una Tesis se encontrará en una misma línea. A continuación podemos ver un ejemplo de fichero de datos para el primer escenario que se corresponde con el ejemplo de la sección 2.2 de este enunciado:

```
La Academia fue fundada por “1”
Tesis de “2” dirigida por “1”
Tesis de “3” dirigida por “2”
Tesis de “4” dirigida por “2”
Tesis de “5” dirigida por “4”
Tesis de “6” dirigida por “2”
Tesis de “7” dirigida por “3”
```

La acción esperada por el programa será construir la Academia con Árbol Genealógico Académico Simplificado (utilizando para ello un objeto de la clase **`AcademiaS`**).

Un ejemplo de fichero de datos para el segundo escenario (basado en el ejemplo de la sección 2.3) sería el siguiente:

```
La Academia fue Fundada por “1”
Tesis de “2” dirigida por “1”
Tesis de “3” codirigida por “1” y “2”
Tesis de “4” dirigida por “2”
Tesis de “5” codirigida por “1” y “3”
Tesis de “6” codirigida por “4” y “5”
Tesis de “7” dirigida por “3”
```

La acción esperada por el programa será construir la Academia con Árbol Genealógico Académico Completo (utilizando para ello un objeto de la clase **`AcademiaC`**).

3.3 Estructura del fichero de operaciones.

El fichero de operaciones contendrá una operación por línea. Existirán cinco tipos de operaciones de consulta (una por cada método de consulta que existe en la interfaz **`DoctorIF`** más el método **`getSupervisor()`** de la clase **`DoctorC`** o **`getSupervisors()`** de la clase **`DoctorC`**), que se identificarán por los dos primeros caracteres de la línea:

1. Operación de consulta de director(es) de tesis. Los dos primeros caracteres de la línea serán `"SU"`, a continuación un espacio y luego el identificador del doctor del cual se desea conocer su(s) director(es) de tesis encerrado entre comillas. Por ejemplo:

```
SU "3"
```

Esta operación de consulta deberá devolver:

```
Tesis de "3" dirigida por "2"                  (para el primer escenario)
Tesis de "3" codirigida por "1" y "2"          (para el segundo escenario)
```

2. Operación de consulta de ancestros académicos. Los dos primeros caracteres de la línea serán `"AN"`, a continuación un espacio, el nombre del doctor del cual se desea conocer sus ancestros académicos encerrado entre comillas y el número de generaciones a considerar. Por ejemplo:

```
AN "4" 2
```

Cuyo resultado, en ambos ejemplos, sería:

```
Los ancestros de "4" hasta 2 generaciones son "1" y "2"
```

3. Operación de consulta de estudiantes. Los dos primeros caracteres de la línea serán `"ST"`, a continuación un espacio y el nombre del doctor del cual se desea conocer aquellos estudiantes a quienes ha dirigido sus Tesis Doctorales encerrado entre comillas. Por ejemplo:

```
ST "3"
```

Cuyo resultado deberá ser:

```
"3" ha dirigido la Tesis de "7"                (para el primer escenario)
"3" ha dirigido las Tesis de "5" y "7"         (para el segundo escenario)
```

4. Operación de consulta de descendientes académicos. Los dos primeros caracteres de la línea serán `"DE"`, a continuación un espacio, el nombre del doctor del cual se desea conocer sus descendientes académicos encerrado entre comillas y el número de generaciones a considerar. Por ejemplo:

```
DE "2" 2
```

Deberá devolver, para ambos escenarios, lo siguiente:

```
Los descendientes de “2” hasta 2 generaciones son “3”, “4”, “5”, “6” y “7”
```

5. Operación de consulta de hermanos académicos. Los dos primeros caracteres de la línea serán `“SI”`, a continuación un espacio y el nombre del doctor del cual se desea conocer sus hermanos académicos encerrado entre comillas. Por ejemplo:

```
SI “3”
```

Que deberá devolver:

```
Los hermanos de “3” son “4” y “6”              (para el primer escenario)
Los hermanos de “3” son “2”, “4” y “5”         (para el segundo escenario)
```

Hay que recordar que el orden de las coleccions no es relevante, por lo que las respuestas aquí mostradas han de tomarse como una posible de entre todas las correctas.

3.4 Medición de tiempos.

Se desea realizar un estudio empírico del coste que emplean para su ejecución los métodos de consulta indicados en la interfaz **`DoctorIF`** y los métodos **`getSupervisor()`** y **`getSupervisors()`** de las clases **`DoctorS`** y **`DoctorC`** respectivamente.

La implementación de los métodos necesarios para realizar este estudio será proporcionada por el Equipo Docente.

Estudio empírico del coste temporal (trabajo del estudiante)

3.1 Realice un estudio empírico del coste temporal de los métodos de consulta de la clase **`DoctorS`** en el que se mida cómo varía el coste según el número de doctores que haya en la Academia. Compárelo con el estudio teórico realizado en la pregunta 1.2.

3.2 Realice un estudio empírico del coste temporal de los métodos de consulta de la clase **`DoctorC`** en el que se mida cómo varía el coste según el número de doctores que haya en la Academia. Compárelo con el estudio teórico realizado en la pregunta 2.4.

3.5 Salida del programa.

La salida del programa se realizará por la salida estándar de Java y consistirá en:

- La primera línea indicará cuántos doctores diferentes forman la Academia.
- Para cada operación presente en el fichero de operaciones se indicará el resultado (ver punto 3.3) y, en la línea siguiente, se indicará su coste.

Si consideramos las consultas utilizadas en el apartado 3.3 de este enunciado, la salida esperada para el primer escenario (y teniendo en cuenta que el coste indicado es ficticio a modo de ejemplo y que el orden en las colecciones no es relevante) sería:

```
Doctores en la Academia: 7.
Tesis de “3” dirigida por “2”
-Tiempo: 6
Los ancestros de “4” hasta 2 generaciones son “1” y “2”
-Tiempo: 20
“3” ha dirigido la Tesis de “7”
-Tiempo: 10
```

```
Los descendientes de “2” hasta 2 generaciones son “3”, “4”, “5”, “6” y “7”
-Tiempo: 25
Los hermanos de “3” son “4” y “6”
-Tiempo: 11
```

Por otro lado, la salida esperada para el segundo escenario sería:

```
Doctores en la Academia: 7.
Tesis de “3” codirigida por “1” y “2”
-Tiempo: 6
Los ancestros de “4” hasta 2 generaciones son “1” y “2”
-Tiempo: 20
“3” ha dirigido las Tesis de “5”, y “7”
-Tiempo: 10
Los descendientes de “2” hasta 2 generaciones son “3”, “4”, “5”, “6” y “7”
-Tiempo: 25
Los hermanos de “3” son “2”, “4” y “5”
-Tiempo: 11
```

4. Ejecución y juegos de prueba.

Para la ejecución del programa se deberá abrir una consola y ejecutar:

```
java -jar eped2016.jar <escenario> <datos> <operaciones>
```

- `<escenario>` selección del escenario a considerar (`S` ó `C`).
- `<datos>` fichero de datos para construir la Academia.
- `<operaciones>` fichero de operaciones con consultas sobre los Doctores.

El Equipo Docente proporcionará, a través del curso virtual, juegos de prueba consistentes en un fichero de datos, un fichero de operaciones y una salida esperada para comprobar el correcto funcionamiento del programa.

Para la realización del estudio empírico del coste, los estudiantes podrán crear diferentes ficheros de datos con un número diferente de doctores, de manera que se pueda medir cómo varía el tiempo que emplean las operaciones según dicho número.

5. Documentación y plazos de entrega.

La práctica supone un 20% de la calificación de la asignatura, y es necesario aprobarla (5 puntos) para superar la asignatura. Además será necesario obtener, al menos, un 4 sobre 10 en el examen presencial para que la calificación de la práctica sea tenida en cuenta de cara a la calificación final de la asignatura.

Los estudiantes deberán asistir a una sesión obligatoria de prácticas con su tutor en el Centro Asociado. Estas sesiones son organizadas por los Centros Asociados teniendo en cuenta sus recursos y el número de estudiantes matriculados, por lo que en cada Centro las fechas serán diferentes. Los estudiantes deberán, por tanto, dirigirse a su tutor para conocer las fechas de celebración de estas sesiones.

De igual modo, el plazo y forma de entrega son establecidos por los tutores de forma independiente en cada Centro Asociado, por lo que deberán ser consultados también con ellos.

La documentación que debe entregar cada estudiante consiste en:

- Memoria de práctica, en la que se deberán responder a las preguntas teóricas e incluir el estudio empírico del coste detallado en este documento.
- Implementación en Java de la práctica, de la cual se deberá aportar tanto el código fuente como el programa compilado.