Tests de caja negra

Se han implementado un total de 17 tests (3 de ellos happy paths). A continuación, se describen en profundidad:

- **testForEmptyGraph()**:
  - Clases cubiertas: `Graph.java` e `In.java`.
  - Se basa en la premisa de que al invocar al constructor de Graph, se intentará leer de un filename (en nuestro caso). Si este fichero es inválido (null en este caso), en la clase `In.java` se elevará la excepción `IllegalArgumentException`. Comprobamos así si salta esa excepción. La única caracterización sería filename válido o no (true o false).

- **testForNoDelimiter()**:
  - Es muy parecido al anterior, pero teniendo en cuenta ahora el argumento *delimiter* (fijado a null en este ejemplo). De nuevo, la única caracterización sería *delimiter* válido o no (true o false).

- **testForNoFile()**:
  - Clases cubiertas: `Graph.java` e `In.java`, nuevamente.
  - Al llamar al constructor de Graph, se intentará leer de un **filename** (en nuestro caso). Si este fichero es inválido (no existe), en la clase `In.java` se elevará la excepción `IllegalArgumentException`. Comprobamos así si salta esa excepción. La única caracterización sería filename existente o no (true o false).

- **testForIncompatibleTypes()**:
  - Clase cubierta: `PathFinder.java`.
  - Comprobamos si salta la excepción oportuna cuando se llama al constructor de Pathfinder(graph,name), al intentar añadir en su atributo **dist** (del tipo ST) el string "" (name). Queda explicado en el propio `GraphFuncionalityTest.java`. La única caracterización sería **name** válido o no (true o false).

- **testForSizeZeroGraph()**:

    - Clase cubierta: `GraphFuncionality.java`.
    - Métodos cubiertos: `doDistance()`, `doGraphFilter()` o `doRanking()`, ya que todos tienen como entrada un argumento de tipo Graph.
    - Se basa en la premisa de que un grafo con 0 vértices no es válido (posible **filename** erróneo). Utilizaremos un fichero vacío como argumento para el constructor de Graph, y un delimitador correcto. El grafo creado tendrá 0 vértices, y se elevará la excepción. La única caracterización sería **graph** válido o no (true o false).

- **testForCloseConnection()**:

    - Clase cubierta: `GraphFuncionality.java`.
    - Método cubierto: `nameChecker(Connection conn, String name)`.
    - Tratamos el caso de que si no introducimos un Connection correcto (porque está cerrado), se eleve la excepción correspondiente. La única caracterización sería **conn** válido o no (cerrado o abierto, true o false).

- **testForInvalidName()**:

    - Clase cubierta: `GraphFuncionality.java`.
    - Método cubierto: `nameChecker(Connection conn, String name)`.
    - Test para comprobar que salta la excepción si por lo que sea a `nameChecker()` le llega un name="" (inválido). Es muy poco probable, ya que en `doDistance()` y `doGraphFilter()` comprobamos si sus strings de entrada no son "". Aún así, por si "la liamos" y cambiamos **name**, estamos cubiertos en este caso. La única caracterización es **name** válido o no (true o false).

- **testForInvalidName2()**:

    - Clase cubierta: `GraphFuncionality.java`.
    - Método cubierto: `doRanking(Graph g, String number)`.
    - Análogo al anterior, pero para otro método. Comprueba que salta la excepción si por lo que sea al método le llega un number="" (inválido). La única caracterización es **number** válido o no (true o false).

Y ahora, los 3 happy paths:

- **validGraph()**: test para comprobar que se crea un grafo correctamente si *filename* y *delimiter* son correctos.
- **validPathFinder()**: test para comprobar que se crea un pathfinder correcto (con ruta) dado dos nombres relacionados.
- **validNameChecker()**: test para comprobar si `nameChecker()` retorna un ArrayList válido dado un nombre válido.

Se han dejado comentados otros tests. No se han incluido porque hacen referencia a los distintos `IllegalArgumentException` lanzados por los métodos `doDistance()` y `doGraphFilter()`, ya que son tratados con catch (son Checked Exceptions).

- **Método String relatedMovies2(Graph graph, String movie)**

  i. Basado en la interfaz (dominio de la entrada):
     - En cuanto a `graph`, en la primera línea de código se comprueba si grafo es válido (es decir, tiene vértices) o no. En caso de no ser válido se elevaría una excepción de tipo `NullPointerException`. --> `testForRelatedMovies2_InvalidGraph()`
     - En cuanto a `movie`, al ser un String se nos podría ocurrir dos alternativas: String vacío y String normal. Bien, el funcionamiento con String vacío no se podría testar en este método ya que en el `Main` comprobamos si eso sucede, y en tal caso nos devolvería un String con un mensaje HTML, pero no haría una llamada a este método. Por lo que sólo podremos testear cuando `movie` es un String cualquiera. Pero esto carece de interés, así que iremos más allá en el test basado en la funcionalidad.

  ii. Basado en la funcionalidad:
     - Aquí ya sí, debido a nuestro conocimiento sobre el método, podemos testear y comprobar comportamientos del método más interesantes:
       - `movie` puede ser vértice del grafo o puede no serlo. (No tests ya que método devolvería en ambos casos un String).
       - `movie`, siendo vértice del grafo, puede ser actriz/actor o puede ser película. Hemos hecho este método de tal manera que, obedeciendo al `Main`, solo se ejecutará si el String es vértice y película. Si es vértice pero es actriz/actor, `Main` devolverá un String con mensaje HTML indicando que lo que se ha introducido no corresponde con una película. Aún así, en el método hemos puesto una línea comprobando que es película y, si no lo fuera, elevaría una excepción de tipo `IllegalArgumentException`. --> `testForRelatedMovies2_NotAMovie()`

- **Método String show_items(ArrayList items)**

  i. Este método solo tiene sentido testearlo basándose en su interfaz, ya que recibe como argumento un ArrayList de Strings:
     - Este ArrayList de Strings, al que hemos llamado `items` puede estar vacío, o puede no estar vacío. (No tests ya que método devolvería en ambos casos un String).

- **Método ArrayList nameCheckerMovie(Connection conn, String movie_name)**

  i. Este método, cuyo funcionamiento se basa en recibir un String y devolver un ArrayList de Strings, no merece la pena ser testeado en base a la interfaz (parámetros de entrada), ya que el String puede ser vacío o no vacío. Pero, en caso de ser vacío el

nombre que se introduce el formulario, es el propio `Main` el que actúa y el que devuelve un String con el HTML pertinente. Es por ello por lo que no habría invocación al método `relatedMovies2()`, y éste, evidentemente, al no ser invocado, no puede invocar al método `nameCheckerMovie()`. (Es importante recordar que este método solo se ejecutará siendo invocado por el método `relatedMovies2()`, cuando el String introducido no es un vértice del grafo). De esta manera, ese método sólo recibirá Strings distintos al String vacío. Esto será más interesante de abordar en el testing basado en la funcionalidad.

ii. Basado en la funcionalidad:

- String puede ser vértice del grafo o puede no serlo. Este método solo será ejecutado, obedeciendo al método `relatedMovies2()`, si no es un vértice del grafo, ya que, de no ser así, este método `nameCheckerMovie()` no sería ejecutado. --> `testForNameCheckerMovie()`.

Si forzamos a que este método reciba un String que sea vértice (y película), lo que sucedería sería que el único nombre de película que nos propondría sería el propio String introducido. Es decir, el ArrayList devuelto tendría solo un elemento. A no ser que dicho vértice fuera una subcadena de caracteres de otro vértice (p. ej. Spider-Man y Spider-Man2). Por ello, para este caso, probaremos que el ArrayList devuelto contiene el vértice introducido. --> `testForNameCheckerMovie_IsAVertex()`

- Nuestros métodos **String relatedActors(Graph graph, String actor)** , **ArrayList nameCheckerActor(Connection conn, String actor_name)** y **ArrayList nameChecker(Connection conn, String name)**, quedarían testados una vez lo fueran los anteriores, ya que poseen un funcionamiento similar.

Tests de caja blanca

Dados los grafos de cada método, se expondrán las rutas seguidas por cada test (que cubra la clase `GraphFuncionality.java`):

- **testForSizeZeroGraph()**: utilizamos `doDistance()` como método de ejemplo (misma ruta en `doGraphFilter()`). Ruta: [1,2].
- **testForCloseConnection()**: método `nameChecker()`. Ruta: [1,3,10].
- **testForInvalidName()**: método `nameChecker()`. Ruta: [1,2].
- **testForInvalidName2()**: método `doRanking()`. Ruta: [1,2].
- **validGraph()**: `Main.java`. No tenemos su grafo.
- **validPathFinder()**:
- **validNameChecker()**:
- **testForRelatedMovies2_InvalidGraph()**: Mirando el grafo `relatedMovies2.png`, ruta:

[1,2,3].

- **testForRelatedMovies2_NotAMovie()**: Mirando el grafo `relatedMovies2.png`, ruta: [1,2,4,5,6,8].
- **testForNameCheckerMovie()**: Mirando el grafo `relatedMovies2.png`, ruta: [1,2,4,5,7,14,16,17] para que se ejecute el método `relatedMovies2()` que invocará al método `nameCheckerMovie()`, el cual seguirá la ruta: [1,2,4,5,6,7,7,7,7,7,7,8].
- **testForNameCheckerMovie_IsAVertex()**: Igual que el anterior.