Programación II

Curso 24-25

Práctica 3

0 Cuestiones previas

En esta práctica crearemos clases para desarrollar un centro de coworking, que usará las clases definidas en la práctica anterior para modelar los cotrabajadores. Crea el directorio `p3` dentro del directorio `Practicas`, y copia a él el fichero *coworkers.txt* y las clases de los cotrabajadores de la práctica 2, actualizándolas para que pertenezcan al paquete `p3`. Crea la clase `P3`, y copia a ella los arrays de cotrabajadores de la `P2` y el método `readCoworkersFile`.

1 Actividad 1: crear las clases del centro de coworking

Crea las siguientes clases, con sus atributos, constructores, y *getters*/*setters*:

- **`OfficeLocator`**: representa el localizador de una oficina del centro de coworking, con tres atributos privados:
  - **floor**: un número entero (*short*), que representa la planta de la oficina (entre 0 y 9).
  - **corridor**: un número entero (*short*), que representa el pasillo (entre 0 y 9).
  - **num**: un entero (*short*), que representa el número de la oficina en el pasillo (entre 0 y 5)

  Añade a esta clase los siguientes métodos:

  - `public boolean equals (OfficeLocator anotherOfficeLocator) {...}`

    Este método debe comprobar si el localizador sobre el que se invoca es igual a otro (`anotherOfficeLocator`) y devolver `true/false` según sea el caso, es decir, si coinciden los valores de sus tres atributos.

  - `public String toString () {...}`

    Devuelve una cadena con el localizador, siguiendo el formato "*floor:corridor:num*".

- Clase **`Office`**: representa una oficina del centro, con los siguientes atributos privados:
  - **officeLocator**: un objeto de la clase `OfficeLocator`, que indica la localización de la oficina.
  - **idCoworker**: Una cadena de tamaño fijo 9, que representa el *id* (dni) del cotrabajador que ocupa la oficina. Si la oficina está libre, este campo se pondrá a `null`.

  Añade a esta clase su correspondiente método `equals` (que comprueba sólo que sus objetos `OfficeLocator` sean iguales, delegando en el `equals` de `OfficeLocator`), y el siguiente método:

  - `public String toString () {...}`

    devuelve una cadena "*officeLocator;idCoworker*" (p.ej. "0:2:4;11114444B"). Si la oficina está vacía, devolverá sólo el localizador ("0:2:4"), el cual se obtendrá siempre a partir del `toString` de la clase `OfficeLocator`.

- Clase **`CoworkingCenter`**, para representar la estructura y contenido del centro de coworking, y para realizar la gestión de las entradas y salidas de los cotrabajadores. Contiene los siguientes atributos privados:
  - **sizeFloors**: un entero (*short*) para indicar el número de plantas del centro.
  - **sizeCorridors**: un entero (*short*) para indicar el número de pasillos del centro.
  - **sizeNums**: un entero (*short*) para indicar cuántos números hay por pasillo (el mismo para todos).

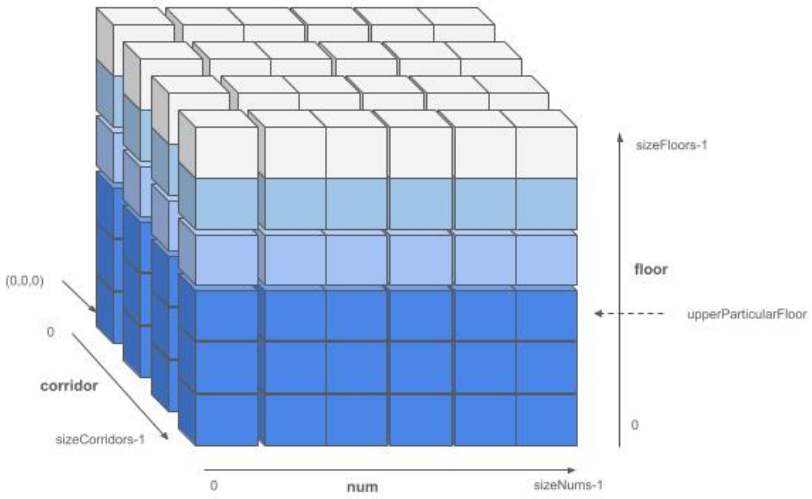
    Si, por ejemplo, **sizeFloors**=3, **sizeCorridors**=4 y **sizeNums**=6, y el centro tiene 6x4x3=72 oficinas (0:0:0, 0:0:1, ..., 2:3:5).

  - **upperParticularFloor**: un entero (short) para indicar la planta más alta para trabajadores particulares. Este número permitirá asignar las oficinas según las reglas siguientes:

- Desde la planta 0 a la planta **upperParticularFloor** para cualquier cotrabajador.
- Desde la planta **upperParticularFloor+1** a la planta **sizeFloors-2,** oficinas reservadas para ejecutivos y empleados (objetos `Executive` y `Employee`)
- La planta superior (**sizeFloors-1)** es solo para ejecutivos (objetos `Executive`).

- **offices**: un array tridimensional de oficinas (objetos de la clase `Office`), cuyas tres dimensiones son las correspondientes a los atributos **sizeFloors, sizeCorridors** y **sizeNums**.

El siguiente gráfico muestra cómo sería el centro de coworking teniendo en cuenta la información anterior.



2 Actividad 2: crear y guardar el centro de coworking

En esta actividad definiremos el formato de un fichero que almacena el estado inicial del centro, e introduciremos los métodos para crear un centro de coworking a partir de un fichero, y para guardarlo a un fichero.

Recuerda que, dado que no capturas las excepciones que se pueden producir en el manejo de los ficheros, debes declararlas en la definición de los métodos (con el correspondiente `throws`).

2.1 Formato de un fichero del centro de coworking

El fichero contiene una línea con las dimensiones del centro, y un conjunto de líneas representando las oficinas ocupadas (una oficina en cada línea).

La primera línea especificará el número de plantas (**sizeFloors**), el número de pasillos (**sizeCorridors**), la cantidad de números por pasillo (**sizeNums**), y el número de la primera planta que pueden ocupar los cotrabajadores empleados (**upperParticularFloor**), según el formato siguiente:

```
sizeFloors;sizeCorridors;sizeNums;upperParticularFloor
```

Las siguientes líneas del fichero, que identifican cada oficina ocupada del centro, consistirán en el localizador de la oficina y el ID del cotrabajador que la ocupa. Puede haber líneas con comentarios (su primer carácter distinto de un espacio es '#'). Un ejemplo de este fichero podría ser:

```
     # 6 plantas, con 5 pasillos, cada una con 5 números
# la planta más alta para Particulares es la 3
6;5;5;3
# oficinas ocupadas
0:1:0;11114444A
5:3:1;22225555B
2:4:1;33336666C
```

2.2 Crear un centro de coworking a partir de un fichero

Para crear el centro a partir del fichero, añade el siguiente constructor a la clase `CoworkingCenter`:

```
public CoworkingCenter(String fileName) {...}
```

donde `fileName` indicará el nombre del fichero del centro. El código de este constructor tiene que:

1. Abrir el fichero, leer la línea con las dimensiones del centro, e inicializar los atributos de la clase. Debe crear el array tridimensional de oficinas (**offices**) según las dimensiones leídas, e inicializarlo con oficinas (objetos de la clase `Office`) inicialmente vacías, con su localizador correspondiente y el `idCoworker` a `null`.
2. Leer el resto de líneas del fichero, que representan las oficinas ocupadas. Para cada oficina, debe construir un objeto `OfficeLocator` con el localizador de la oficina ocupada, y recorrer luego el array **offices** buscando la oficina (objeto de la clase `Office`) correspondiente a ese localizador. Al encontrarlo, debe actualizar el `idCoworker` de esa oficina para reflejar que está ocupada por el cotrabajador correspondiente.

   Nota: usa el método `equals` de la clase `OfficeLocator` (o el de la clase `Office`) para comparar cada oficina leída del fichero con las oficinas del array **offices**.

2.3 Almacenar un centro de coworking en un fichero

Para almacenar en un fichero el estado del centro, debes crear el siguiente método en la clase `CoworkingCenter`:

```
public void saveCoworkingCenterToFile (String fileName) {...}
```

Este método creará el fichero y guardará en la primera línea las dimensiones del centro, y, a continuación, la información sobre todas las oficinas ocupadas, una en cada línea, de acuerdo al formato descrito anteriormente. Las oficinas ocupadas se guardarán de forma ordenada (plantas de mayor a menor, y, dentro de cada planta, primero el pasillo 0, y, para cada pasillo, primero la número 0).

Pista: fíjate en que la información que hay que guardar sobre cada oficina ocupada es exactamente la que devuelve el método `toString` de la clase `Office`.

NOTA: El fichero creado no debe contener ninguna línea de comentarios.

3 Actividad 3. Entradas y salidas del centro de coworking

3.1 Especificación del fichero de entradas y salidas

La secuencia de las entradas y salidas del centro se leerá de un fichero, que contendrá una o más líneas con el siguiente formato (además de comentarios en cualquier línea):

```
<typeIO>;<id>
```

- **typeIO*:*** un carácter que indica si es una entrada ('I') o una salida ('O').
- **id**: el ID del cotrabajador que entra/sale.

Un ejemplo, con 3 entradas y una salida, sería:

```
I;11114444A
I;22225555B
I;33336666C
O;33336666C
```

Para obtener el tipo de cotrabajador, crea el método:

```
char getCoworkerTypeFromId (String id) {...}
```

que busque en los 3 arrays el `id` del cotrabajador (el argumento), y retorne el tipo de cotrabajador asociado.

3.2 Entradas al centro

La asignación de oficina a cada trabajador que entra se rige por la siguiente norma:

1. Asignación de planta.

   Si es un cotrabajador particular, se comenzará a buscar por la planta `upperParticularFloor`, descendiendo hasta la planta 0.

   Si es un ejecutivo, se comenzará por la planta `sizeFloors` - 1, hasta la 0.

   En otro caso, se comenzará por la planta `sizeFloors` - 2, `hasta la planta 0.`

2. Asignación de oficina por planta.

   Se comenzará a buscar por el pasillo 0, hasta el último. Dentro de cada pasillo, se comenzará por num=0, hasta el último, hasta encontrar una oficina libre (**no contemplaremos la posibilidad de que no haya una oficina libre).**

Para implementar la entrada de un cotrabajador, crea en la clase `CoworkingCenter` el siguiente método:

```
public void coworkerIn(String id, char coworkerType) {...}
```

donde *id* es el ID del cotrabajador que entra, y *coworkerType* el tipo de cotrabajador.

Este método debe acceder al array de oficinas **offices**, y localizar la primera oficina libre (según el tipo de cotrabajador y el procedimiento anteriormente descrito). Debe almacenar el *id* del cotrabajador en el campo correspondiente de la oficina, marcándola así como ocupada.

3.3 Salidas del centro

Para implementar la salida de un cotrabajador, crea en la clase `CoworkingCenter` el siguiente método:

```
public void coworkerOut(String idCoworker) {...}
```

Este método debe buscar en el array **offices** la oficina que ocupaba el trabajador. Tras encontrarla, debe liberar la oficina, poniendo su campo *idCoworker* a `null`.

3.4 Número de oficinas ocupadas

Añade a la clase CoWorkingCenter el método:

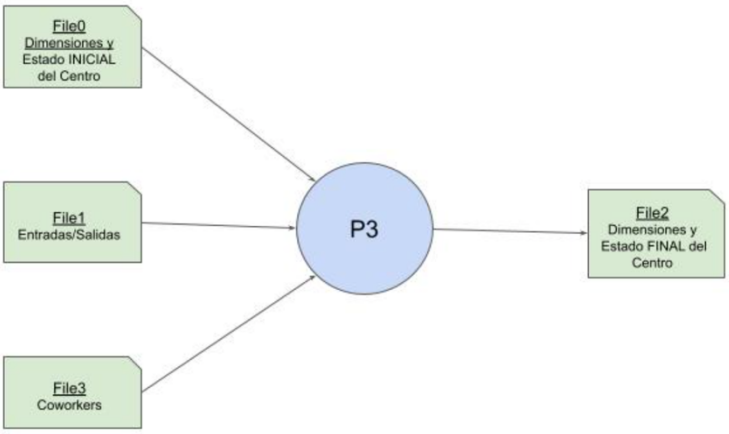
```
public int getNumCoworkers ()
```

que devuelva el número de oficinas ocupadas.

4 Actividad 4: Crear la clase principal

Completa la clase principal P3 que:

1. Reciba y lea cuatro argumentos con los nombres de cuatro ficheros:
   - `file0` (un fichero existente con la estructura y contenido de un centro de coworking)
   - `file1` (un fichero existente de entradas y salidas)
   - `file2` (el nombre del fichero en el que guardar el centro tras actualizarlo)
   - `file3` (un fichero existente con los cotrabajadores)



2. Llama al método `readCoworkersFile()` para leer los cotrabajadores del fichero existente `file3`.
3. Crea un objeto `miCoworkingCenter` (una variable local) de la clase `CoworkingCenter` a partir del fichero existente `file0`, que contiene el estado de un centro.
4. Crea la variable `numCoworkers` de tipo entero, inicializándola con el número de cotrabajadores que hay en el centro, llamando al método `getNumCoworkers` de la clase `CoworkingCenter`. Imprime su valor por pantalla.
5. Añade, en esta clase, el método `processIO` para leer las entradas y salidas de un fichero y actualizar el centro de acuerdo a las mismas, según:

   ```
   public static void processIO (CoworkingCenter unCoworkingCenter,
                                 int numCoworkers, String filename)
   ```

   Este método deberá modificar el argumento de entrada `numCoworkers` según los trabajadores vayan entrando y/o saliendo del centro.
6. Invoca el método `processIO` pasándole `miCoworkingCenter`, el valor de la variable `numCoworkers` y el fichero existente `file1`.
7. Imprime por pantalla la línea:

   Valor de la variable `numCoworkers` = <VV>, número actual de cotrabajadores = <RR>

   donde <VV> es el valor de la variable `numCoworkers` y <RR> el valor del número real de trabajadores del centro, tomado del método `getNumCoworkers` de `miCoworkingCenter`.

   Fíjate que, tras invocar a `processIO`, `numCoworkers` no ha cambiado, pero `miCoworkingCenter` sí. Pregunta al profesor si tienes dudas de por qué funciona así.
8. Guarda el estado del centro en el fichero a crear `file2`, invocando `saveCoworkingCenterToFile`.

NOTA: En todos los casos, se puede suponer que los ficheros serán siempre correctos de acuerdo al formato correspondiente, por lo que no es necesario realizar ningún tipo de comprobación sobre su contenido.

5 Actividad 5. Representar gráficamente el centro

Representaremos el centro de coworking completo según indica la siguiente figura (para un centro [4;3;4;1]), en el orden de plantas descendente:

```
------------------------------------------------------------------------ FLOOR 3
X  | 3:0:0 11115555A | 3:0:1          | 3:0:2          | 3:0:3          |← pasillo 0
X  | 3:1:0           | 3:1:1          | 3:1:2          | 3:0:3          |
X  | 3:2:0           | 3:2:1          | 3:2:2          | 3:0:3          |
------------------------------------------------------------------------ FLOOR 2
XE | 2:0:0           | 2:0:1          | 2:0:2          | 2:0:3          |
XE | 2:1:0           | 2:1:1          | 2:1:2          | 2:1:3          |
XE | 2:2:0           | 2:2:1          | 2:2:2          | 2:2:3          |
------------------------------------------------------------------------ FLOOR 1
XEP| 1:0:0           | 1:0:1          | 1:0:2          | 1:0:3          |
XEP| 1:1:0           | 1:1:1          | 1:1:2          | 1:1:3          |
XEP| 1:2:0           | 1:2:1          | 1:2:2          | 1:2:3          |
------------------------------------------------------------------------ FLOOR 0
XEP| 0:0:0           | 0:0:1          | 0:0:2          | 0:0:3          |
XEP| 0:1:0           | 0:1:1          | 0:1:2          | 0:1:3          |
XEP| 0:2:0           | 0:2:1          | 0:2:2          | 0:2:3          |
------------------------------------------------------------------------
```

Para cada planta:

1. Primero se pone una línea de guiones, según indica la figura, terminada por un espacio en blanco y luego la palabra FLOOR seguida por el número de la planta que se va a dibujar a continuación. El número de guiones será 4 más 18 (lo que ocupa una oficina) multiplicado por el tamaño del pasillo (**sizeNums)**.
2. Luego, se crea una línea por cada pasillo (empezando por el pasillo 0 hasta el último). Para cada pasillo, en las 3 primeras posiciones se indica si la planta está reservada a ejecutivos ('X☐☐'), a ejecutivos y empleados ('XE☐') o vale para todos ('XEP'), donde ☐ indica un espacio en blanco. Luego un <SEPARADOR_VERTICAL>. Y a continuación van las oficinas de cada pasillo de forma consecutiva y en la misma línea (empezando por la de número 0 hasta la última), representando la información de cada oficina de la forma siguiente:

   ☐<Localizador>☐<Id>☐<SEPARADOR_VERTICAL>

   - <Localizador>: el localizador de la oficina.
   - <Id>: el ID del cotrabajador que ocupa la oficina, o 9 espacios si está libre (es decir, si el ID es `null`).
   - <SEPARADOR_VERTICAL>, un carácter de separación vertical, que por defecto será el símbolo '|'.

   Recuerda lo que devuelven los métodos `toString` de las clases `Office` y `OfficeLocator`.
3. Se termina el dibujo con una línea de guiones (el mismo número que las de separación entre plantas).

5.1 Crear el dibujo de un centro

Añade a la clase `CoworkingCenter` el método `toMap`, que genera el dibujo del centro tal como se ha descrito, y lo devuelve como una cadena. Proporciona dos versiones, una para usar el carácter de separación vertical '|', y otra que reciba el carácter de separación vertical deseado como parámetro:

```
public String toMap () {...}
public String toMap (char c) {...}
```

5.2 Guardar el dibujo del centro de coworking

Añade a la clase P3 un método:

```
saveCoworkingCenterMap(String fileName, CoworkingCenter elCoworkingCenter)
```

que llame al método `toMap` de la clase `CoworkingCenter` para obtener el dibujo del centro `elCoworkingCenter` y lo guarde en el fichero `fileName`. En `main`, tras procesar las entradas y salidas, llama a `saveCoworkingCenterMap` para guardar el mapa del centro en el fichero *CoworkingCenterMap_output.txt*.