Programación II
Curso 24-25

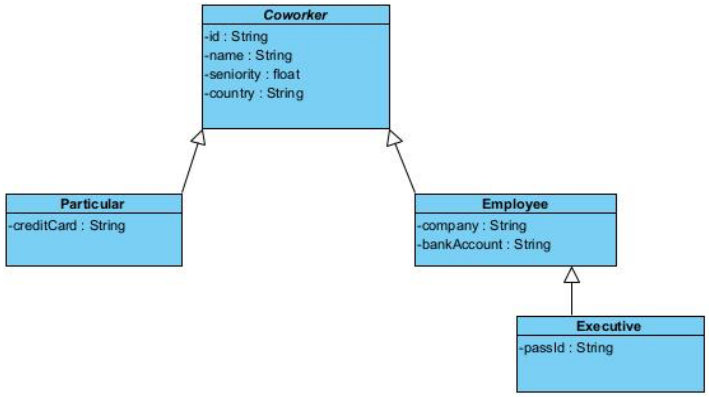
Práctica 4

0 Cuestiones previas

En esta práctica modificaremos las clases de la práctica anterior y añadiremos alguna nueva para introducir el concepto de herencia de la Programación Orientada a Objetos en la aplicación que estamos desarrollando (gestión de un centro de cotrabajo). Crea el directorio p4 dentro del directorio `Practicas` y copia a él los ficheros `.txt` usados y las clases de la práctica anterior (renombrando P3 a P4), y modifícalas según lo descrito en esta práctica.

1 Actividad 1: Crear el árbol de herencia de los distintos tipos de cotrabajador

En esta actividad vas a crear el árbol de herencia de las clases relacionadas con los cotrabajadores, de acuerdo con el siguiente diagrama UML.



Para ello, lleva a cabo las siguientes modificaciones:

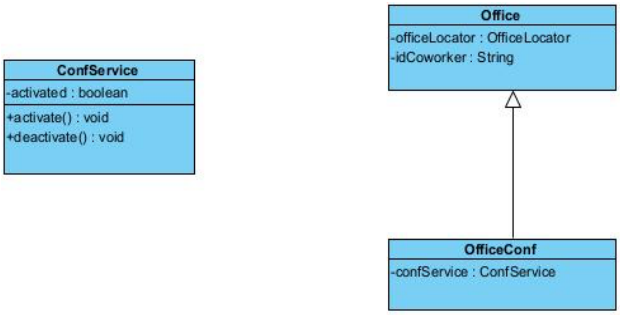
- **Superclase `Coworker`**. Crea una nueva clase **abstracta** `Coworker`. Incorpora en ella todos los atributos y métodos (estáticos y de instancia) comunes a las 3 clases `Particular`, `Employee` y `Executive`.
  - Los atributos de instancia no serán accesibles desde el exterior de la clase. Define un constructor vacío y otro completo (con argumentos para los atributos de la clase).
  - Incluye el método `toString` en la clase `Coworker`, que devuelve la cadena común del método `toString` de las tres anteriores clases de cotrabajadores ("I;N;S;C").

- **Clases `Particular` y `Employee`.** Modifica las clases `Particular` y `Employee` de forma que queden como subclases de la clase `Coworker`. Los atributos de instancia propios de cada clase también serán inaccesibles desde el exterior de la clase. Además:
  - Modifica el código de sus constructores de forma que se inicialicen los atributos de la superclase `Coworker`.
  - Modifica el método `toString` de estas clases para que usen el `toString` de su padre para lograr el mismo efecto que tenían antes.

- **Clase Executive.** Modifica la clase `Executive` de forma que quede como subclase de la clase `Employee`. Sus atributos de instancia propios también serán inaccesibles desde el exterior de la clase. Además:
  - Modifica el código de sus constructores para que se inicialicen los atributos de la superclase `Employee`.
  - Modifica el método `toString`, de tal forma que use el método `toString` de su clase padre (`Employee`) para lograr el mismo efecto que tenía antes. Fíjate que hay algo que debes cambiar en lo que genera el `toString` de la clase `Employee`.

2 Actividad 2. Oficinas con equipos de multiconferencia

Las plazas exclusivas de `Employee` o `Executive` (desde la última hasta `upperParticular+1`) tienen la opción de alquilar un equipo de multiconferencia para mantener reuniones con otros trabajadores de la empresa. Para modelar esas oficinas, crearemos las clases del siguiente diagrama UML:



- Clase `ConfService`, para modelar el dispositivo de multiconferencia, con sólo un atributo:
  - Un atributo **activated**. Un valor `boolean` que indica si el servicio está activo (`true`) o no (`false`).
  - Añade su getter y setter.

  Añade a esta clase un constructor vacío (que inicialice el atributo **activated** a `false`). Añade los métodos:
  - `activate()`. Este método debe poner el atributo **activated** a `true`.
  - `deactivate()`. Este método debe poner el atributo **activated** a `false`.

- Clase `OfficeConf`**.** Subclase de la clase `Office`, con el siguiente atributo para incluir el acceso al equipo de multiconferencia:
  - **confService**. Un objeto de clase `ConfService`.

  Crea su constructor, que cree y asigne el objeto **confService** e inicialice los atributos de su superclase `Office`.

3 Actividad 3. Trabajando con un único array general de cotrabajadores

Crea la clase `CoworkerDB`. En esa clase:

- Crea un único array llamado `coworkers` de objetos de clase `Coworker`, un atributo privado de tamaño 100 cotrabajadores.
- Mueve a esta clase el método `readCoworkersFile` de la clase P3. Modifica su código para que lea las líneas del fichero de cotrabajadores que recibe como parámetro, cree objetos `Coworker`, y los guarde en el nuevo array `coworkers`. Añade el cotrabajador al array sólo si todos sus datos son correctos. Si tienes que cambiar algún modificador, asegúrate de entender por qué (pregúntale al profesor si tienes dudas).
- Mueve aquí desde la P3 el método `getCoworkerTypeFromId`, redefiniéndolo como:

  ```
  Coworker getCoworkerFromId(String id)
  ```

  que reciba el id de un cotrabajador y devuelva el objeto `Coworker` del array `coworkers` que tiene ese id.
- Igual que en la práctica 2, añade a `CoworkerDB` el método `computeAverageSeniority`, que calcule y devuelva la media de antigüedad de los cotrabajadores.
- Crea el método `computeRatioEmployees` para calcular y devolver la ratio de cotrabajadores `Employee` (no `Executive`) respecto del total de cotrabajadores.

  ```
  public float computeRatioEmployees()
  ```

  Para ello debes recorrer el array de cotrabajadores y sumar los `Employee` no `Executive` usando el operador de Java `instanceof`. Este operador devuelve `true` si su operando por la izquierda es un objeto de la clase definida por su operando por la derecha. Por ejemplo, la siguiente operación devuelve `true` sólo si `objetoCoworker` es de tipo `Executive` (comprueba que lo hace bien):

  ```
  objetoCoworker instanceof Executive
  ```

4 Actividad 4. Modificando la clase CoworkingCenter

En esta sección modificaremos el código de la clase `CoworkingCenter` para tener en cuenta los cambios introducidos.

4.1 Modificación del fichero de definición de un centro de cotrabajo

Añadimos a cada línea de un cotrabajador un nuevo campo `<conf>`, que indica si tiene el equipo de multiconferencia activado. Ahora, cada línea de esos tipos de cotrabajador será:

```
<Loc>;<id>;<conf>
```

- **conf*:*** un carácter que indica si tiene alquilado el equipo ('Y') o no ('N').

Un ejemplo sería:

```
5:5:5;11111111A;Y  # Executive (podría ser N)
0:0:0;72222222B;N  # Particular (siempre es N)
```

4.2 Modificación al fichero de entradas y salidas

Añadimos a cada línea de entrada un nuevo campo `<conf>` para indicar si desea alquilar el equipo de multiconferencia. Ahora, cada línea será:

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

- **conf*:*** un carácter que indica si quiere alquilar el equipo ('Y') o no ('N').

Un ejemplo sería:

```
I;72345678B;N  # Particular (siempre va a ser N)
I;17654321C;Y  # Employee o Executive (podría ser N)
```

4.3 Creación del centro de cotrabajo

Modifica el constructor de la clase `CoworkingCenter` para que:

1. Al inicializar el array de oficinas, si la oficina corresponde a la zona exclusiva de `Employee` o `Executive`, se añada al array un objeto de clase `OfficeConf`, en lugar de `Office`.
2. Al ocupar una oficina, si es del tipo `OfficeConf` y tiene el equipo alquilado, se active el servicio.

4.4 Entradas al centro de cotrabajo

Para implementar la entrada de un cotrabajador al centro, modifica el método `coworkerIn` para que sea:

```
public void coworkerIn (Coworker coworker, boolean conf) {...}
```

donde `conf` indica si pide el equipo de multiconferencia o no.

A partir del objeto `Coworker` recibido, mediante el operador `instanceof`, averigua de qué tipo es para saber qué oficina asignarle. Si es un `Executive` o `Employee` debes intentar asignarle prioritariamente una de las oficinas reservadas para ellos (según el mismo criterio de la práctica anterior), y activar el equipo de multiconferencia si lo solicitan (en caso de que se le asigne una oficina con ese servicio).

4.5 Salidas del centro de cotrabajo

Actualiza el código del método `coworkerOut`, para que sea:

```
public void coworkerOut (Coworker coworker)
```

En el caso de que la oficina tenga equipo de multiconferencia y esté activado, desactívalo.

4.6 Guardar el fichero de definición de un centro de cotrabajo

Si una oficina de tipo `OfficeConf` tiene el equipo de multiconferencia alquilado, esto se reflejará en el fichero con la sintaxis anteriormente indicada (es decir, incluyendo la indicación de si el equipo de multiconferencia está o no activado). Fíjate que el `toString` de `Office` debe ser distinto al `toString` de `OfficeConf`.

5 Actividad 5. Creando la clase principal

Modificaciones a la clase P4, que contiene el método `main` y es la clase principal de la práctica.

Modifica el método `processIO` para que sea:

```
void processIO (CoworkerDB cdb, CoworkingCenter theCenter, String fileName)
```

que lee el fichero de entradas/salidas `fileName` y actualiza el centro invocando a sus métodos `coworkerIn` y `coworkerOut` para cada movimiento. En el caso de las entradas, tras leer un `id` del fichero, tendrás que llamar al método `getCoworkerFromId` del objeto `cdb` para obtener el objeto `Coworker` con el que invocar a `coworkerIn`.

Ese método `main`:

1. Recibe y lee cuatro argumentos, los nombres de cuatro ficheros:
   - `file0:` un fichero existente con la estructura y contenido inicial del centro.
   - `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. Crea un objeto de la clase `CoworkerDB` y asígnalo a la variable `cdb`. Invoca su método `readCoworkersFile` para leer el fichero `file3` con los cotrabajadores y que éstos se guarden en el array `coworkers` del objeto creado `cdb.`
3. Invoca el método `computeAverageSeniority` del objeto `cdb` para averiguar la media de antigüedad de los cotrabajadores, y muestra el resultado por pantalla según el formato *"Average seniority = 4,45"*.
4. Invoca el método `computeRatioEmployees` del objeto `cdb` para averiguar la ratio de cotrabajadores de tipo `Employee`, y muestra el resultado por pantalla según el formato "*Employee ratio = 0,42*".
5. Crea un objeto de la clase `CoworkingCenter` (asígnalo a la variable local `theCenter`) a partir del fichero `file0`.
6. Actualiza el centro a partir del fichero de entradas/salidas `file1`, llamando a `processIO`.
7. Guarda el centro, invocando el método `saveCoworkingCenterToFile` con `file2`.
8. Guarda el mapa del centro llamando al método `saveCoworkingCenterMap`.