Laboratorio 5 - Contenedores

Esta sesión de laboratorio propone el desarrollo y prueba de una sencilla aplicación orientada a objetos, formada por distintas clases, utilizando las clases que proporciona Java en el package `java.util` para implementar contenedores de tipo lista, mapa y conjunto. Además de practicar el uso de los métodos básicos de dichas clases, esta sesión de laboratorio servirá para comprender mejor sus diferencias, haciéndolas más o menos aconsejables en función del código específico a implementar.

Objetivos

Los objetivos específicos de esta sesión son:

- Implementar una aplicación orientada a objetos sencilla, formada por varias clases.
- Practicar el uso de clases del package `java.util` que implementan contenedores de tipo lista, mapa y conjunto, presentadas en las sesiones de teoría de la asignatura.
- Comprender las diferencias e identificar la aplicabilidad de listas, mapas y conjuntos, dada la funcionalidad esperada del código a implementar.
- Comprobar el correcto funcionamiento del código implementado.

Contenedores de tipo Lista

La aplicación que pretendemos implementar en esta sesión de laboratorio consiste en una herramienta de gestión de las impresoras disponibles en una oficina.

Para ello, empezad creando un nuevo proyecto Netbeans llamado `Laboratorio 5 - Contenedores`. En este proyecto, cread también un nuevo package, llamado `edu.upc.etsetb.poo.laboratorio.sesion5.main`.

Dentro de este package, vuestra primera tarea será la de implementar una nueva clase, llamada `Trabajo`, que servirá para representar aquellos trabajos (documentos) que se envían a las impresoras de la oficina para su impresión.

La clase `Trabajo` debe definir 3 atributos, todos ellos con visibilidad privada (`private`):

- id (`int`): Guardará el identificador del trabajo
- usuario (`String`): Guardará el nombre de usuario que ha enviado ese trabajo a imprimir
- descripcion (`String`): Guardará el nombre del documento que se ha enviado a imprimir

Además, la clase `Trabajo` debe implementar los siguientes métodos:

`public Trabajo(int id, String usuario, String descripcion)`; método constructor de la clase `Trabajo`. Inicializa los atributos `id`, `usuario` y `descripcion` del nuevo objeto de tipo `Trabajo` con los valores que recibe como parámetro.

`public int getId()`; método getter encargado de devolver el valor del atributo `id` del objeto de tipo `Trabajo` que recibe su invocación.

`public String getUsuario()`; método getter encargado de devolver el valor del atributo `usuario` del objeto de tipo `Trabajo` que recibe su invocación.

`public String toString()`; método encargado de devolver, en un nuevo objeto de tipo `String`, la información de los atributos del objeto de tipo `Trabajo` que recibe su invocación. El formato deseado del `String` devuelto es: `ID: <id> [<usuario>]: <descripción>`. Por ejemplo: `ID: 17 [User-1]: documento.docx`.

Una vez completada la clase `Trabajo`, debéis implementar otra clase llamada `Impresora` en el mismo package, que servirá para representar cada una de las impresoras de la oficina, como objetos de tipo `Impresora`.

La clase `Impresora` debe definir 2 atributos, ambos con visibilidad privada (`private`):

- nombre (`String`): Guardará el nombre de la impresora.
- cola (`LinkedList<Trabajo>`): Representa la cola de impresión de la impresora, que guarda los trabajos pendientes de impresión, en orden de llegada. Los nuevos trabajos que llegan a la impresora deben almacenarse al final (en la última posición) de su cola de impresión. En cambio, cuando la impresora procesa el siguiente trabajo pendiente, debe extraer (eliminar) el primer trabajo de la cola (el que está almacenado en su primera posición), en caso que la cola no esté vacía.

Nota: la cola de impresión de una impresora se implementa mediante un contenedor de tipo lista. Se ha decidido así, puesto que existe un orden entre los objetos de tipo `Trabajo` almacenados. Cuando tengáis que asegurar un orden entre los objetos guardados en un contenedor, vuestra elección deberá ser, muy probablemente, un contenedor de tipo lista, `ArrayList` o `LinkedList`.

Además, la clase `Impresora` debe implementar los siguientes métodos:

`public Impresora (String nombre)`; método constructor de la clase Impresora. Inicializa los atributos `nombre` y `cola` de un nuevo objeto de tipo `Impresora`. El atributo `nombre` lo inicializa a partir del nombre que recibe como parámetro. El atributo `cola` lo inicializa creando una nueva `LinkedList<Trabajo>` vacía.

`public void addTrabajoEnCola (Trabajo trabajo)`; método encargado de añadir el trabajo pasado como parámetro al final de la cola de impresión de la impresora que recibe la invocación del método.

`public boolean procesaSiguienteTrabajo()`; método encargado de eliminar el trabajo en la primera posición de la cola de impresión de la impresora que recibe la invocación del método. Si la cola no estaba vacía y dicho trabajo ha podido eliminarse, el método debe devolver `true`. En caso contrario, si no existe trabajo alguno en la cola para procesar, el método devuelve `false`.

`public boolean priorizaTrabajoEnCola(int id)`; método encargado de mover el trabajo con identificador igual a `id` pasado como parámetro directamente a la primera posición de la cola de impresión de la impresora que recibe la invocación del método. Si el trabajo se ha encontrado y ha podido moverse satisfactoriamente, el método acaba devolviendo `true`. Si el trabajo con el identificador proporcionado no se ha encontrado, el método devuelve `false`.

`public void limitaLongitudCola (int maxTrabajos)`; método encargado de limitar el número de trabajos en la cola de impresión de la impresora a `maxTrabajos` como máximo. Si existen más trabajos en la cola de impresión, deberán eliminarse aquellos que lleven esperando menos tiempo en ella, es decir, almacenados en sus últimas posiciones.

`public String toString()`; método encargado de devolver, en un nuevo objeto de tipo `String`, la información de los atributos del objeto de tipo `Impresora` que recibe su invocación, incluyendo su nombre y contenido de la cola de impresión. Por ejemplo:

```
Impresora: LaserPrinter-1
Trabajos actualmente en cola:
      ID: 1 [User-1]: documento.docx
      ID: 2 [User-12]: listado.pdf
      ID: 3 [User-7]: horario.docx
```

Nota: para implementar los métodos especificados para la clase `Impresora`, deberéis utilizar varios de los métodos proporcionados por la clase `LinkedList`, como por ejemplo `add()` o `remove()`. Podéis encontrar fácilmente en Internet la documentación de la clase `LinkedList` de Java para recordar así la

funcionalidad de estos métodos y su lista de parámetros. Recordad también que el recorrido completo de una `LinkedList` requiere utilizar un objeto `Iterator` o bucle for-each para que este sea eficiente.

Comprobación del correcto funcionamiento de las clases Trabajo e Impresora

En este punto debéis comprobar el correcto funcionamiento de las clases `Trabajo` e `Impresora` implementadas.

Una buena práctica de programación consiste en separar el código de comprobación del código propio de la aplicación desarrollada. En el explorador de proyectos de Netbeans encontraréis en vuestro proyecto, además de la carpeta llamada *Source Packages* otra llamada *Test Packages*. En esa carpeta es donde debéis depositar vuestro código de testeo. Para ello, cread ahí un nuevo package, llamado `edu.upc.etsetb.poo.test`. Además, cread una nueva clase en ese package, llamada `Tests`.

La clase `Tests` será clase principal (la deberemos ejecutar como programa), implementando el método `public static void main (String[] args)`; desde donde se llamarán los métodos auxiliares implementados a continuación en esa misma clase con los distintos juegos de prueba.

En la clase `Tests`, implementad el método de pruebas `public static void juegoPruebasTrabajoImpresora1()`; en ese método debéis:

1. Crear un nuevo objeto de tipo `Impresora`.
2. Crear 6 nuevos objetos de tipo `Trabajo`.
3. Añadir los 6 objetos de tipo `Trabajo` creados a la cola de impresión de la impresora.
4. Mostrar por pantalla el resultado de invocarle el método `toString()` a la impresora creada.
5. Procesar los dos siguientes trabajos de la impresora.
6. Volver a mostrar por pantalla el resultado de invocarle el método `toString()` a la impresora creada, comprobando que los trabajos procesados ya no existen en la cola de impresión.

Ahora, llamad este método desde `main()` y ejecutad la clase `Tests` como programa. El resultado de la ejecución debería ser similar a:

```
Impresora: LaserPrinter-1
Trabajos actualmente en cola:
      ID: 1 [User-1]: documento.docx
      ID: 2 [User-12]: listado.pdf
      ID: 3 [User-7]: horario.docx
      ID: 4 [User-1]: libro java.pdf
      ID: 5 [User-3]: nomina.pdf
      ID: 6 [User-3]: tareas.txt

Impresora: LaserPrinter-1
Trabajos actualmente en cola:
      ID: 3 [User-7]: horario.docx
      ID: 4 [User-1]: libro java.pdf
      ID: 5 [User-3]: nomina.pdf
      ID: 6 [User-3]: tareas.txt
```

Como podéis observar, los trabajos con identificadores 1 y 2, ubicados en las primeras posiciones de la cola de impresión, no vuelven a aparecer por pantalla, puesto que han sido procesados y eliminados de la cola de impresión.

En la misma clase `Tests`, codificad ahora el método `public static void juegoPruebasTrabajoImpresora2()`; en ese método debéis:

1. Crear un nuevo objeto de tipo `Impresora`.
2. Crear 6 nuevos objetos de tipo `Trabajo`.

3. Añadir los 6 objetos de tipo `Trabajo` creados a la cola de impresión de la impresora.
4. Mostrar por pantalla el resultado de invocarle el método `toString()` a la impresora creada.
5. Priorizar el trabajo ubicado en la última posición de la cola de impresión, para que pase a ser el primero a procesar.
6. Limitar la longitud de la cola de impresión de la impresora a 3.
7. Volver a mostrar por pantalla el resultado de invocarle el método `toString()` a la impresora creada, comprobando que sólo quedan los 3 primeros trabajos en la cola de impresión, siendo el primero de ellos el priorizado en el punto anterior.

Llamad ahora este método desde `main()` y ejecutad la clase `Tests` como programa. El resultado de la ejecución debería ser similar a:

```
Impresora: LaserPrinter-1

Trabajos actualmente en cola:
      ID: 1 [User-1]: documento.docx
      ID: 2 [User-12]: listado.pdf
      ID: 3 [User-7]: horario.docx
      ID: 4 [User-1]: libro java.pdf
      ID: 5 [User-3]: nomina.pdf
      ID: 6 [User-3]: tareas.txt

Impresora: LaserPrinter-1
Trabajos actualmente en cola:
      ID: 6 [User-3]: tareas.txt
      ID: 1 [User-1]: documento.docx
      ID: 2 [User-12]: listado.pdf
```

Contenedores de tipo Mapa

Una vez implementadas y testeadas las dos clases anteriores, continuaremos el desarrollo de la aplicación añadiendo en el package `edu.upc.etsetb.poo.laboratorio.sesion5.main` una nueva clase llamada `Oficina`, que servirá para representar la oficina en sí, cuyas impresoras deben gestionarse.

Esta clase definirá un atributo, llamado impresoras, de tipo `HashMap<String, Impresora>` con visibilidad `private`. Este atributo guardará las impresoras de la oficina identificadas por su nombre como clave, que asumimos único (no habrá dos impresoras en la oficina con mismo nombre). Así, podremos conseguir fácilmente las impresoras de la oficina a partir de su nombre. Además, definirá tres constantes (`public static final`), llamadas `IMPRESORA_NO_EXISTE`, `TRABAJO_NO_EXISTE` y `OK_ACTION`, inicializadas igual a -1, -2 y 0 respectivamente.

Nota: las impresoras de la oficina se almacenan en un contenedor de tipo mapa. Se ha decidido así, puesto que los objetos de tipo `Impresora` se pueden identificar unívocamente a partir del valor de uno de sus atributos, su `nombre`. Guardar los objetos de tipo `Impresora` en un mapa identificados por su nombre como clave facilitará posteriores búsquedas de impresoras a partir de su nombre. Si detectáis que los objetos que debéis guardar en un contenedor pueden ser identificados a partir del valor de uno de sus atributos, podéis utilizar un mapa, `HashMap` o `TreeMap`. Sin embargo, debéis tener en cuenta que los mapas no están pensados para asegurar un orden dentro de ellos. Si el orden es imprescindible, un contenedor de tipo lista puede seguir siendo más apropiado.

En cuanto a los métodos, la clase `Oficina` deberá implementar los siguientes:

`public Oficina()`; método constructor de la clase `Oficina`. Debe inicializar el atributo impresoras tal que sea un nuevo `HashMap<String, Impresora>` vacío.

`public boolean nuevaImpresora (String nombre)`; método encargado de crear y guardar una nueva impresora con nombre igual al pasado como parámetro. Si ya existía previamente otra impresora

en la oficina con el mismo nombre, el método debe devolver `false` directamente. En caso contrario, si la operación es satisfactoria, debe devolver `true`.

`public int addTrabajoImpresora (String nombre, Trabajo trabajo)`; método encargado de añadir el trabajo pasado como parámetro a la cola de impresión de la impresora con nombre igual al también pasado como parámetro. Si no existe ninguna impresora con ese nombre en la oficina, el método debe devolver `IMPRESORA_NO_EXISTE.` En caso contrario, si la operación es satisfactoria, debe devolver `OK_ACTION`.

`public int procesaTrabajoImpresora (String nombre)`; método encargado de procesar el siguiente trabajo de la impresora con nombre igual al pasado como parámetro. Si no existe ninguna impresora con ese nombre en la oficina, el método debe devolver `IMPRESORA_NO_EXISTE.` Si la impresora existe, pero su cola está vacía el método debe devolver `TRABAJO_NO_EXISTE`. En caso contrario, si la operación es satisfactoria, debe devolver `OK_ACTION`.

`public String getInfoImpresoras()`; método encargado de devolver un `String` con la información completa de todas las impresoras de la oficina (nombre y trabajos pendientes). Si la oficina no dispone de ninguna impresora, el método debe devolver `"No se ha encontrado ninguna impresora."`

Nota: para implementar los métodos especificados para la clase `Oficina`, deberéis utilizar varios de los métodos proporcionados por la clase `HashMap`, como por ejemplo `put()` o `get()`. Podéis encontrar fácilmente en Internet la documentación de la clase `HashMap` de Java para recordar así la funcionalidad de estos métodos y su lista de parámetros.

Comprobación del correcto funcionamiento de la clase Oficina

En este punto debéis comprobar el correcto funcionamiento de la clase `Oficina` implementada, mediante un nuevo juego de pruebas implementado en un método auxiliar en la clase `Tests`.

Codificad el método de pruebas `public static void juegoPruebasOficina()`; en ese método debéis:

1. Crear un nuevo objeto de tipo `Oficina`.
2. Crear 3 nuevos objetos de tipo `Impresora`, inicializados con distintos nombres.
3. Crear 6 nuevos objetos de tipo `Trabajo`.
4. Añadir dos de los objetos de tipo `Trabajo` creados a la cola de impresión de cada impresora.
5. Mostrar la información de todas las impresoras de la oficina.
6. Intentar crear una nueva impresora con idéntico nombre al de una anteriormente creada, imprimiendo el retorno del método de la clase `Oficina` invocado para ello.
7. Intentar añadir un nuevo objeto de tipo `Trabajo` a una impresora inexistente, imprimiendo el retorno del método de la clase `Oficina` invocado para ello.
8. Intentar procesar el siguiente trabajo de una impresora inexistente, imprimiendo el retorno del método de la clase `Oficina` invocado para ello.
9. Intentar procesar los siguientes 3 trabajos de una impresora de la oficina, imprimiendo el retorno del método de la clase `Oficina` invocado cada vez para ello.
10. Mostrar la información de todas las impresoras de la oficina.

Llamad ahora este método desde `main()` y ejecutad la clase `Tests` como programa. El resultado de la ejecución debería ser similar a:

```
Impresora: LaserPrinter-3
Trabajos actualmente en cola:
      ID: 5 [User-3]: nomina.pdf
      ID: 6 [User-3]: tareas.txt

Impresora: LaserPrinter-2
Trabajos actualmente en cola:
```

```
      ID: 3 [User-7]: horario.docx
      ID: 4 [User-1]: libro java.pdf

Impresora: LaserPrinter-1
Trabajos actualmente en cola:
      ID: 1 [User-1]: documento.docx
      ID: 2 [User-12]: listado.pdf

false
-1
-1
0
0
-2

Impresora: LaserPrinter-3
Trabajos actualmente en cola:
      ID: 5 [User-3]: nomina.pdf
      ID: 6 [User-3]: tareas.txt

Impresora: LaserPrinter-2
Trabajos actualmente en cola:
      ID: 3 [User-7]: horario.docx
      ID: 4 [User-1]: libro java.pdf

Impresora: LaserPrinter-1
Trabajos actualmente en cola:
```

Como puede observarse, al procesar todos los trabajos de la impresora con nombre `LaserPrinter-1`, su cola está actualmente vacía. Incluso se ha intentado procesar un trabajo adicional cuando su cola de impresión ya estaba vacía, de ahí el retorno igual a -2 mostrado.

Contenedores de tipo Conjunto

Finalmente, en la misma clase `Oficina` queremos añadir un método adicional:

```
public void muestraUsuariosActualmenteImprimiendo();
```

Este método debe mostrar por pantalla los nombres de los usuarios que actualmente están imprimiendo algún trabajo en cualquiera de las impresoras de las oficinas. Debe tenerse en cuenta que un usuario concreto puede estar imprimiendo múltiples trabajos en un momento determinado. Sin embargo, el método debe mostrar el nombre de cada usuario una única vez, como máximo. Para implementar esta funcionalidad, se sugiere utilizar un contenedor de tipo `HashSet<String>`.

Nota: una de las características principales de los contenedores de tipo conjunto, implementados por las clases `HashSet` y `TreeSet` de Java, es que no admiten elementos duplicados dentro de ellos. Cuando se intenta añadir elementos a un contenedor de tipo conjunto mediante la invocación del método `add()`, si el elemento no existía aún en el contenedor, este lo acepta y el método devuelve `true`. Sin embargo, si el elemento ya existía en el contenedor, el método `add()` lo rechaza, devolviendo `false` directamente. Es importante tener en cuenta para permitir la correcta funcionalidad de los conjuntos, éstos deben poder determinar cuando dos objetos son iguales o distintos. Para ello, la clase de objetos guardada en el conjunto debe implementar los métodos `equals()` y `hashCode()` que, por ejemplo, la clase `String` de Java ya implementa. Si queremos usar conjuntos para almacenar objetos de nuestras propias clases de usuario, deberemos implementarlos.

Una vez implementado el método, debéis comprobar su funcionalidad, mediante un nuevo juego de pruebas implementado en un método auxiliar en la clase `Tests`. El nuevo método con este juego de pruebas deberá llamarse `public static void juegoPruebasConjuntos1()`; esta vez será vuestra tarea la de proponer un juego de pruebas adecuado para validar el método implementado, tomando como referencia los juegos de pruebas implementados anteriormente.

Implementación de métodos equals() y hashCode()

Para finalizar la presente sesión, implementaremos los siguientes métodos en la clase `Trabajo`, que permitirían la detección de objetos de tipo `Trabajo` duplicados en un conjunto: `public boolean equals(Object o);` y `public int hashCode();`

Ambos métodos, en realidad, sobreescriben su implementación heredada de la superclase `Object`. Es por ello que deben implementarse con exactamente esa cabecera, sin cambiar su visibilidad, tipo de retorno, nombre y tipo de parámetros.

El primer método `equals()` debe devolver `true` si el objeto que recibe la invocación del método es igual al que se pasa como parámetro, o `false` en caso contrario. Es en este método donde podemos establecer nuestro criterio para decidir si dos objetos de tipo `Trabajo` son iguales o distintos. Por ejemplo, podríamos decidir que dos objetos `Trabajo` son iguales si su identificador, usuario y descripción son iguales, o false si cualquiera de ellos es distinto entre ambos objetos.

Normalmente, el método `equals()` requiere una operación de cast, para indicar al compilador de Java que el objeto recibido como parámetro es en realidad de un cierto tipo (típicamente del tipo definido por la clase que sobreescribe el método). Por ejemplo:

```
public class Trabajo {

    public boolean equals (Object o) {
        if (o == null || o instanceof Trabajo == false)
            return false;   //Si recibimos null como param., o "o" no apunta a un objeto
                            //de tipo Trabajo, seguro que son objetos distintos

        Trabajo trabajo = (Trabajo)o;    //Operación de cast

        //Comparación de "this" con "trabajo", para acabar devolviendo true o false
    }
}
```

En cuanto a `hashCode()`, este método devuelve un valor entero, obtenido a partir de los valores de los atributos del objeto que recibe su invocación. El punto importante a tener en cuenta es el siguiente: si el resultado de la comparación entre dos objetos mediante el método `equals`() devuelve `true` como resultado, determinando que ambos objetos son iguales, la invocación de `hashCode`() sobre cada uno de ellos debe devolver el mismo valor.

Como ejercicio, empezad implementando en la clase `Tests` un nuevo método auxiliar, llamado `public static void juegoPruebasConjuntos2()`; en este método cread 3 objetos de tipo `Trabajo`, inicializando dos de ellos con mismo identificador, usuario y descripción. Ahora cread un nuevo contenedor de tipo `HashSet<Trabajo>`, al que debéis añadirle los 3 objetos de tipo `Trabajo`. Finalmente, debéis recorrer los objetos dentro del `HashSet` y mostrarlos por pantalla. ¿Ha detectado correctamente el HashSet los dos objetos de tipo Trabajo "iguales", admitiendo sólo uno de ellos?

Ahora, implementad en la clase `Trabajo` los métodos `equals()` y `hashCode()` tal y como se ha descrito en los párrafos anteriores. Puesto que equals() debe devolver true si el valor de los 3 atributos de ambos objetos `Trabajo` son iguales, una estrategia para implementar `hashCode()` es concatenar, en un único `String` la representación textual de los 3 atributos del objeto `Trabajo` que recibe la invocación del método. Y sobre este `String` invocarle a su vez el método `hashCode()` para acabar devolviendo el

resultado devuelto por este. Puesto que dados dos `String` iguales, el valor devuelto por `hashCode()` sobre ellos es es mismo, conseguiremos así nuestro objetivo.

Después de implementar ambos métodos en la clase Trabajo, volved a ejecutar el juego de pruebas anterior. ¿Ha detectado ahora el HashSet los dos objetos de tipo Trabajo “iguales”?