Sesión 11. Adición de capacidades a la sesión 10 de laboratorio

Descripción

En esta sesión se realizarán 4 partes:

1- Revisar implementación de FileUtil incluyendo refactorización con superclase. Añadir nueva versión en el proyecto mp.util
2- Revisar la actualización de la clase Logger para que grave en un fichero
3- Revisar los test implementados como tarea no presencial
4- Abordar la ordenación de listas con operaciones compareTo y compare
5- Incluir opciones de ordenación en proyecto newsstand (como tarea No presencial)
6- Iniciar proyecto de gestión de exámenes.

Ordenación

Problema: Ordenar una lista de objetos por algún criterio.

Cuando queremos ordenar objetos de una lista, debemos hacerlo por algún criterio. Por ejemplo, ordenar personas por nombre, por apellido, por edad… e incluso podríamos preferir criterios formados a partir de varios campos, por ejemplo ordenar por edad y a igual edad ordenar por nombre, o bien ordenar por apellido y a igual apellido ordenar por nombre.

Para realizar la ordenación de una lista es necesario ir comparando cada elemento con los que ya están ordenados en una lista ordenada e insertar en la posición correcta. Esta comparación de dos elementos se puede realizar de dos modos:

1. Uso del método **compareTo(Object o)** de la interfaz **Comparable<Object>.**

   Para comparar dos objetos podemos hacer algo similar a la operación equals. Equals solo comprueba si son iguales. El método compareTo, que se añade a una clase, compara el objeto propio (el ejecuta el método compareTo) con el que recibe como parámetro, comprueba cuál es mayor y devuelve 0 si son iguales, > 0 si el propio objeto es mayor que el que recibe y < 0 si el objeto es menor que el que recibe. Esta operación compareTo está definida en la interfaz Comparable<Object> de Java y se puede implementar en la clase que haga falta (por ejemplo, en la clase Person el método compareTo puede comparar si la edad de uno es mayor que la del otro, o puede comparar si el nombre de uno es mayor (alfabéticamente) que el del otro).

   Java tiene implementado el método compareTo en las clases Integer y String. Por tanto, si en Person se desea comparar por Nombre, en su compareTo devolveremos el resultado de comparar sus nombres con el compareTo de la clase String.

   Si se usa esta opción (incluir el método compareTo en una clase) se conoce como método natural de ordenación y en ella se suele incluir el criterio más comúnmente utilizado (por ejemplo en Persona ordenar por apellido.

Además se pueden necesitar otros criterios de ordenación, para poder tener diferentes ordenaciones (por ejemplo, ordenar por apellido sería uno y ordenar por edad sería otro) . Para ello se usaran comparadores externos que serán explicados a continuación.

2. Uso del método **compare(Object o1, Object o2)** de la interfaz **Comparator<Object>.**

Es posible crear una clase que implementa la interfaz Comparator<Object> con un método compare(Object o1, Object o2), que compare por algún criterio los dos objetos que recibe como parámetro.

Por ejemplo crearíamos la clase AgePersonComparator con el método compare(Person p1, Person p2) para comparar dos personas por su edad. Este métdo devuelve 0 si tienen la misma edad, un número > 0 , si la primera es mayor que la segunda y un número < 0 si la primera es menor que la segunda.

Implementación de método sort de ordenación de una lista

Vamos a crear un método de ordenación estático dentro de la clase Collections que luego se podrá usar como utilidad de para cualquier ordenación de listas.

Partiremos del esqueleto proporcionado que contiene las pruebas para el método sort de la clase Collections.

Primera parte: dentro de la clase `Collections` crearemos el método:

- **public static void sort(List list)**
  Ordena la lista especificada en orden ascendente, de acuerdo con el orden natural de sus elementos (especificado en su método `compareTo(Object o)`). Para ello, los elementos de la lista deben implementar la interfaz `Conparable<Object>`.

Segunda parte: dentro de la clase `Collections` crearemos el método:

- **public static void sort(List list, Comparator<Object> comparator)**
  Ordena la lista especificada de acuerdo con el orden indicado por el comparador proporcionado. Crearemos un comparador de tipo Comparator<Object> con el método compare que compara por el criterio requerido. (por ejemplo, ByAgePersonComparator para comparar personas por edad de manera que ordene de menor a mayor.

Nota: Para el algoritmo de ordenación crearemos una nueva lista vacía para guardar los elementos ordenados e iteraremos sobre la lista recibida introduciendo cada elemento, en orden, en la nueva lista ordenada. Una vez finalizado, copiaremos la lista ordenada en la lista original para que ésta quede ordenada.

Primera iteración

- El método sort recibe una lista de objetos de tipo Integer (implements Comparable<Integer>

Segunda iteración

- El método sort recibe una lista de objetos de tipo String (implementas Comparable<String>

Tercera iteración

Implementamos un método sort que sirva para cualquier lista. Para ello no es posible creando listas de Object puesto que no podemos implementar el método compareTo en Object. Tenemos que usar la genericidad.

Importante:

- Para ordenar personas con el método sort(list) debemos tener implementado el método compareTo en la clase Person
- Para ordenar personas con el método sort(list, comparator) debemos tener implementado una clase Comparator<Person> con el método compare donde establecemos el criterio de comparación.

Inclusión de ordenación en proyecto newsstand

1- Realiza una copia del proyecto correspondiente a la tarea Sesión 11.
2- Añade una opción 9 "Ordenar publicaciones por nombre" al menú e impleméntala oportunamente. Esta opción ordena las publicaciones por el criterio natural que sería por nombre de publicación. Utiliza para ello la operación sort de la clase Collections realizada anteriormente enlazando para ello este proyecto con el otro. Implementa el método compareTo en la clase Publication para realizar la comparación.
3- Añade una opción 10 "Ordenar publicaciones por ventas". Esta segunda ordenación será por ventas realizadas. En este caso deberás crear un comparador para este criterio.
4- Añade la opción 11 "Ordenar peticiones por pedidos y nombre". A igual número de pedidos se deberá ordenar por nombre. Usa un comparador para realizar esta ordenación.

Modificación de método log para grabación de mensaje en fichero

Sustitutir la salida en System.err por una impresión del mensaje en un fichero.

Utilizaremos para escritura en este caso un flujo de tipo PrintStream, que tiene operaciones con formato como print o println. Nótese que los objetos System.out y System.err también son de tipo prinStream.

Para crear un flujo de tipo PrintStream debe recibir la información de otro flujo que le proporcione bytes (no caracteres) así que crearemos un flujo de tipo FileOutputStream que escribe bytes a un fichero.

Para asegurarnos de que escribe el mensaje al final del fichero ya existente habrá que añadir un segundo parámetro en la creación de FileOutputStream de tipo booleano con valor True.

Inicio del proyecto de gestión de exámenes

Ver enunciado independiente Proyecto Calificaciones

Tarea NO Presencial.

- Completar ordenación del proyecto Newsstand y dejar el proyecto completamente acabado (excepciones, test, lógica, carga y descarga.
- Completar las operaciones de la clase ExamMarker: loadQuestions, loadEstudentExams, getQuestions y getStudentExams. Hacer test de los 4 métodos.

Esta semana no habrá que revisar tarea de otros.