SESIÓN 04

La práctica corresponde al
Tema 3: Estructuras de Datos Lineales

Peso Total en la nota: 0%

Objetivos

Con esta práctica se pretende que el alumno se familiarice y comprenda:

1. Inicio al estudio de la jerarquía básica del *Java Collections Framework*
   a) Se mostrará la estructura básica de Interfaces
   b) Se explicará la jerarquía de herencia de `java.util.List`
2. Las diferentes variantes de las estructuras de datos basadas en Listas
   a) Implementan el interfaz `java.util.List`
   b) Estructura interna de cada una y como afecta a su rendimiento
3. El coste computacional que conlleva a la utilización de una u otra estructura.
   a) Identificar en que ocasiones resulta más favorable el uso de una u otra.

Descripción

Código Fuente

Se pide al alumno construir una **clase estática** que permita el manejo avanzado de elementos sobre estructuras de datos de tipo Lista.

Los métodos de dicha clase recibirán como parámetro un objeto de cualquier clase que implemente el interfaz `java.util.List` y permitirán la selección de contenidos de las listas por medio de su índice.

La clase se deberá incluir en el paquete `es.ubu.gii.edat.s04`; tendrá por nombre `SeleccionListas` e implementará los métodos:

- `public <E> List<E> seleccionMultiple(List<E> lista, int[] seleccionados)`

Este método permite hacer una selección de elementos no consecutivos dentro de una lista facilitada, construyendo con ellos una sublista de la original.

El método recibe como parámetros una Lista y un array de enteros, que contiene los indices de los elementos que deberán incluirse en la nueva sublista. El orden en que se incluirán en la nueva sublista debe ser el mismo en el que aparecen en el array de “seleccionados” (no tiene por qué respetarse el orden en que aparecían en la lista original). Devuelve como salida un objeto de una clase que implemente `java.util.List`, incluyendo los elementos solicitados.

- `public <E> List<E> seleccionInversaMultiple(List<E> lista, int[] eliminados)`

Este método funciona de manera muy similar al anterior. La diferencia es que se pasa como parámetro los índices de los elementos que No se incluirán en la sublista. De esta forma, la sublista resultante será la formada por todos los elementos cuyos índices no aparecen en el array facilitado. Recibe igualmente como parámetros una Lista y un array de enteros, que contiene los índices de los elementos que deberán excluirse en la nueva sublista. Devuelve como salida un objeto de una clase que implemente java.util.List, incluyendo los elementos solicitados.

- `public <E> List<List<E>> partición (List<E> lista, int[] destino)`

Este método permite dividir o particionar la lista original en varias sublistas diferentes. Se le pasará como parámetro la Lista original y un array de enteros (de la misma longitud que la lista), que indica el número de sublista en la que se desea incluir el elemento que ocupa esa posición en la lista original. En caso de que No se desee incluir en ninguna de las sublistas, la posición se marcará con el elemento -1. Se devolverá como resultado una lista de listas, en el que cada elemento será una de las sublistas en la que se pretende particionar la lista original. En caso de que el tamaño de la lista y del array de destinos no coincidan, se lanzará la excepción `NoSuchElementException`.

Para **los tres métodos:** La lista original que se pasa como parámetro NO podrá ser modificada y permanecerá igual que antes de llamar al método correspondiente. En caso de que se pida seleccionar un índice fuera de los disponibles en la lista, se lanzará una `IndexOutOfBoundsException`. Si la lista sobre la que se debe seleccionar no está disponible (p. ej. es nula) se lanzará `NoSuchElementException`.

Informe

Como se explica en la sesión de prácticas correspondiente, existen varias implementaciones diferentes del interfaz `List` en java. La clase que se pide implementar deberá funcionar de la misma forma con cualquiera de ellos.

Se pretende además que el alumno compruebe sobre un caso práctico las diferencias explicadas en clase entre las diferentes listas. Por ello se pide al alumno que, una vez completada la clase a programar, realice el siguiente experimento: dado que se facilita al alumno una forma de comprobar el tiempo que se tarda en completar cada uno de los métodos; se debe modificar este fichero para que se ejecute con varios de los tipos de listas diferentes, a fin de compararlos.

Se pide que se genere un informe en el que:

1. Se comparen al menos 2 de los diferentes tipos de listas explicados. Para esta comparativa, se necesitarán generar algunas gráficas, similares a las mostradas en el Tema 2 – Algoritmos: Introducción.
2. Se justifique el porqué de las diferencias que aparecen en el tiempo de ejecución de cada uno de los métodos al emplear una u otra variante de la lista.

La extensión del informe será de aproximadamente 5 páginas (incluyendo portadas, índices, gráficas...etc.).

Recursos Necesarios

Se facilita junto con este enunciado un fichero de test de jUnit (v.4) que permite comprobar que el código generado responde a unos mínimos de corrección incluidos en la descripción de métodos anterior. Estos métodos de pruebas aparecen en el fichero `TestSeleccionListas`.

Igualmente, se facilita un fichero de test que simplemente comprueba el tiempo de ejecución de cada uno de los métodos pedidos, variando el número de elementos. Este segundo fichero (`TestRendimientoSeleccionListas`) se puede emplear para completar el informe pedido.

Para obtener información adicional, se sugiere la consulta de los recursos:

http://docs.oracle.com/javase/tutorial/collections/interfaces/list.html

http://docs.oracle.com/javase/tutorial/collections/implementations/list.html

Condiciones de Entrega

La entrega consistirá en el código fuente generado en la solución, junto con los comentarios incluidos en el fichero, el informe de texto con la comparativa de la complejidad algorítmica y la documentación generada en ficheros aparte con la herramienta Javadoc. Se solicita que los ficheros aparezcan contenidos en la misma estructura de paquetes que se facilita en el material.

- La práctica se realizará en grupos de 1 o 2 personas.
- La entrega se hará SOLAMENTE por medio de la plataforma UBUVirtual. Cada miembro del grupo deberá subir su propia copia de la solución. No se admitirán soluciones fuera de plazo o por otros medios de entrega (e-mail, mensajería interna, …).
- La entrega incluirá la información de quienes son los autores de la misma, tanto en los ficheros de código fuente como en los documentos asociados.
- Cada entrega consistirá en un fichero comprimido (formato .zip), con la estructura de nombre
  "`Apellidos1Nombre1_Apellidos2Nombre2`"
- Qué deberá incluir el fichero comprimido:
  - Código fuente de la solución (dentro de la estructura de paquetes necesaria)
  - Documentación en formato Javadoc
  - Informe en formato PDF (en aquellas prácticas que se solicite).
- No hace falta entregar ficheros binarios

Criterios de Evaluación

- **Corrección del funcionamiento**. Se valorará como aspecto más importante que el funcionamiento de la estructura de datos implementada cumpla con todas las condiciones solicitadas, tanto en la documentación oficial como en el enunciado. Se facilitan al alumno un subconjunto de los tests que se emplearán en la corrección, pero se aconseja que el alumno realice de forma adicional los que crea conveniente.
- **Complejidad algorítmica**: Como parte fundamental de los conocimientos de la asignatura, se considerará la complejidad algorítmica de las soluciones proporcionadas el segundo aspecto más importante a valorar en los ejercicios. Penalizará en la nota aquellas soluciones

que, a pesar de proporcionar salidas correctas, lo hagan con una complejidad mucho mayor de la estrictamente necesaria.

- **Informe**: se tendrá en cuenta la corrección del informe de complejidad algorítmica de la solución. No es necesario un informe excesivamente extenso, sino que se centre en la comparación de la complejidad algorítmica solicitada. La extensión aproximada es de 5 páginas.
- **Corrección del código**. Ausencia de *warnings* u otros elementos no deseables como variables no utilizadas, código que no se emplea, catch vacíos, ausencia de marcas sobre los métodos (*@Override*, @...) etc.
- **Documentación**. Las explicaciones y razonamientos que se incluyan a modo de comentarios en el código que entrega el alumno. Se solicita que el alumno entregue adicionalmente la documentación generada en formato javadoc, junto con los ficheros de código fuente.