PRÁCTICA 05

La práctica corresponde al:
Tema 3: Estructuras Arbóreas

Peso en la nota final:
6%

Objetivos

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

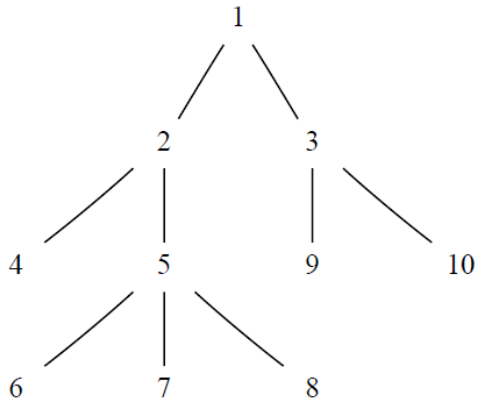
1. Estudio de la jerarquía básica del Java Collections Framework
   a) Se estudian brevemente las clases abstractas del Framework
   b) Se estudia la forma de extender la estructura básica del Framework
2. El funcionamiento interno de una estructura en forma de Árbol
   a) Varios requisitos se pueden completar aprovechando características como la recursividad.
3. Como se puede implementar una clase desde el interfaz Map.

Conceptos Teóricos

Se pretende generar una clase que permita almacenar un conjunto de objetos de forma que se preserve una jerarquía entre ellos. De esta forma, se podrán realizar consultas como "que elementos forman parte de una determinada categoría" o "a que categorías pertenece un determinado elemento". Su nombre será `ArbolTabulado` y se incluirá en el paquete `es.ubu.gii.edat.pr05`.

Para su correcto funcionamiento, el árbol deberá permitir la asociación de cada nodo a insertar con su padre en la estructura (que será único). Extenderá por tanto de la clase `AbstractMap`.

A pesar de no ser la opción más eficiente, se pide implementar la estructura del árbol apoyándose en una tabla de datos debido a la sencillez de su implementación (consultar los apuntes de teoría). La representación gráfica de la implementación sugerida se incluye en el esquema de la Ilustración 1:



Nodo	1	2	3	4	5	6	7	8	9	10
Padre	0	1	1	2	2	5	5	5	3	3

Ilustración 1: Representación de un árbol empleando una tabla

Para simplificar el manejo de la tabla de datos, se sugiere que la estructura de árbol a construir se apoye internamente en la clase que implementa la **tabla hash** en el *Java Collections Framework*.

Trabajo del Alumno

1. Programa

Se solicita que la clase incluya los siguientes métodos:

- `public ArbolTabulado()` y
- `public ArbolTabulado(int initSize)`

  Constructores que permiten definir la clase con un tamaño inicial o dejar este sin especificar, dejando esta decisión al programador usuario de la clase.

- `public E put(E hijo, E padre)`

  Método que permite insertar un nuevo elemento en el árbol. Necesita como parámetros tanto el dato a insertar como el elemento que se considerará padre del mismo. Devuelve el anterior padre de ese nodo en caso de estar ya incluido anteriormente o `null` en caso contrario. Se considerará la raíz del árbol aquel elemento cuyo padre sea `null`. (este será único en el árbol). En caso de solicitar incluir un dato cuyo padre no está incluido previamente en el árbol, se lanzará la excepción `IllegalArgumentException`.

- `public E remove(Object objeto)`

  Método que permite eliminar un elemento del árbol. En caso de tener hijos asociados a ese elemento, estos pasarán a ser hijos del elemento padre del eliminado. Devuelve el elemento padre del objeto eliminado. En caso de eliminar la raíz, se escogerá una nueva de entre sus elementos hijos de forma aleatoria.

- `public List<E> descendants(E elemento)`

  Método que permite consultar todos los elementos descendientes de un elemento dado. No necesitan devolverse en un orden particular.

- `public List<E> ancestors(E elemento)`

  Método que permite consultar todos los elementos ancestros de un elemento dado. No necesitan devolverse en un orden particular.

- `public int depth(E elemento)`

  Método que permite consultar la profundidad a la que se encuentra un elemento dentro del árbol (a partir de la raíz).

- `public int height(E elemento)`

  Método que permite consultar la altura a la que se encuentra un elemento dentro del árbol.

- `public List<E> breadthTraverse()`

  Permite obtener todos los elementos contenidos dentro del árbol obtenidos por medio de un recorrido en anchura (es decir, se devuelven ordenados en función de los niveles del árbol).

2. Informe

Se pide al alumno que, una vez completada la clase a programar, realice el un informe sencillo en el que se incluya un razonamiento de la complejidad algorítmica de cada uno de los métodos que se han implementado. No es necesario realizar un análisis completo y detallado de cada método (es decir, implementar las pruebas de tiempo de ejecución para el análisis de complejidad), pero sí incluir un razonamiento que relacione el código fuente que se facilita y a la complejidad que se considera que tiene el mismo.

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

3. Material Adicional

Se facilita junto con este enunciado un fichero de test de jUnit (v.5) 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 `ArbolTabuladoTest`.

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

- Map (Java SE 24 & JDK 24)
- AbstractMap (Java SE 24 & JDK 24)

Condiciones de Entrega

- 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.