Examen de Estructuras de Datos y Algoritmos
(Ingeniería Informática)

Febrero 2008

Primera parte (50% nota del examen)

1) Se desea escribir un método Java al que se le pasan como parámetros una lista de elementos de la clase `String` y dos strings `a` y `b`. El método retorna otra lista que es la sublista que contiene los elementos de la lista original situados entre `a` y `b`, inclusives. Si `a` no está en la lista original, se retornará `null`. Si `b` no está en la lista original se retornará la sublista que comienza en `a` y finaliza en el último elemento. Si `a` o `b` están en la lista varias veces, se tomará en cuenta sólo la primera de sus apariciones.

   Suponer que en la lista original el acceso posicional es lento, por lo que se requiere el acceso secuencial, con un iterador. Para mayor eficiencia, se requiere un solo recorrido de la lista.

   La cabecera del método que se pide es:

```
public static List<String> sublista
        (List<String> listaOriginal, String a, String b)
```

2) Se dispone de un mapa que sigue la interfaz `Map` de Java y que relaciona números de vuelo, que son `String`s, con listas de pasajeros, que son listas de `String`s, alojadas en objetos que siguen la interfaz `List` de Java. Se desea hacer un método que tenga la cabecera indicada abajo y que permita encontrar los vuelos en los que aparece un determinado `pasajero` que se pasa como parámetro. El método retornará un conjunto que siga la interfaz `Set` de Java, y que contenga los números de vuelo encontrados.

```
public static Set<String> encuentraVuelos
        (Map<String,List<String>> mapa, String pasajero)
```

3) Se dispone de un grafo que muestra las relaciones de amistad entre un grupo de científicos. En este grafo cada vértice contiene el nombre del científico (un `String`). Si hay un arco de un vértice a otro esto implica una relación de amistad. El grafo es dirigido, por lo que al ser la relación de amistad mutua, por cada arco de una persona a otra, hay otro en sentido contrario.

   Algunos científicos han escrito artículos y los han enviado a un congreso. Se han asignado revisores a estos trabajos, para su evaluación. Los revisores son otros científicos que no tengan relación de amistad con el autor del trabajo. Es preciso verificar si la asignación de revisores es válida o no, según la ausencia o presencia de relaciones de amistad. La asignación de revisores es una lista que sigue la interfaz `List` de Java y en la que los elementos son objetos de la clase `Asignacion` cuya interfaz se muestra abajo y que contienen cada uno un revisor y un trabajo. Cada trabajo es a su vez un objeto de la clase `Trabajo`, parte de cuya interfaz se muestra abajo.

   Escribir un método que, dado un grafo que contiene las relaciones de amistad y que sigue la interfaz `Grafo` vista en clase, y dada una asignación de revisores descrita en una lista, retorne un booleano que indique si la asignación es compatible o incompatible con las reglas de amistad descritas arriba. La interfaz de este método será:

```
public static boolean esCompatible
        (Grafo<String> amistades,
         List<Asignacion> asignacionRevisores)


public class Asignacion {
    /** retorna el nombre del revisor */
    public String revisor()

    /** retorna el trabajo */
    public Trabajo trabajo()
}


public class Trabajo {
    /** retorna la referencia del trabajo */
    public String referencia()

    /** retorna el autor del trabajo */
    public String autor()
    ...
}
```

Examen de Estructuras de Datos y Algoritmos
(Ingeniería Informática)

Febrero 2008

Segunda parte (50% nota del examen)

4) Se desea implementar un conjunto siguiendo una técnica similar a la de las tablas de troceado abierto. Los elementos del conjunto se guardan en una tabla, en la casilla obtenida mediante la función `valorHash` que se muestra abajo. Cada casilla de la tabla contiene una lista de elementos no repetidos. La clase se denomina `Conjunto`. Sigue la interfaz `Set`, y sus atributos se muestran abajo. Las casillas de la tabla que no albergan ningún elemento contienen una referencia nula (`null`)

   Lo que se pide es implementar la función `add`, que añade un elemento al conjunto. Si el elemento ya existe, retorna `false`; si no existe, lo añade a la lista de su casilla. Antes de ello, si la lista no existe (es decir, si la casilla de la tabla vale `null`) hay que crearla.

```
import java.util.*;
public class Conjunto<E> implements Set<E>
{
    // atributos privados
    private LinkedList<E>[] tabla;

    /**
     * Constructor; se le pasa el tamaño de la tabla
     */
    public Conjunto(int tamano) {
        tabla =  new LinkedList[tamano];
    }

    private int valorHash(E elem) {
        return elem.hashCode()% tabla.length;
    }

    public boolean add(E elem) {...}
    ...
}
```

5) Escribir el *pseudocódigo* de un método que calcula la distancia entre dos nudos hermanos de un árbol, cuyos valores se pasan al método mediante parámetros, `a` y `b`, estando `b` a la derecha de `a`. La distancia es la cantidad de hermanos situados entre ambos, más uno. Si el nudo `a` no está en el árbol, se lanzará `NoExiste`. Si el nudo `b` no se encuentra entre los hermanos situados a la derecha de `a` se lanzará `NoSonHermanos`.

   Para el acceso al árbol se dispone de las operaciones del tipo abstracto de datos *árbol* visto en clase. Para la implementación se dispone de un método `busca` (ya hecho) al que se le pasa un `elemento` y un iterador de árbol colocado en la raíz, y retorna otro iterador de árbol situado en el nudo que contiene ese `elemento`, o retorna `null` si no se encuentra el `elemento` en el árbol.

```
método estático busca (IteradorDeArbol<E> iterador, E elemento)
    retorna  IteradorDeArbol<E>
```

6) Se dispone de una pila implementada mediante una lista enlazada simple, con nudos dispuestos como se indica en la figura. Escribir el *pseudocódigo* de un método interno a la clase (y por tanto con acceso al atributo `principio` de la pila, y a los atributos `contenido` y `siguiente` de cada nudo) que permita eliminar un elemento cualquiera en una pila enlazada. El elemento se le pasa como parámetro y el método retornará el elemento borrado si lo ha encontrado, o `null` en caso contrario.

Indicar la eficiencia del método diseñado.