ESTRUCTURA DE DATOS Y ALGORITMOS **13-Enero-2011**
Titulación: Ingeniero Técnico en Informática de Gestión **Curso: 2º**
Nombre y apellidos: _____ Nota:

- Escribe tu **nombre y apellidos** en esta hoja e inmediatamente en todas las suplementarias, incluso las de sucio. El no hacerlo puede suponer tu expulsión
- Puedes utilizar el **lápiz** para tus respuestas. No está permitido el uso de apuntes, notas o libros. No puedes tener un **móvil** encendido, ni utilizar cualquier otro **aparato electrónico**.
- **Todos los alumnos implicados en una copia** de un ejercicio **tendrán una nota final de 0**. El alumno es responsable de velar por su examen. Es decir **tanto el que copia como el que se deja copiar (ya sea de manera activa o pasiva) recibirán el mismo castigo sin que exista atenuante alguno**

1. Ejercicio (3 puntos)

a) ¿Cuál es la estructura de datos que utiliza un intérprete de un lenguaje de programación para llevar a cabo la recursión? ¿Por qué?

Una **pila**. La razón fundamental es que se tiene que **mantener el orden de llamadas a los métodos en orden inverso al realizado**. De esa forma, la ejecución de la última llamada (la que está en la cima de la pila) será la primera en terminar (**LIFO**) y el interprete puede volver a la llamada anterior del método, que se encontrará en la cima de la pila una vez que se haya eliminado la información relativa al método finalizado. Para cada llamada se almacenará su estado (parámetros, variables locales, número de línea de la llamada a otro método, …) de ejecución antes de realizar una llamada a otro método (independiente que sea recursivo o no).

b) Suponiendo que un árbol binario representa una expresión aritmética, **describe** que datos se almacenan en los nodos internos y externos. ¿Qué recorrido utilizarías para evaluar la expresión? ¿Por qué?

La mejor opción sería que los operadores (+, -, *, /,…) se almacenen en nodos internos y los operandos (o datos, valores) en nodos externos. A la hora de evaluar la expresión utilizaríamos el recorrido post-orden, porque para poder realizar la operación descrita en un nodo interno antes necesitamos calcular los operandos, que se encuentran en sus respectivos subárbol izquierdo y derecho.

c) ¿Cuál es la ventaja principal de la estructura de datos TablaHash respecto a un array, Vector o Lista enlazada? Y la desventaja principal?

La ventaja principal es que las operaciones principales de una estructura de datos (búsqueda, inserción y eliminación) las realiza en tiempo constante, siempre que el valor utilizado para la búsqueda o eliminación es la clave de la información almacenada en la estructura.
La desventaja principal es que los datos no se pueden mantener ordenados y por lo tanto, en caso de que se necesite está operación será más costosa que si utilizáramos cualquier otra estructura de datos en su versión ordenada.

d) Una lista enlazada ¿Es una estructura de datos lineal o no-lineal?

Una lista enlazada es lineal, porque los datos son almacenados en forma secuencial o un orden lógico. Pero también es una estructura de datos no-lineal, por la forma en que son almacenados en memoria. Exactamente, los datos no se almacenan de forma contigua en memoria.

e) Problema práctico: La gerencia de un taller de autos quiere establecer un orden para el arreglo de los coches según la categoría de fidelidad del cliente y el orden de llegada del coche al taller. Para ello, categorizan a sus clientes de 1 a 5 según el gasto realizado (siendo el 1 el de mayor gasto, el más fiel). Si en un momento se da el caso de que haya más de uno que tengan la misma categoría de fidelidad y esta sea la mayor del momento, entonces el coche será elegido según el orden de llegada. ¿Qué estructura de datos utilizarías para organizar los datos y poder elegir de forma eficiente el próximo coche para ser arreglado? ¿Por qué?

La estructura de datos más conveniente es una cola de prioridad porque sus características se ajustan claramente con las del problema. Exactamente, a la hora de extraer los datos (el coche) almacenados en la estructura se quiere dar una prioridad (la categoría de fidelidad del cliente) para su selección y además, en caso de igualdad interesa que la estructura gestione el orden de inserción de los datos en la estructura.

f) Vamos a suponer que en un mismo programa se han definido dos instancias de la clase iteradora sobre una misma lista enlazada simple. ¿Se pueden producir problemas? En caso afirmativo, explica como o cuando puede ocurrir. En caso negativo, explica cual es la característica principal para que no ocurra ningún problema.

Su uso puede ser problemático, exactamente en caso de que si una de las iteradoras elimina justo el nodo de la lista que está referenciado por la variable *current* de la otra iteradora.

ESTRUCTURA DE DATOS Y ALGORITMOS **13-Enero-2011**
Titulación: Ingeniero Técnico en Informática de Gestión **Curso: 2º**
Nombre y apellidos: _____

- Escribe tu **nombre y apellidos** en esta hoja e inmediatamente en todas las suplementarias, incluso las de sucio. El no hacerlo puede suponer tu expulsión
- Puedes utilizar el **lápiz** para tus respuestas. No está permitido el uso de apuntes, notas o libros. No puedes tener un **móvil** encendido, ni utilizar cualquier otro **aparato electrónico**.
- **Todos los alumnos implicados en una copia** de un ejercicio **tendrán una nota final de 0**. El alumno es responsable de velar por su examen. Es decir **tanto el que copia como el que se deja copiar (ya sea de manera activa o pasiva) recibirán el mismo castigo sin que exista atenuante alguno**

2. *Ejercicio (2 puntos)*

Se pide:

a) **Implementar** el algoritmo **genérico** `insertionSort`.

b) **Implementar** un método que dado un array de *`Persona`*s, ordene el array por el nombre de la persona y en orden descendente, utilizando el algoritmo **genérico** `insertionSort`. La clase *`Persona`* contiene la información del nombre y la edad.

1. opción: usando la interfaz Comparable

```
public class Persona implements Comparable {
  private String nombre;
  private int edad;

  public Persona(String nombre, int edad) {
    this.nombre = nombre;
    this.edad = edad;
  }
  // los metodos get y set de cada variable

  public int compareTo(Object o) {
    Persona p = (Persona) o;
    return this.nombre.compareTo(p.nombre);
  }
}

public class Algoritmos{
 public static void insertionSort(Comparable a[]){
  for(int p=1; p < a.length; p++) {
    Comparable tmp=a[p];
    int j;
    for(j=p; j>0 && tmp.compareTo(a[j-1]) > 0; j--)
        a[j]=a[j-1];
    a[j]=tmp;
  }
 }
}

public class Principal {
  public void ordenarPersonas(Persona p[]){
    Algoritmos.insertionSort(p);
  }
}
```

2. opción: usando la interfaz Comparator

```
import java.util.Comparator;
public class Persona {
  private String nombre;
  private int edad;
  public static final Comparator compararPorNombre;
 static {
  compararPorNombre = new CompararPersonaPorNombre();
 }

  public Persona(String nombre, int edad) {
    this.nombre = nombre;
    this.edad = edad;
  }
  // los metodos get y set de cada variable
}

import java.util.Comparator;
public class CompararPersonaPorNombre implements
Comparator {

  public int compare(Object o1, Object o2) {
    Persona p1 = (Persona) o1;
    Persona p2 = (Persona) o2;
    return p1.getNombre().compareTo(p2.getNombre());
  }
}

public class Algoritmos{
 public static void insertionSort (Object a[],
                                   Comparator comparator) {
   for(int p=1; p < a.length; p++) {
     Object tmp=a[p];
     int j;
     for(j=p; j>0 && comparator.compare(tmp, a[j-1]) > 0; j--)
        a[j]=a[j-1];
      a[j]=tmp;
   }
 }
}

public class Principal{
  public void ordenarPersonas(Persona p[]){
    Algoritmos.insertionSort(p,
                      Persona.compararPorNombre);
  }
}
```

ESTRUCTURA DE DATOS Y ALGORITMOS **13-Enero-2011**
Titulación: Ingeniero Técnico en Informática de Gestión **Curso: 2º**

3. *Ejercicio (2 punto)*

Se quiere ampliar la funcionalidad de las listas doblemente enlazadas, vistas en clase, con el método `eliminarSig`. Dicho método eliminará el elemento situado tras la posición actual (`current`). ¿Cómo se tratan los posibles errores?
Se pide:
a) **Implementar** las clases necesarias (sin los métodos) para representar una lista doblemente enlazada.

b) **Implementar** el método `eliminarSig` y cualquier otro que se utilice.

```
public Object eliminarSig() throws Exception{
  if ((current != null) && (current.next != null)){
    Object obj = current.next.element; 'el contenido del nodo candidato a eliminar
    if (current.next == theList.bottom) 'si el nodo candidato a eliminar es el último
        theList.bottom = current;
    else
        current.next.next.previous = current;
    current.next = current.next.next;
    return obj;
  }
  else if (current != null)
    throws new Exception("El índice 'current' no referencia ningún nodo de la lista")
  else
    throws new Exception("No existe siguiente elemento en la lista")
}
```

c) **Describir** gráficamente las siguientes situaciones, utilizando distintos ejemplos:
1) ¿Cómo se realiza la eliminación?
2) ¿Cómo se tratan los posibles errores?

4. *Ejercicio (3 puntos)*

1. Especificar / Parametrizar

Entrada: A, un árbol binario que representa un árbol binario de autoedición (con coordenadas vacías).
Salida: el mismo árbol binario, pero con las coordenadas calculadas de la siguiente forma: 1) la coordenada `x` se calcula usando el número que le corresponde en el recorrido en orden simétrico (EnOrden); y 2) la coordenada `y` se calcula usando la altura del nodo. Además, se mostrará en pantalla las dimensiones (anchura y altura) del árbol A.

```
public Vector calcularCoordenadas ();
```

Para resolver el problema recursivamente, utilizaremos la definición recursiva de árboles binarios, representado mediante la clase BTNode. Por otro lado, detallaremos mejor la especificación del problema.
Entrada: *A*, un árbol binario que representa un árbol binario de autoedición (con coordenadas vacías) y *un número entero*, que representa el número con el que hay que empezar el recorrido simétrico, es decir, la coordenada *x* del nodo más a la izquierda.
Salida: *dos enteros,* que representan la anchura y la altura del árbol A, es decir, sus dimensiones. Para que Java pueda devolver dos valores definiremos la clase `Dimension`.

```
private Dimension calcularCoordenadas (BTNode A, int sigNumero)

class Dimension {
    public int anchura;
    public int altura;
}
```

2. Diseño

Casos triviales

Si esVacio (A) → **devolver** new Dimension(-1, -1)
Si esHoja (A) → A.raiz.x = sigNumero; A.raiz.y = 0;
devolver new Dimension(0, 0)

Caso General

Si noEsVacio(A) y noEsHoja(A) y esVacio(subarbolIzq(A)) →
A.raiz.x = sigNumero;
dimensionDch = ***calcularCoordenadas*** (subarbolDch(A), A.raiz.x + 1);
A.raiz.y = dimensionDch.altura + 1;
devolver new Dimension(dimensionDch.anchura + 1, A.raiz.y)
Si no noEsVacio(A) y noEsHoja(A) y noEsVacio(subarbolIzq(A)) →
dimensionIzq = ***calcularCoordenadas*** (subarbolIzq(A), sigNumero);
A.raiz.x = dimensionIzq.anchura + 1;
dimensionDch = ***calcularCoordenadas*** (subarbolDch(A), A.raiz.x + 1);
A.raiz.y = maximo(dimensionIzq.altura, dimensionDch.altura) + 1;
devolver new Dimension(dimensionIzq.anchura + dimensionDch.anchura + 1, A.raiz.y);
fin si

3-Implementación

```
public void calcularCoordenadas (){
   Dimension d = calcularCoordenadas (bTree.root, 0);
   System.out.println("Las dimensiones son: ("+d.anchura+"u. x "+d.altura+"u.)");
}

private Dimension calcularCoordenadas (BTNode A, int sigNumero) {
   if (A == null)
      return new Dimension(-1, -1);
   else {
      Coordenada c = (Coordenada) A.getContent();
      if (A.getLeft() == null) {
         c.setX(sigNumero);
         if (A.getRight() == null){
            c.setY(0);
            return new Dimension(0, 0);
         }
         else {
            Dimension dimDch = calcularCoordenadas (A.getRight(), c.getX()+1);
            c.setY(dimDch.altura + 1);
            return new Dimension(dimDch.anchura +1, c.getY());
         }
      }
      else {
         Dimension dimIzq = calcularCoordenadas (A.getLeft(), sigNumero + 1);
         c.setX(dimIzq.anchura +1);
         Dimension dimDch = calcularCoordenadas (A.getRight(), c.getX() + 1);
         c.setY( Math.max(dimensionIzq.altura, dimensionDch.altura) + 1);
         return new Dimension(dimensionIzq.anchura + dimensionDch.anchura + 1,
                              c.getY());
      }
   }
}
```