ESTRUCTURA DE DATOS Y ALGORITMOS **9-Septiembre-2009**
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 **lápiz** para tus respuestas. No está permitido el uso de apuntes o libros. No puedes tener un **móvil** encendido ni utilizar **calculadora**.
- **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 (2 puntos)

a) Un vector tiene la siguiente propiedad:
   1. Inserciones y borrados eficientes en medio de la estructura
   2. Inserciones y borrados eficientes en ambos extremos de la estructura
   3. **Inserciones y borrados eficientes al final de la estructura**
   4. Ninguna de las anteriores

b) Una lista enlazada simple tiene la siguiente propiedad:
   1. **Acceso secuencial**
   2. Acceso bidireccional
   3. Acceso aleatorio
   4. Todas las anteriores

c) Cual de las siguientes afirmaciones es cierta:
   1. Las estructuras de datos almacenan operaciones y las aplican sobre un objeto
   2. Las estructuras de datos restringen el uso de las variables
   3. **Las estructuras de datos almacenan referencias a objetos**
   4. Las estructuras de datos almacenan variables de objetos

d) Las principales diferencias entre las distintas estructuras son:
   1. La forma de organizar las operaciones y las restricciones que imponen sobre los datos
   2. **La forma de organizar los datos y las restricciones que imponen sobre las operaciones**
   3. La forma en que aplican la abstracción sobre variables de objetos
   4. En principio, no existe ninguna diferencia sustancial, depende del problema planteado

ESTRUCTURA DE DATOS Y ALGORITMOS **9-Septiembre-2009**
Titulación: Ingeniero Técnico en Informática de Gestión **Curso: 2º**
Nombre y apellidos: _____

2. *Ejercicio (4 puntos)*

Se pide:

a) (*1 punto*)

```
public class Resolver{
      Hashtable variables = new Hashtable();

      public void anadirVariable(String nombre, int valor){
            Variable var = new Variable(nombre, valor);
            variables.put(var.getKey(), var.getValue());
      }
      public boolean esVariable(String nombre){
            Variable var = new Variable(nombre, 0);
            return variables.containsKey(var.getKey());
      }
      public int variable(String nombre){
            Variable var = new Variable(nombre, 0);
            Integer i = (Integer) variables.get(var.getKey());
            return i.intValue();
      }
}
public class Variable implements Entry{
      private String nombre;
      private Integer valor;
      Variable (String nombre, int valor){
            this.nombre = nombre;
            this.valor = new Integer(valor);
      }
      public Object getKey(){
            return nombre;
      }
      public Object getValue(){
            return valor;
      }
      public Object setValue(Object newValue){
            Integer old = valor;
            valor = newValue;
            return old;
      }
}
```

b) (*3 puntos*)

1. Especificar / Parametrizar

Entrada: A, un arbol binario.
Salida: muestra en pantalla el resultado de la evaluación o el punto dónde se ha producido el error

```
public void evaluar(Resolver resolver);
```

A la hora de tratar un nodo del árbol y poder evaluarlo, debemos de conocer: 1) si el nodo actual es un número entonces debe ser hoja; 2) si el nodo actual es un operador entonces debe de tener dos operandos; y 3) los operandos deben de ser evaluados antes del nodo actual y además, comprobar que no hay ningún error para poder seguir con la evaluación o en caso contrario mostrar el error, indicando donde se ha producido. → INMERSION

Entrada: A, un arbol binario y expresion, una cadena de caracteres que representa la expresión evaluada hasta el momento.
Salida: tres valores: una cadena de caracteres, un número real y otra cadena de caracteres. El primero almacenará el mensaje de error en caso se producirse. El segundo representará el resultado de la expresión evaluada hasta el momento y el tercero representará la expresión evaluada hasta el momento .

Se necesita una clase Contexto para poder devolver tres valores:

```
public class Contexto {
    String error;
    float resultado;
    String expresion;
}
```

```
private Contexto evaluar1(Resolver resolver, BTNode A, String expresion);
```

2. Diseño

Casos triviales

Si esVacio (A) → devolver new Contexto(null, 0, "")
Si esHoja (A) y esNumero(raiz(A)) →
    devolver new Contexto(null, raiz(A), expresión+raiz(A))
Si esHoja (A) y esVariable(raiz(A)) →
    devolver new Contexto(null, resolverVariable(raiz(A)), expresión+raiz(A))
Si esHoja (A) y no(esNumero(raiz(A))) y no(esVariable(raiz(A))) y no(esOperador(raiz(A))) →
    devolver new Contexto("simbolo desconocido", 0, expresión+raiz(A))
Si esOperador(raiz(A)) y menos de dos hijos(A) →
    devolver new Contexto("el operador "+raiz(A)+" requiere de 2 operandos",
        0, expresión+"("+raiz(A)+")")

Caso General

Si esOperador(raiz(A)) y tiene dos hijos(A) →
  error = null;
  resultado = 0;
  expresionActual = expresion + "(";
  contextoIzq = ***evaluar1***(subarbolIzq(A), expresionActual);
  si (contextoIzq.error == null) entonces
    expresionActual = contextoIzq.expresion + raiz(A);
    contextoDch = ***evaluar1*** (subarbolDch(A), expresionActual);
    si (contextoDch.error == null) **entonces**
      expresionActual = contextoDch.expresion + ")";
      resultado = Expr.evaluar(raiz(A), contextoIzq.resultado, contextoDch.resultado)
    si no
      error = contextoDch.error
      expresionActual = contextoDch.expresion
    fin si
  si no
    error = contextoIzq.error
    expresionActual = contextoIzq.expresion
  fin si
  devolver new Contexto(error, resultado, expresionActual)

3-Implementación

```
public void evaluar() {
      Contexto contexto = evaluar1 (bTree.root, "");
      if (contexto.error == null)
            System.print.out("Resultado: "+contexto.resultado);
      else {
            System.out.println("Error: "+contexto.error);
            System.out.println("Mensaje: "+contexto.expresion);
      }
}

private Contexto evaluar1(Resolver resolver, BTNode A, String expresion) {
      String error, expresionActual;
      float resultado;

      if (A== null)
        return new Contexto(null, 0, "");
      else if (A.left==null && A.right==null){
        if (Expr.esNumero(A.content))
          return new Contexto(null, A. content, expression+A.content);
        else if (Expr.esVariable(A.content)
          return new Contexto(null, resolver.variable(A.content),
                              expression+A.content);
        else if (!Expr.esOperador(A.content))
          return new Contexto("simbolo desconocido", 0, expression+A.content);
      }
      else if (Expr.esOperador(A.content) {
         if ((A.left!=null && A.right==null) || (A.left==null && A.right!=null))
             return new Contexto("el operador "+A.content+" require 2 operandos", 0,
                                 expression+"("+A.content+")");
         else if (A.left!=null && A.right!=null){
            error = null;
            resultado = 0;
            expresionActual = expression + "(";
            Contexto contextoIzq = evaluar1(resolver, A.left, expresionActual);
            if (contextoIzq.error == null) {
               expresionActual = contextoIzq.expresion + A.content;
               contextoDch = evaluar1 (resolver, A.right, expresionActual);
               if (contextoDch.error == null) {
                  expresionActual = contextoDch.expresion + ")";
                  resultado = Expr.evaluar(A.content, contextoIzq.resultado,
                                           contextoDch.resultado);
               }
               else {
                  error = contextoDch.error;
                  expresionActual = contextoDch.expresion;
               }
            }
            else {
               error = contextoIzq.error;
               expresionActual = contextoIzq.expresion;
            }
            return new Contexto(error, resultado, expresionActual);
        }
      }
 }

public class Contexto {
      String error, expresion;
      float resultado;
      public Contexto(String error, float resultado, String expresion) {...}
}
```

3. *Ejercicio (4 puntos)*

a) (1 punto)

```
public class ERE {
        public static int NUM_CATEGORIAS = 5
        Queue[] trabajando = new Queue[NUM_CATEGORIAS+1]; //Defino un array de colas, uno por categoría.
                                                          // En las colas almacenaremos instancias de la clase EmpleadoFecha.
        Queue[] enParo = new Queue[NUM_CATEGORIAS+1];

        public ERE(){
                for(int i=1; i<=NUM_CATEGORIAS;i++){
                        trabajando[i] = new Queue();
                        enParo[i] = new Queue();
                }
        }

        public void inicializar(String fichero){
          Vector[] empleados = new Vector[NUM_CATEGORIAS+1];
          for(int i=1; i<=NUM_CATEGORIAS;i++)
                empleados[i] = new Vector();

          // almacenar los datos en vectores para ordenarlos.
          Token datos = new Token(fichero);
          while (datos.hasMoreTokens()){
                Empleado empleado = datos.getToken();
                Empleados[empleado.getCategoria()].addElement(empleado);
          }
          // ordenar los datos de los vectores utilizando un algoritmo de ordenación
          for(int i=1; i<=NUM_CATEGORIAS;i++){
                Comparable[] empArray = (Comparable[]) empleados[i].toArray(new Empleado[empleados[i].size());
                selectionSort(empArray); // se podría utilizar cualquier otro algoritmo de ordenación
                inicializarTrabajando(trabajando[i], empArray);
          }
        }

        private inicializarTrabajando (Queue trabajando, Object[] empleados){
           for(int i=0; i < empleados.length(); i++){
              EmpleadoFecha empleadoFecha = new EmpleadoFecha ((Empleado)empleados[i], hoy());
              trabajando.enqueue(empleadoFecha);
           }
        }

        public static void selectionSort(Comparable[] datos){
             //implementación en los apuntes
        }

   Public class EmpleadoFecha{
      …
   }
```

b) **Implementa** los métodos *inicializar* y *enviarAlParo* de la clase *ERE*.

```
        public void enviarAlParo(int numero){
          int cont = 0;
          while (cont < numero && sePuedeEnviarAlParoCategoria(1)){
            i = NUM_CATEGORIAS;
            while (i>1 && (!sePuedeEnviarAlParoCategoria(i) || !mantieneProporcion(i)))
                    i--;
           //Por lo menos se envia a una persona
            enviarEmpleadoAlParo(i);
            cont++;
            // se puede enviar a más empleados de la misma categoría, si no hay
            // empleados de categorías inferiores que puedan ser enviadas al paro
             If (!sePuedeEnviarAlParoCategoria(i+1)){
                    while ( cont<numero && sePuedeEnviarAlParoCategoria(i) && mantieneProporcion(i)){
                        enviarEmpleadoAlParo(i);
                        cont++;
                     }
             }
        }

        private boolean sePuedeEnviarAlParoCategoria(int categoria){
          if (categoria < 1 || categoria > NUM_CATEGORIAS)
            return false;
          else
            return (hayEmpleadosTrabajando(categoria) &&
                            cumplenCondicionEnvioAlParo(categoria));
        }

        private boolean hayEmpleadosTrabajando (int categoria){
          if (categoria < 1 || categoria > NUM_CATEGORIAS)
            return false;
          else
            return trabajando[categoria].size() > 0;

        private boolean cumplenCondicionEnvioAlParo(int categoria){
          if (categoria < 1 || categoria > NUM_CATEGORIAS)
            return false;
          else
            Date fecha = ((EmpleadoFecha)trabajando[categoria].front()).getFecha();
            return (diferenciaDias(fecha, hoy) >= 15);
        }

        private boolean mantieneProporcion(int categoria){
          if (categoria <= 1 || categoria > NUM_CATEGORIAS )
                    return true;
          else
                    return trabajando[categoria].size() >
                              trabajando[categoria-1].size()*categoria;
        }

        private void enviarEmpleadoAlParo(int categoria){
                    EmpleadoFecha empFecha = (EmpleadoFecha) trabajando[categoria].dequeue();
                    empFecha.modificarFecha(hoy());
                    enParo[categoria].enqueue(empFecha);
        }
  }
```