Solución al Examen de Prácticas de Programación (Ingeniería Informática)

Junio 2006

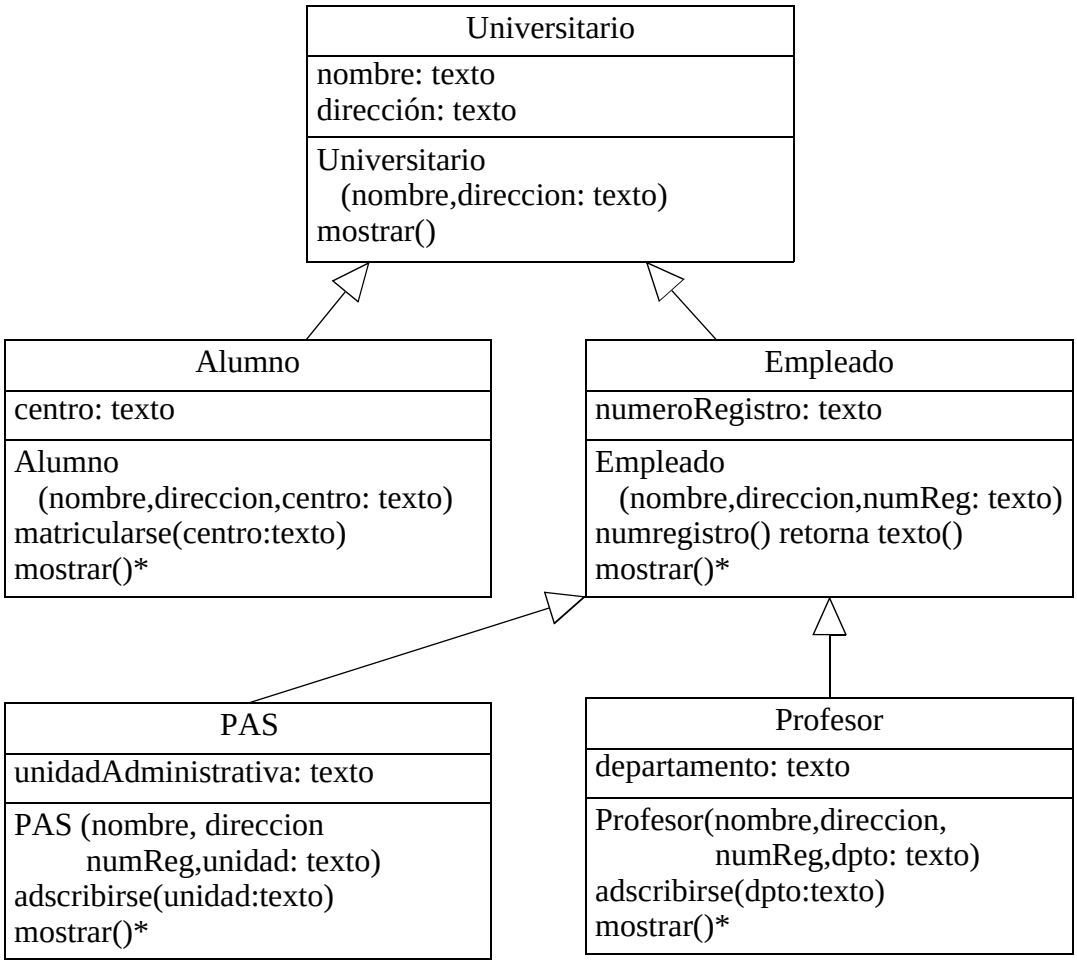
Parte I. Cuestiones (3 puntos=50% nota del examen)

1) Se desea crear un conjunto de clases para representar en un programa las personas pertenecientes a la Universidad. Existen tres tipos de universitarios: alumnos, profesores, y personal de administración y servicios (PAS). Los dos últimos son empleados de la Universidad (los alumnos no).

   Se pide hacer el diseño arquitectónico de una jerarquía de clases que representen a estas personas, teniendo en cuenta los siguientes datos (todos de tipo texto) y operaciones:
   - Todos los universitarios tienen nombre y dirección, y operación para mostrar datos
   - Los alumnos tienen centro y operación para matricularse en un centro
   - Los profesores tienen departamento y operación para adscribirse a él
   - Los PAS tienen unidad administrativa y operación para adscribirse a ella
   - Los empleados tienen número de registro y operación para obtenerlo

   Se pide también el diseño detallado de los constructores de cada clase, a los que se pasarán como parámetros los valores iniciales de cada atributo.

Diseño arquitectónico



* operación redefinida

Diseño detallado de los constructores

```
método Universitario(nombre,direccion : texto)
   this.nombre:=nombre;
   this.direccion:=direccion;
fmétodo

método Alumno(nombre,direccion,centro : texto)
   super(nombre,direccion);
   this.centro:=centro;
fmétodo

método Empleado(nombre,direccion,numRegistro : texto)
   super(nombre,direccion);
   this.numeroRegistro:=numRegistro;
fmétodo

método PAS(nombre,direccion,numReg,unidad : texto)
   super(nombre,direccion,numReg);
   this.unidadAdministrativa:=unidad;
fmétodo

método Alumno(nombre,direccion,numReg,dpto : texto)
   super(nombre,direccion,numReg);
   this.departamento:=dpto;
fmétodo
```

2) Decimos que una pila *p* es sombrero de otra pila *q* si todos los elementos de *p* están en *q* en el mismo orden y en las posiciones más próximas a la cima. La pila nula se considera sombrero de cualquier otra pila. Por ejemplo, si *A*:=[7,9,4,2], *B*:=[7,9,4,2,5,3,6] y *C*:=[7,9,2,4,5,3,6], entonces *A* es sombrero de la pila *B*, pero no de la pila *C*.

   Se supone que el tipo genérico `elemento` dispone de una operación de comparación: `x.equals(y)` que aplicado a dos elementos `x` e `y` retorna cierto si y sólo si `x` e `y` son iguales. Se pide:

   - Especificar y diseñar un método estático recursivo `esSombrero(p,q)` que determine si la pila `p` es sombrero de la pila `q`.
   - Diseñar el método anterior mediante un bucle.

definimos la función esSombrero como sigue:

esSombrero(p.pilaVacia(),q)=cierto,
esSombrero(p.apilar(x),q)=falso si q.esVacia(),
esSombreo(p.apilar(x),q.apilar(y))= (x=y) && (esSombrero(p,q)).

Ahora implementamos la función esSombrero mediante un método impEsSombrero:

```
método estático impEsSombrero(p, q: Pila) retorna esSomb: booleano
{Pre.- p=P && q=Q}
   si p.esVacia() entonces
         esSomb:= cierto
   si no
          si q.esVacia() entonces
                esSomb:= falso
           si no
                esSomb:=(p.cima().equals(q.cima()))&&
                            impEsSombrero(p.desapilar(),q.desapilar());
           fsi
   fsi
   retorna esSomb;
{Post.- b=esSombrero(P,Q)}
fmétodo
```

Finalmente, implementamos la misma función de forma iterativa

```
método estático esSombreroIterativo(p,q : pila) retorna esSomb:booleano
     esSomb:=cierto
     mientras no p.esVacia() && esSomb  hacer
       si q.esVacia() entonces
              esSomb:=falso
       sino
              esSomb:= cima(p).equals(cima(q))
              p.desapilar()
              q.desapilar()
       fsi
     fmientras
     retorna esSomb
fmétodo
```

* Nota: Ambos métodos modifican las pilas p y q. Si fuese necesario no modificarlas, habría que copiarlas (clonarlas) y trabajar con las copias, o usar algún tipo de iterador

3) Se dispone de la siguiente clase ya realizada, que representa una red de comunicación:

```
/** Clase que contiene una operación para recibir
 *  mensajes de texto por un canal de comunicación */
public class CanalComunicacion
{
```

```
    /**
     * Recibir un mensaje de texto del puerto y emisor indicados
     */
    public String recibe(int puerto, int emisor) throws NoDisponible
    {...}
}
```

Se pide escribir un método para esa clase, con los mismos parámetros y valor de retorno que `recibe()` y que llame a éste un máximo de 5 veces (usando en la llamada los parámetros obtenidos). Si alguna de las llamadas tiene éxito (porque no se lanza `NoDisponible`), finalizar el método retornando el valor retornado a su vez por `recibe()`. Después del 5 intento sin éxito, lanzar la excepción `Imposible`. Ambas excepciones están declaradas en clases aparte en la forma:

```
public class NoDisponible extends Exception {}
public class Imposible extends Exception {}
```

```
    public String recibeReintentando (int puerto, int emisor)
       throws Imposible
    {
        // 5 reintentos
        for (int i=0;i<5;i++) {
           try {
               // si va bien, finaliza el metodo
               return recibe(puerto,emisor);
           } catch (NoDisponible e) {
               // Si falla no hacemos nada
           }
        }
        // si llega aquí ha fallado 5 veces
        throw new Imposible();
    }
```

---

4) Escribir en Java la implementación del método `inserta()` de la clase siguiente:

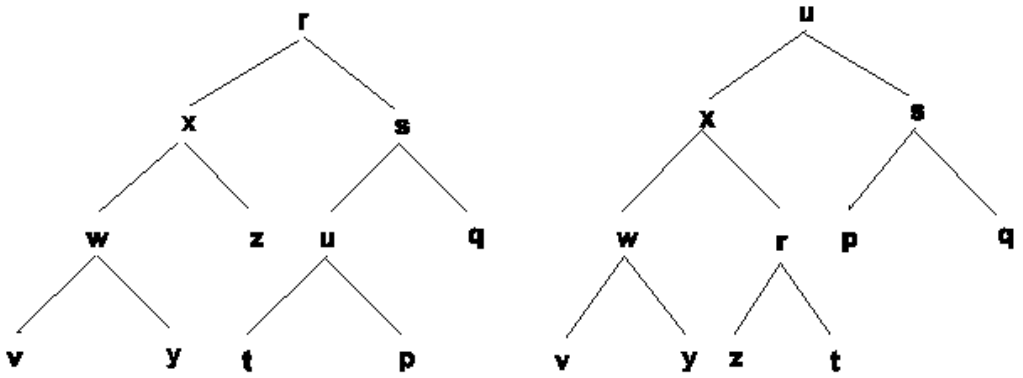
```
/**
 * Clase que representa una lista de nombres
 */
public class ListaNombres
{
    // atributo, que contiene la lista
    private LinkedList<String> lista=new LinkedList<String>();

    /**
     * Inserta sin repeticiones los strings del array nombres
     * en la lista, añadiéndolos uno por uno al final. Aquellos que
     * ya están en la lista no se añaden.
     */
    public void inserta(String[] nombres)
    {... }
}
```

```
    public void inserta(String[] nombres)
    {
       // recorrido del array nombres
       for (String s:nombres) {
          // lo añade si no está ya incluido
          if (!lista.contains(s)) {
              lista.addLast(s);
          }
       }
    }
```

---

5) Un árbol binario (como el *A1* o el *A2* de la figura) puede recorrerse de muchas formas, en particular en inorden, es decir, primero se lista el hijo izquierdo en inorden, luego la raíz y luego el hijo derecho en inorden. Esta definición recursiva se completa definiendo el inorden del árbol vacío como la lista vacía. En la figura se muestra a modo de ejemplo el recorrido en inorden de *A1*.



A1

A2

{v,w,y,x,z,r,t,u,p,s,q}

Responder a la siguiente pregunta: ¿puede recuperarse un árbol binario conociendo la lista de sus nodos en inorden?. Si la respuesta es negativa exhibir un contraejemplo. Si es afirmativa argumentar la razón.

La lista de nodos en inorden no permite recuperar el árbol binario, tal como puede verse en el contraejemplo de los árboles A1 y A2 de la figura. El desarrollo en inorden de A2 es {v,w,y,x,z,r,t,u,p,s,q}, idéntico al de A1. Sin embargo, A1 y A2 son árboles diferentes.

Examen de Prácticas de Programación (Ingeniería Informática)

Junio 2006

Parte II. Problema (3 puntos=50% nota del examen)

Se desea hacer parte del programa de gestión del inventario de los artículos que se venden en una tienda. Para ello, se dispone de la siguiente clase ya realizada, cuya interfaz es:

```
/**
 * Clase que representa un artículo que se vende en una tienda
 */
public class Articulo
{
    /**
     * Constructor al que se le pasan los datos del artículo
     */
    public Articulo(String nombre, int cantidad, double precioUnidad) {...}

    /**
     * Retorna el nombre
     */
    public String nombre() {...}

    /**
     * Retorna el número de unidades disponibles
     */
    public int cantidad() {...}

    /**
     * Retorna el precio de una unidad
     */
    public double precioUnidad() {...}

    /**
     * Vende num artículos (restándolos de la cantidad disponible), si los
     * hay. En caso contrario, no modifica la cantidad y
     * lanza NoDisponible.
     */
    public void vende(int num) throws NoDisponible {...}
}
```

Lo que se pide es hacer una clase que almacene la lista de artículos y que responda a la siguiente interfaz

```
/**
 * Clase que contiene una lista con los artículos en venta en una tienda
 */
public class Tienda
{
    /**
     * Constructor al que se le pasa el nombre de la tienda
     * y que deja la lista vacía
     */
    public Tienda(String nombre) {...}

    /**
     * Añade un artículo a la lista
     */
    public void inserta (Articulo a) {...}

    /**
     * Busca y retorna el artículo cuyo nombre se indica
     * Lanza NoExiste si no se encuentra
     */
    public Articulo busca (String nombre) throws NoExiste {...}
```

```
    /**
     * Vende num unidades del artículo del nombre indicado. Para ello busca
     * el articulo en la lista y usa su método vende. Si no encuentra
     * el artículo lanza NoExiste. Si la cantidad queda a cero o si se ha
     * lanzado NoDisponible, elimina el artículo de la lista. Retorna el
     * valor total de las unidades vendidas.
     */
    public double vende (String nombre, int num) throws NoExiste {...}

    /**
     * Escribe en el fichero de texto cuyo nombre se indica
     * aquellos artículos de la lista de los que queden menos
     * de min unidades. Se escriben uno por línea, cada uno
     * con sus tres datos. En la primera línea se escribirá el
     * nombre de la tienda
     */
    public void escribePedido(String nombreFichero, int min)  {...}

}
```

La clase tendrá como atributos privados una lista de artículos (usar por ejemplo la clase `LinkedList`), y el nombre de la tienda (texto).

Las excepciones están declaradas en clases aparte en la forma:

```
public class NoDisponible extends Exception {}
public class NoExiste extends Exception {}
```

Nota: Se valorará cada parte de la clase `Tienda` según su dificultad.

```
import java.io.*;
import java.util.*;
/**
 * Clase que contiene los artículos en venta en una tienda
 */
public class Tienda
{
    // atributos
    private LinkedList<Articulo> lista;
    private String nombre;

    /**
     * Constructor al que se le pasa el nombre de la tienda
     */
    public Tienda(String nombre)
    {
        this.nombre=nombre;
        lista= new LinkedList<Articulo>();
    }

    /**
     * Añade un artículo a la lista
     */
    public void inserta (Articulo a)
    {
        lista.add(a);
    }

    /**
     * Busca y retorna el artículo cuyo nombre se indica
     * Lanza NoExiste si no se encuentra
     */
```

```java
public Articulo busca (String nombre) throws NoExiste
{
    int i=0;
    boolean encontrado=false;
    Articulo a=null;
    // esquema de busqueda en tabla
    while (i<lista.size() && ! encontrado) {
        a=lista.get(i);
        if (nombre.equals(a.nombre())) {
            encontrado=true;
        } else {
            i++;
        }
    }
    // retorna valor o lanza excepcion
    if (encontrado) {
        return a;
    } else {
        throw new NoExiste();
    }
}

/**
 * Vende num unidades del artículo del nombre indicado. Para ello busca
 * el articulo en la lista y usa su método vende. Si no encuentra
 * el artículo lanza NoExiste. Si la cantidad queda a cero o si se ha
 * lanzado NoDisponible, elimina el artículo de la lista. Retorna el
 * valor total de las unidades vendidas.
 */
public double vende (String nombre, int num) throws NoExiste
{
    // No hace falta lanzar NoExiste, pues ya lo hace busca
    Articulo a=busca(nombre);
    int disponible=a.cantidad();
    int vendidos=num;
    try {
        // vende las unidades pedidas
        a.vende(num);
        if (disponible==num) {
            // ya no quedan articulos
            lista.remove(a);
        }
    } catch (NoDisponible e) {
         // no habia suficientes
         vendidos=disponible;
         lista.remove(a);
    }
    return vendidos*a.precioUnidad();
}

/**
 * Escribe en el fichero de texto cuyo nombre se indica
 * aquellos artículos de la lista de los que queden menos
 * de min unidades. Se escriben uno por línea, cada uno
 * con sus tres datos. En la primera línea se escribirá el
 * nombre de la tienda
 */
public void escribePedido(String nombreFichero, int min)
{
    try {
        // Abrir el fichero
        FileWriter f = new FileWriter(nombreFichero);
        PrintWriter p = new PrintWriter(f);
        // Escribir la primera linea
        p.println("Nombre tienda: "+nombre);
        // recorrer todos los articulos
        Articulo a;
        for (int i=0; i<lista.size(); i++) {
            a=lista.get(i);
            // escribir el articulo si la cantidad es menor que min
```

```
                if (a.cantidad()<min) {
                    p.println("Articulo:"+a.nombre()+
                              " cantidad:"+a.cantidad()+
                              " precio:"+a.precioUnidad());
                }
            }
            // Cerrar el fichero
            p.close();
        } catch (IOException e) {
            System.out.println("Error inesperado: "+e);
        }
    }
}
```