Técnicas de Desarrollo de Programas
Ingeniería Informática
Curso 2008 / 2009

Ejercicios de Patrones de Diseño:

Handler, Adapter, Iterator, Composite, Patrones de Creación, Strategy

Ejercicio 1 (examen de septiembre año 2003/04)

Una empresa dedicada al transporte de mercancías ofrece a sus clientes que elijan el transportista que va a hacer efectivo su envío. Cada transportista calcula el coste del envío de distintas maneras. Por ejemplo, el envío estándar establece una tarifa por kilo para los envíos locales y otra tarifa distinta para los envíos de larga distancia, mientras que otros transportistas, como UPS, FedEx, etc. pueden hacer el cálculo de la tarifa teniendo en cuenta solamente el peso del paquete o cualquier otra política distinta para llevar a cabo el mismo cálculo.

Se pide:

1. Criticar el código que aparece a continuación, teniendo en cuenta que en el futuro se prevé añadir nuevos transportistas.

```
public class Envio {

	private String _origen;
	private String _destino;
	private float _peso;

	public Envio(String origen, String destino, float peso){
		_origen = origen;
		_destino = destino;
		_peso = peso;
	}

	/**
	* Calcula el precio del envío del paquete.
	* @param tipo: 1- Estandar, 2- FedEx, 3-UPS
	* @return Coste del envío con el transportista indicado.
	*/
	public float getPrecio(int tipo){
		switch (tipo){
			case 2:  return calcularFedEx();
			case 3:  return calcularUPS();
			case 1:  // siguiente
			default: return calcularEstandar();
		}
	}

	private float calcularEstandar(){
		float precioLocalPorKilo = 0.089f;
		float precioLargaDistanciaPorKilo = 0.123f;
		float precioTotal = 0;
		if (_destino.equals(_origen)) {
			precioTotal = precioLocalPorKilo;
		}
```

```
                else {
                        precioTotal = precioLargaDistanciaPorKilo;
                }
                return peso * precioTotal;
        }

        private float calcularFedEx(){
                float costeInicial = 5.00f;
                float costePorKilo = 1.00f;
                float multiplicadorLargaDistancia = 1.33f;
                float coste;
                if (_origen.equals(_destino)){
                        coste = costeInicial + costePorKilo * peso;
                }
                else {
                        coste = costeInicial +
                        (costePorKilo*peso) * multiplicadorLargaDistancia;
                }
                return coste;
        }

        private float calcularUPS(){
                float costePorKilo = 2.00f;
                return costePorKilo * peso;
        }
}

public class CalculadorEnvio {

        public static void main(String[] args) {
                // Un cliente envía un paquete con 8.7 kilos de peso de
                // Madrid a Madrid, y quiere saber su coste con cada
                // método de envío
                Envio envio = new Envio("MA", "MA", 8.7f);
                System.out.println("Coste con envío estandar:" +
                                        envio.getPrecio(1));
                System.out.println("Coste con envío FedEx :" +
                                        envio.getPrecio(2));
                System.out.println("Coste con envío UPS : " +
                                        envio.getPrecio(3));
        }
}
```

2. Implementar otra solución utilizando patrones de diseño. Incluir el diagrama UML con un breve comentario y la implementación.
3. Actualmente sólo se necesita conocer el peso, el origen y el destino del paquete para calcular su coste. Se prevé que en el futuro estos parámetros se puedan ver incrementados en nuevos métodos de cálculo del coste. ¿Qué alternativas ofrece el patrón para la comunicación de estos datos a los distintos métodos de cálculo del coste? ¿Qué ventajas e inconvenientes presenta cada una de ellas?

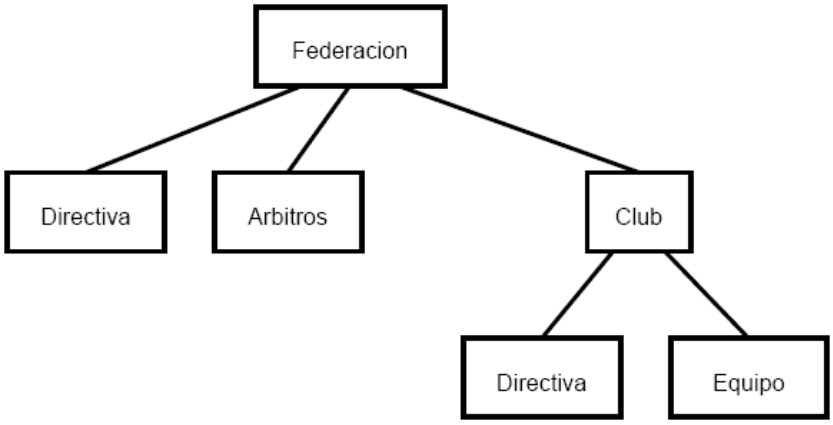
Ejercicio 2 (examen de junio año 2004/05)

La Federación Española de Canicas tiene un fichero con todas las personas que participan en sus actividades como directivos, árbitros y jugadores, pero tiene todas las fichas mezcladas en el mismo cajón. En un momento de lucidez deciden informatizar el sistema organizando a todos los participantes en equipos arbitrales, clubs, equipos de jugadores y equipos directivos.

Se pide:

1. Proponer, utilizando patrones de diseño, una manera uniforme de gestionar las fichas de todos los componentes de la reestructuración. Dibujar el diagrama UML correspondiente a partir de las siguientes clases.

2. Escribir en Java una manera de controlar que la jerarquía de la siguiente figura es la única posible y que, por ejemplo, un equipo de jugadores no puede contener un club.



3. La Federación sospecha de algunos componentes de los equipos arbitrales, por lo que decide investigar si alguno de ellos es familiar de los componentes del Club “Los Fenix del gua”, cuyo primer equipo gana sospechosamente demasiados partidos. Utilizando iteradores, definir dos posibles formas de control de los mismos para que la Federación pueda averiguar de manera sencilla dicha información.

Ejercicio 3 (examen de junio año 2004/05)

La gestión de la red eléctrica se fundamenta en el control de las instalaciones eléctricas que la conforman. A fin de poder registrarlas, toda instalación dispone de un identificador cuyo formato puede variar según el tipo de instalación, la empresa propietaria o la legislación vigente.

Dada la relevancia de las instalaciones, y su influencia en todas las labores de distribución de energía, la compañía eléctrica HISPALUZ S.A. ha contratado nuestros servicios para el desarrollo de una librería de clases que permita la gestión homogénea de las mismas. Los requisitos impuestos por el cliente son:

- Existe una base de datos de instalaciones que mantiene las características eléctricas y de conexión de cada una de las instalaciones de la red.
- Las instalaciones se clasifican en alguno de los siguientes tipos:
    a. Subestación. Desciende el nivel de potencia a Media Tensión. Identificadas mediante una combinación de tres letras denominada matrícula.
    b. Centro de Transformación (desde ahora, CT). Convierte la energía de Media a Baja Tensión. Tiene un identificador numérico de seis dígitos.
    c. Transformador (desde ahora, Trafo). Identificado mediante una matrícula de ocho dígitos.

  La empresa HISPALUZ S.A. dispone de participaciones en compañías eléctricas de otras latitudes, por lo que debe contemplar la posibilidad de modificaciones en las características de los identificadores.
- Las características de las instalaciones se representan utilizando los siguientes componentes:
    a. Etiqueta. Determina el tipo de instalación mediante un literal.
    b. Símbolo representativo. Caracteriza de manera inequívoca el tipo de instalación.
- Dependiendo de la aplicación que utilice la librería de clases los componentes pueden crearse en uno de los siguientes modos:
    a. Texto. Utilizado por aplicaciones de gestión interesadas únicamente en los valores alfanuméricos que definen las características eléctricas de la instalación.
    b. Gráfico. Pensado para aplicaciones desarrolladas en cliente pesado, permite una representación gráfica de la instalación.

  El cliente ha expuesto su intención de ampliar estas categorías en el futuro definiendo, por ejemplo, modos para trabajar con cliente ligero o dispositivos móviles.
- La creación de las instalaciones deberá estar centralizada en un único punto a fin de facilitar labores de auditoría o control sobre el tipo y número de instalaciones utilizadas.

Tomando en consideración estos requisitos, se pide:

1. Enumerar tres patrones de diseño que permitirían el desarrollo óptimo de la librería de clases y explicar brevemente el porqué de su elección.
2. Modificar el siguiente código a fin de adaptarlo al uso del patrón de diseño correspondiente. Si fuera necesario añadir alguna clase para la correcta resolución del problema deberá incluirse su código.

```
public interface Instalacion {

	public String obtenerIdentificador();
	public boolean esIgual(Instalacion otra);
	public String toString();
}

public class Subestacion implements Instalacion {

	private String _matricula;

	public Subestacion(String matricula) {
		_matricula = matricula;
	}

	public String obtenerIdentificador() {
		return _matricula;
	}

	public boolean esIgual(Instalacion otra) {
		return _matricula.equals(otra.obtenerIdentificador());
	}

	public String toString() {
		return ("Matricula: " + _matricula);
	}
}
```

3. Especificar en UML el diseño del patrón (o patrones) de creación utilizados por la librería de clases.

Ejercicio 4 (examen parcial año 2007/08)

Un banco desea desarrollar una aplicación para gestionar las hipotecas que contratan sus clientes. Por cada hipoteca les cobra una cuota periódica cuya cuenta de cargo es una cuenta abierta en el mismo banco. Cada cuenta tiene un único titular y está identificada por un código de 20 dígitos. Cada cliente puede tener varias hipotecas (por ejemplo una para la casa y otra para el coche) que pueden estar asociadas a la misma cuenta o a cuentas distintas.

Las hipotecas tienen un identificador que coincide con el número de cuenta al que están asociadas. Al contratar una hipoteca se debe indicar el importe del préstamo que realiza el banco (*deuda primaria*), la tasa de interés anual que cobrará al cliente, y el número de años de duración de la hipoteca. Las cuotas que el cliente deberá pagar se calculan mediante uno de los dos siguientes sistemas de amortización, que el cliente decide cuando contrata la hipoteca: sistema de amortización francés o sistema de amortización alemán. En el sistema francés las cuotas se pagan mensualmente, y su cuantía depende de cuatro factores: la deuda pendiente, la tasa de interés, el número de meses que dura el préstamo y el número que hace la cuota (esto es, 1 si es la primera cuota a pagar, 2 si es la segunda, y *n* si es la *n*-ésima cuota). En el sistema alemán las cuotas se pagan anualmente y su cuantía depende de tres factores: la deuda pendiente, la tasa de interés y el número de años que dura el préstamo.

La aplicación a desarrollar debe permitir:

- calcular la siguiente cuota que un cliente debe pagar, y detraerla del saldo de la cuenta asociada, operación conocida como *amortizar*.
- imprimir por pantalla las condiciones de contratación de una hipoteca dada.
- imprimir por pantalla los datos de aquellos clientes que tienen contratada una hipoteca.
- imprimir por pantalla los datos de los clientes hipotecados morosos (con una hipoteca y saldo negativo en la cuenta asociada).

Está previsto que el módulo para el cálculo de hipotecas se integre en un futuro próximo en el portal web que el banco posee, de tal modo que los clientes puedan realizar la simulación de sus hipotecas cómodamente desde sus casas.

Se pide:

1. Realizar el diagrama de clases UML que constituye el diseño óptimo de esta aplicación.
2. Indicar qué patrones de diseño se han utilizado en el apartado 1), en caso de haber usado alguno. ¿Qué beneficios aporta el uso de tales patrones en esta aplicación concreta?
3. Realizar el diagrama de secuencia de la creación de una hipoteca con sistema de amortización francés, y la amortización de su primera cuota (operación *amortizar*).
4. Especificar el código en java de la clase Hipoteca.

Nota: documenta cualquier suposición sobre la aplicación que no esté en el enunciado.

Ejercicio 5 (examen parcial año 2007/08)

Un almacén desea desarrollar una aplicación para gestionar los pedidos que realizan sus clientes. Al realizar un pedido, el cliente indica al almacén sus datos personales (nombre, dirección, teléfono y e-mail), así como el número de unidades de cada uno de los productos de los que consta su pedido.

Para minimizar el tiempo de espera desde que el cliente realiza un pedido hasta que lo recibe, el almacén sigue la siguiente política de entregas. Cuado un cliente realiza un pedido, se le permite agrupar los productos de que consta el pedido en pedidos parciales (*subpedidos*) que se entregan en cuanto hay unidades disponibles en stock, sin tener que esperar a que todo el pedido esté preparado. Por tanto la entrega de un pedido puede constar de varios envíos: uno por cada subpedido indicado, más un envío final con el resto de productos. De igual modo el cliente puede agrupar los productos de cada subpedido en subpedidos menores, de manera recursiva.

A continuación se muestra un ejemplo de pedido válido:

```
PEDIDO
producto "PR01": 3 unidades
producto "PR02": 4 unidades
subpedido 1:
   producto "PR03": 1 unidad
   producto "PR04": 1 unidad
subpedido 2:
   producto "PR05": 2 unidades
```

El pago de un pedido se puede realizar a través de una o varias cuentas de pago. Cada cuenta está asociada a una tarjeta de crédito, y tiene disponible cierta cantidad de dinero que el cliente debe aumentar periódicamente para poder realizar nuevos pedidos. Un pedido no podrá llevarse a cabo si la cuenta indicada para realizar el pago no tiene suficiente dinero. Cada subpedido está asociado a una única cuenta de pago. Como es de esperar, el sistema debe garantizar que todos los subpedidos de un pedido se pagan con cuentas del mismo cliente.

Se quiere que la responsabilidad del cobro, distribución y confirmación de los pedidos esté centralizada en una clase, de la cual debe haber una única instancia en la aplicación. El cobro de los pedidos se hace una vez al día, y el proceso consiste en comprobar todos los pedidos pendientes de cobro y cobrarlos de la cuenta de pago correspondiente. Si una cuenta no tiene suficiente dinero el pedido se rechaza; si es parte de un pedido que agrupa subpedidos se rechaza el pedido entero. Una vez que un pedido está listo para servirse se ordena su distribución y, una vez entregado, pasa a estar confirmado.

Se pide:

1. Realizar el diagrama de clases UML que constituye el diseño óptimo de esta aplicación.
2. Indicar qué patrones de diseño se han utilizado en el apartado 1), en caso de haber usado alguno. Señala qué clases participan en cada patrón y qué roles desempeñan. ¿Qué beneficios aporta el uso de tales patrones en la aplicación?
3. Realizar el diagrama de secuencia de la operación *obtener_Total* para el pedido que se muestra en el enunciado. Dicha operación calcula el precio total de un pedido a partir del precio de los productos que contiene.

Nota: documenta cualquier suposición sobre la aplicación que no esté en el enunciado.

Ejercicio 6 (examen de junio año 2007/08)

La Universidad Carlos III de Madrid gestiona todas las llamadas telefónicas que se realizan desde el campus a través de una centralita que se encarga de registrarlas antes de redirigirlas a la central de conmutación más adecuada dependiendo del tipo de llamada (provincial, inter-provincial, internacional o móvil). Al realizar una llamada, la centralita detecta desde qué teléfono se hizo y se comunica con el mismo utilizando la siguiente interfaz:

```
public interface Telefono {
   public String obtenerNumeroTfno ();
   public byte[] codificarVoz (byte[] b);
   public boolean enviarVoz ();
   // resto de métodos ...
}
```

Recientemente, la Universidad ha comprado un telefax para transmitir documentos a distancia mediante la red telefónica. El control del telefax se realiza a través de la siguiente clase:

```
public class Telefax {
   public long obtenerNumeroTfax () { ... }
   public byte[] codificarDocumento (byte[] b) { ... }
   public boolean enviarDocumento () { ... }
   // resto de métodos ...
}
```

Se pìde:

1. Utilizando patrones de diseño, especifica mediante sendos diagramas de clases dos posibles formas de poder usar el telefax a través de la centralita, teniendo en cuenta que no se dispone del código fuente de esta última. Comenta brevemente las soluciones propuestas, e indica las ventajas e inconvenientes de cada una de ellas.
2. Programa en java las soluciones propuestas en el apartado 1).
3. Se desea restringir el uso del telefax para que sólo pueda usarlo el personal de administración. Para ello, antes de proceder al envío de un documento, se debe introducir una clave. Implementa en java un *Proxy de Protección* que asegure que el envío de documentos se realiza sólo si la clave es correcta (supón que el proxy define un método `dameClave()` que devuelve la clave introducida). La clave se almacena en el proxy, y éste recuerda si se ha introducido correctamente o no en caso de querer enviar varios documentos.