

# Metodología i Programació Orientada a Objectes. Examen Parcial.

Cuatrimestre Otoño. Curso 2011-2012. Grupo 60

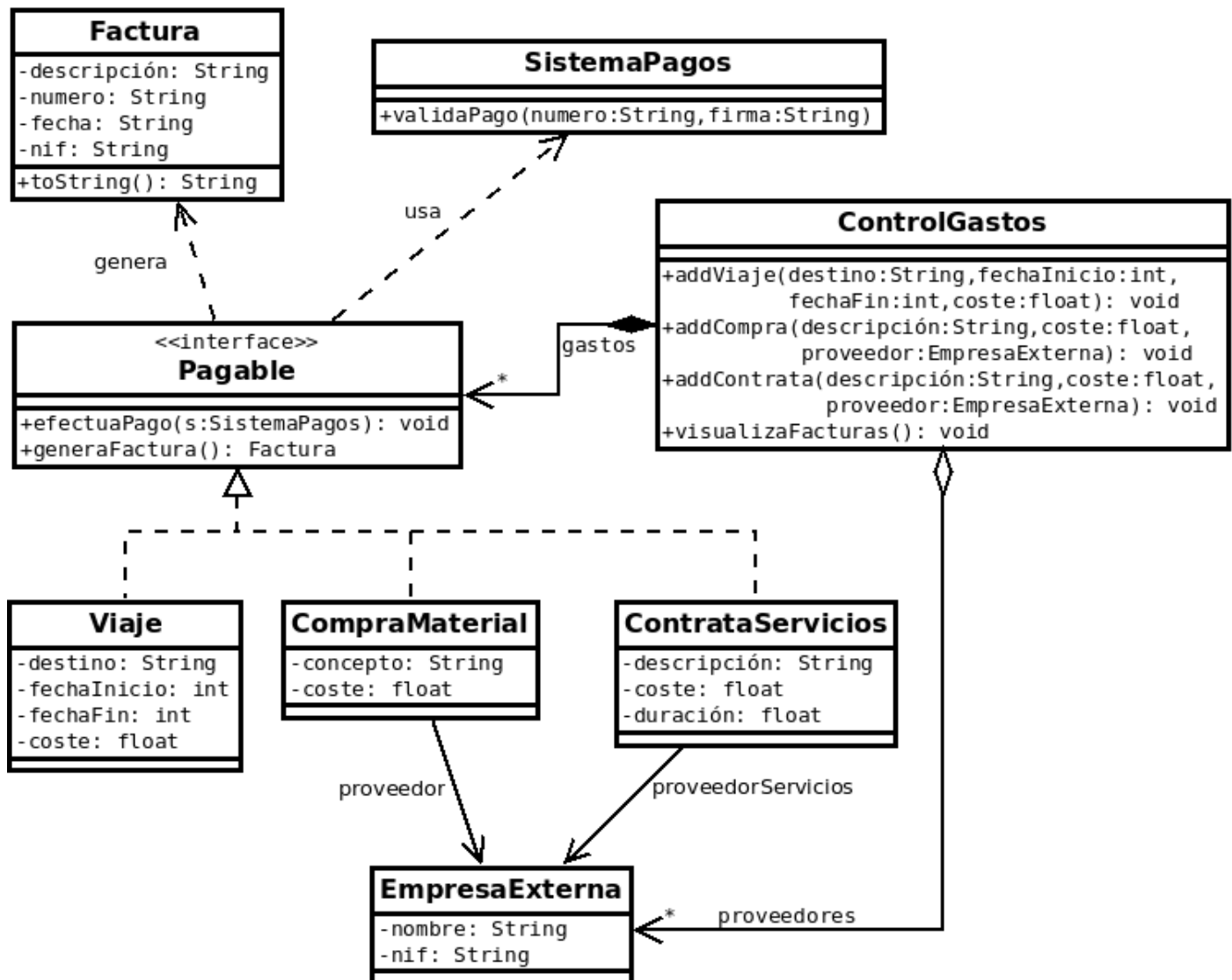
El departamento de compras de una empresa nos ha pedido crear una aplicación que controle los gastos corrientes de la empresa.

El punto central de la aplicación es una clase llamada `ControlGastos`, que es la encargada de registrar cada nuevo gasto que la empresa ha de hacer. Dichos gastos pueden ser de tres tipos:

- Viajes por motivos laborales. Por simplificar, solo se tiene en cuenta la ciudad de destino, la fecha de inicio y fin, y la suma total de los costes (vuelos, hotel, dietas, etc...)
- Compras de material diverso (ya sea una grapadora o un caro servidor informático). Además de la descripción del concepto y el coste de cada nueva adquisición de material, el sistema debe tener en cuenta la empresa proveedora.
- Contrata de servicios (servicios de limpieza, proveedor de telecomunicaciones, electricidad, etc...). Además de la descripción, el coste y la duración de cada servicio externalizado, el sistema debe tener en cuenta la empresa contratada.

Además del registro de cada nuevo gasto, el sistema ha de ser capaz de efectuar el pago en línea mediante algún sistema electrónico de pagos (por ejemplo: PayPal, VISA...) y generar una factura de dicho gasto.

Tras realizar un proceso de análisis y diseño de nuestra aplicación, hemos creado el siguiente UML:



## Preguntas y ejercicios

1. (0.5 puntos) Mirando el UML, ¿puedes ver el polimorfismo en alguna parte (sin tener en cuenta el método `toString()`)? Justifica la respuesta.
2. (0.5 puntos) ¿Crees que hubiera sido más adecuado que `Pagable` fuera una clase en vez de una interfaz? Justifica tu respuesta.
3. (0.5 puntos) Teniendo en cuenta lo que sabes sobre encapsulación, herencia y polimorfismo, ¿qué métodos crees que ha de definir e implementar la clase `Viaje`?
4. (1 punto) Entre la clase `ControlGastos` y la interfaz `Pagable` hay un tipo de relación similar a la existente entre las clases `ControlGastos` y `EmpresaExterna`. ¿Qué dos relaciones son, y en qué se diferencian? ¿Por qué motivo crees que se ha escogido cada una de ellas en este UML?
5. (1 puntos) Describe el tipo de relación que hay entre la interfaz `Pagable` y las clases `SistemaPagos` y `Factura`. ¿Por qué se ha usado concretamente esta relación entre `Pagable` y las clases `SistemaPagos` y `Factura`?
6. (3 puntos) Escribe la parte de definición del código de todas las clases del UML (empezando por el `public class ...`) excepto de la clase `SistemaPagos`. Dentro de cada clase, define solo los atributos y relaciones de éstas (no es necesario que definas ningún método ni constructor). Define también la interfaz `Pagable`, así como sus métodos.
7. (1.5 puntos) Asumiendo que en la clase `Factura` ya está redefinido el método `toString()`, y éste devuelve un `String` con todos los datos de una factura, escribe el código del método `visualizaFacturas` de la clase `ControlGastos`, que muestra por pantalla una lista con las facturas de todos los gastos que hay registrados en el sistema. Puedes asumir que el método `generaFactura()` ya está implementado.
8. (2 puntos) Implementa el código del siguiente método en la clase `ControlGastos`:  

```
public void addCompra(String concepto, float coste, EmpresaExterna proveedor)
```

Dicho método registra en `ControlGastos` una nueva compra de material, dado el concepto, el coste y una empresa proveedora externa. Además, si el proveedor que se pasa por parámetro no está registrado en la lista de proveedores de `ControlGastos`, el método debe añadirlo a ésta.

**NOTA 1:** asegúrate de que todas las relaciones entre las clases implicadas quedan completas.

**NOTA 2:** puedes asumir que el constructor `public CompraMaterial()` ya está creado, así como los *getters* y *setters* que encapsulan los diferentes atributos y relaciones de las clases.

## Solución

1. En las clases que implementan la interfaz `Pagable` (`Viaje`, `CompraMaterial` y `ContrataServicios`) se da polimorfismo, ya que éstas deben implementar los métodos que define `Pagable`. En nuestro programa podremos usar referencias a `Pagable` y, al ejecutar las llamadas a sus métodos, Java automáticamente seleccionará el código que pertoque, dependiendo del tipo de la clase.
2. Es más adecuado que `Pagable` sea una interfaz, ya que lo único que realmente tienen en común `Viaje`, `CompraMaterial` y `ContrataServicios` no es más que un comportamiento (efectuar pago y generar factura).
3. La clase `viaje` debería implementar los *getters* (y opcionalmente los *setters*) de sus atributos, así como los métodos `efectuaPago` y `generaFactura` de la interfaz `Pagable`.
4. Entre `ControlGastos` y `EmpresaExterna` existe una relación de agregación, mientras que entre `ControlGastos` y `Pagable` existe una relación de composición (o agregación fuerte). Ambas relaciones indican una relación entre un contenedor y un objeto o grupo de objetos contenidos. La diferencia es que en una composición, cuando el contenedor es destruido los elementos contenidos en éste también son destruidos. En la agregación no sucede esto.

En el UML del examen se ha escogido una relación de composición entre `ControlGastos` y `Pagable` porque si el objeto `ControlGastos` fuera destruido, los objetos pagables contenidos en él también lo serían. La relación entre `ControlGastos` y `EmpresaExterna` es de agregación porque los objetos `EmpresaExterna` seguirían teniendo vigencia contenidos en otras clases.

5. Entre la interfaz `Pagable` y las clases `Factura` y `SistemaPagos` hay una relación de dependencia. Esto quiere decir que `Pagable` necesita en algún momento de la existencia de estas clases, pero que las clases en sí no son atributos de `Pagable`.

En el UML del examen, la relación entre `Pagable` y `Factura` es de dependencia porque el método `generaFactura` debe instanciar y devolver un objeto del tipo `Factura`. La relación entre `Pagable` y `SistemaPagos` es de dependencia porque el método `efectuaPago` de `Pagable` requiere un objeto del tipo `SistemaPagos` como parámetro.

6. 

```
public class Factura {
    private String descripción, numero, fecha, nif;
}
public interface Pagable {
    public void efectuaPago(SistemaPagos s);
    public Factura generaFactura();
}
public class Viaje implements Pagable {
    private String destino;
    private int fechaInicio, fechaFin;
    private float coste;
}
public class CompraMaterial implements Pagable {
    private String concepto;
    private float coste;
    private EmpresaExterna proveedor;
}
public class ContrataServicios implements Pagable {
    private String descripción;
    private float coste, duración;
    private EmpresaExterna proveedorServicios;
}
public class EmpresaExterna {
    private String nombre, nif;
}
```

```

public class ControlGastos {
    private Set<Pagable> gastos;
    private Set<EmpresaExterna> proveedores;
}

7. public void visualizaFacturas() {
    System.out.println("Este es el registro de facturas en el sistema:");
    Iterator<Pagable> it = gastos.iterator();
    while(it.hasNext()) {
        Pagable p = it.next();
        Factura f = it.generaFactura();
        System.out.println(f.toString());
    }
}

8. public void addCompra(String concepto, float coste, EmpresaExterna proveedor)
{
    if(!proveedores.contains(proveedor)) {
        proveedores.add(proveedor);
    }
    CompraMaterial compra = new CompraMaterial();
    compra.setConcepto(concepto);
    compra.setCoste(coste);
    compra.setProveedor(proveedor);
    gastos.add(compra);
}

```