Curso 2019-2020. Cuatrimestre Otoño. 8 de Enero de 2020

Programación Orientada a Objetos. Examen Final

SOLUCIÓN

Una empresa que fabrica productos para ópticas tiene en plantilla a un cierto número de viajantes que visitan regularmente a sus clientes (ópticas) para ofrecer dichos productos y formalizar pedidos de los mismos con los titulares de dichas ópticas.

Un pedido está constituido por una secuencia de líneas. Cada línea consta de los detalles de un producto y del número de unidades de dicho producto que el cliente (la óptica en cuestión) desea comprar.

La empresa dispone de un programa de privilegios, que otorga a sus clientes en función de diferentes criterios.

Los privilegios pueden ser de dos tipos: descuentos en todos los productos comprados y regalos de productos que se añaden a un pedido a coste 0.

Los privilegios de tipo descuento incluyen el porcentaje de descuento a realizar en los productos de un pedido, y el precio del pedido calculado después de aplicar a todos los productos del mismo dicho descuento.

Los privilegios de tipo regalo incluyen un producto regalado que al aplicarse añadirá dicho producto a un pedido a coste 0.

Un cliente solo puede disponer de un privilegio de tipo descuento, pero puede disponer de varios privilegios de tipo regalo.

Los privilegios de tipo regalo desaparecen al aplicarse a un pedido, es decir, el cliente deja de disfrutar de ellos después de aplicarlos.

NOTA 1: Al resolver las preguntas del examen podéis suponer implementadas los métodos **getXXX() y setXXX()** que acceden a los atributos de las clases. **No es necesario que los implementéis**.

NOTA 2: Si al implementar alguno de los métodos que se solicitan en el examen necesitáis de algún **método diferente a los getXXX() y setXXX(), DEBERÉIS IMPLEMENTARLO**.

NOTA 3: A lo largo del examen, en la implementación de los métodos que se piden, y siempre que sea posible, debéis utilizar los métodos descritos en los apartados anteriores **INDEPENDIENTEMENTE DE QUE HAYÁIS O NO GENERADO SU CÓDIGO**.

NOTA 4: Suponed que se os da implementada la clase Fecha (esto es, **NO TENÉIS QUE IMPLEMENTAR NADA DE ELLA**), con los siguientes métodos ESTÁTICOS (que deberéis usar en aquellos métodos en los que hayáis de gestionar fechas y años):

```
public static String getFechaActual()
```

Este método crea y devuelve un String que representa la fecha en la que dicho método es invocado. Asumid que el formato del String es DDMMAAAA, **aunque para el código que tenéis que desarrollar es IRRELEVANTE y podría ser otro**. NO os tenéis que preocupar de ese detalle.

```
public static int compareTo(String unaFecha, String otraFecha)
```

Este método compara las fechas representadas por los strings pasados como argumentos. Devuelve un entero positivo si la fecha representada por el argumento `unaFecha` es **posterior** a la fecha representada por el argumento `otraFecha`. Devuelve un entero negativo si si la fecha representada por el argumento `unaFecha` es **anterior** a la fecha representada por el argumento `otraFecha` . Devuelve 0 si ambos argumentos representan la misma fecha.

`public static int getAnyo(String fecha)`

Este método devuelve como un entero el año de la fecha representada por el argumento `fecha`.

Ejercicio 1 **(1 punto). A partir del diagrama (incompleto) de clases UML mostrado en la última hoja de este enunciado**, escribid para todas las clases e interfaces mostradas en el diagrama UML, EXCEPTO LA CLASE Fecha, la parte de código de la definición que comienza con el "public class…." correspondiente a la declaración de los todos los atributos de las clases, incluyendo aquellos que aparecen al implementar las relaciones. NO implementéis todavía ningún método.

```
public class Controlador {
    private Catalogo catalogo;
    private Map<String, Cliente> clientes; // Clave: nif
    private Map<String, Viajante> viajantes; // Clave: nif
    private List<Pedido> pedidos;
    …
}
public class Catalogo {
    private Map<String, Producto> productos; // Clave: id
   …
}
public class Viajante {
    private String nif;
    private String nombre;
    private List<Pedido> pedidos;
    …
}
public class Pedido {
    private String fecha;
    private List<Linea> lineas;
    private Cliente cliente;
    private Viajante viajante;
    …
}
```

```
public class Cliente {
    private String nif;
    private String nombre;
    private String telefono;
    private String clienteDesde;
    private double facturado;
    private List<Pedido> pedidos;
    private List<Privilegio> privilegios;
    private int ultimoPrivilegioRegaloEn;
    …
}
public class Producto {
    private String id;
    private double precio;
    private String nombre;
    …
}
public class Linea {
    private int cantidad;
    private Producto producto;
    …
}
public abstract class Privilegio {
    protected Cliente cliente;
    …
}
public class Descuento extends Privilegio{
    private double porcentaje;
    private double costeDespuesAplicarDescuento;
    …
}
public class Regalo extends Privilegio{
    private Producto productoARegalar;
    …
}
```

Ejercicio 2 **(0,5 puntos)**. Implementad el constructor de la clase Cliente. Dicho constructor debe hacer que el atributo `clienteDesde` tome como valor un string que represente la fecha en la que el constructor es invocado, que no tenga ningún pedido realizado ni ningún privilegio. Decidid los argumentos necesarios para dicho constructor.

```
    public Cliente(String nif, String nombre, String telefono) {
        this.nif = nif;
        this.nombre = nombre;
        this.telefono = telefono;
        this.clienteDesde = Fecha.getFechaActual();
        this.pedidos = new ArrayList<>();
        this.privilegios = new ArrayList<>();
    }
```

Ejercicio 3 **(1,5 punto)**. Implementad en la clase Pedido los siguientes métodos:

3.1. (0,5 puntos). Constructor. Debe inicializarlo a un pedido sin productos. Debe hacer que el atributo fecha sea un string que represente la fecha en la que el constructor es invocado. Decidid los argumentos a pasar a dicho constructor.

```
    public Pedido(Viajante viajante, Cliente cliente) {
        this.viajante = viajante ;
        this.cliente = cliente ;
        this.viajante.addPedido(this);
        this.cliente.addPedido(this);
        this.fecha = Fecha.getFechaActual();
        this.lineas = new ArrayList<Linea>();
    }
```

3.2. (0,1 puntos). `public void addLinea(Producto p, int cantidad)`

Este método debe crear una nueva línea a partir del producto referenciado por el argumento `p` y la cantidad pasada en el argumento `cantidad`.

```
    public void addLinea(Producto p, int cantidad){
        this.lineas.add(new Linea(cantidad,p)) ;
    }
En la clase Linea se define el siguiente constructor:
    public Linea(int cantidad, Producto producto) {
        this.cantidad = cantidad;
        this.producto = producto;
    }
```

3.3. (0,4 puntos). `public double calcularCoste()`

Este método debe devolver el coste total del pedido sin tener en cuenta privilegio alguno.

```
public double calcularCoste(){
    double coste = 0 ;
    for(Linea lin: this.lineas){
       Producto p = lin.getProducto() ;
       coste+=lin.getCantidad()*lin.getProducto().getPrecio();
    }
    return coste ;
}
```

3.4. (0,5 puntos). `public Producto getProductoConMasUnidades() ;`

Este método devuelve el producto con más unidades solicitadas en el pedido. Si hay varios productos de los que se han solicitado este número de unidades máximo, el método devuelve uno de ellos solamente; da igual cuál de ellos.

```
public Producto getProductoConMasUnidades() {
    int numUnidadesMayor = 0;
    Producto pConMasUnidades = null;
    Iterator<Linea> it = this.lineas.iterator();

    while (it.hasNext()) {
        Linea lin = it.next();
        if (lin.getCantidad() > numUnidadesMayor) {
            pConMasUnidades = lin.getProducto();
            numUnidadesMayor = lin.getCantidad();
        }
    }
    return pConMasUnidades;
}
```

Ejercicio 4 **(2,75 puntos)**. Implementad en la clase Cliente los siguientes métodos

4.1. (2 puntos). `public List<Pedido> pedidosEnAnyo(int anyo)`

Este método devuelve una lista con los pedidos realizados por el cliente en el año indicado por el argumento `anyo`.

Debéis suponer que los pedidos realizados por un cliente están ordenados por fecha. Para obtener el resultado se os ofrecen dos alternativas:

- Usando la búsqueda dicotómica. Para ello:
  1. Deberá realizar una búsqueda dicotómica de un pedido realizado en el año indicado por el argumento `anyo`.
  2. Deberá añadir en la lista resultado aquellos pedidos que precedan al pedido encontrado en el punto anterior que también se hayan realizado en el mismo año.
  3. Deberá añadir en la lista resultado aquellos pedidos que sigan al pedido encontrado en el

punto 1 que también se hayan realizado en el mismo año.

- Usando una búsqueda secuencial. Obviamente, la valoración de la pregunta será distinta (si utilizáis esta opción tendréis una **penalización de 1 punto** en la nota de esta pregunta).

Recordatorio: la búsqueda binaria o dicotómica es un algoritmo de búsqueda eficiente que permite encontrar la posición que ocupa un elemento (elem) concreto dentro de un vector (v) ordenado. Inicialmente, el algoritmo obtiene los índices del primer (inf) y último (sup) elemento del vector. Además, obtiene el índice de su elemento central (centro) mediante la expresión: centro = (inf + sup)/2. Si v[centro] = elem, el elemento se ha encontrado y la búsqueda finaliza. Si elem < v[centro] o elem > v[centro], el algoritmo debe actualizar los valores de inf o sup y recalcular el nuevo valor de centro, procediendo a comprobar otra vez si v[centro] = elem. El algoritmo finaliza una vez encontrado el elemento, o en caso que la condición inf <= sup deje de cumplirse, lo que indicará que el elemento no ha sido encontrado en el vector.

```
    public List<Pedido> pedidosEnAnyo(int anyo) {
        List<Pedido> result = new ArrayList<Pedido>();
        // Búsqueda dicotómica
        int left = 0;
        int rigth = this.pedidos.size() - 1;
        int center = this.pedidos.size() / 2;
        boolean found = false;
        int whereFound = -1;
        int anyoPed;
        while (!found && left <= rigth) {
            Pedido p2Test = this.pedidos.get(center);
            anyoPed = Fecha.getAnyo(p2Test.getFecha());
            if (anyoPed == anyo) {
                whereFound = center;
                result.add(p2Test);
                found = true;
                break;
            }
            if (anyoPed > anyo) {
                rigth = center - 1;
            } else {
                left = center + 1;
            }
            center = left + (rigth - left) / 2;
        }
        if (!found) {
            return result;
        }
```

```
        // Buscar a la izquierda
        Pedido p2Test ;
        boolean foundOtherYear = false ;
        int i ;
        if (whereFound != 0 ) {
            i = whereFound-1 ;
            while(i>=0 && !foundOtherYear){
                p2Test = this.pedidos.get(i);
                anyoPed = Fecha.getAnyo(p2Test.getFecha());
                if(anyoPed==anyo){
                    result.add(0,p2Test) ;
                    i-- ;
                }else{
                    foundOtherYear = true ;
                }
            }
        }
        // Busqueda a la derecha
        int end = this.pedidos.size() ;
        if (whereFound == end-1) {
            return result;
        }
        i = whereFound+1 ;
        foundOtherYear = false ;
        while(i<end && !foundOtherYear){
                p2Test = this.pedidos.get(i);
                anyoPed = Fecha.getAnyo(p2Test.getFecha());
                if(anyoPed==anyo){
                    result.add(p2Test) ;
                    i++ ;
                }else{
                    foundOtherYear = true ;
                }
        }
        return result;
    }
```

4.2. (0,75 puntos). `public double facturadoEnAnyo(int anyo)`

Este método devuelve la cantidad facturada por los pedidos realizados por el cliente en el año indicado por el argumento `anyo`, sin tener en cuenta privilegio alguno.

```
public double facturadoEnAnyo(int anyo) {
    List<Pedido> pedidos = this.pedidosEnAnyo(anyo);
    double result = 0;
    for (Pedido pedido : pedidos) {
        result += pedido.calcularCoste();
    }
    return result;
}
```

Ejercicio 5 **(2 puntos)**. Implementad en la clase Controlador los siguientes métodos

5.1. (1 punto). `public void asignarPrivilegioDescuento(String nif) throws NoClienteException`

Este método trata de crear y asignar un privilegio de tipo descuento al cliente cuyo NIF coincide con el valor del argumento `nif`.

Si no existe ningún cliente con un NIF que coíncida con el valor del argumento `nif`, lanza la excepción NoClienteException.

Se creará y asignará al cliente un privilegio de tipo descuento si se cumplen las siguientes condiciones:

1. El cliente no tiene ningún privilegio de tipo descuento. IMPORTANTE: suponed que en la clase Cliente se da implementado (NO TENÉIS QUE IMPLEMENTARLO) el método

   `public boolean hayPrivilegioDescuento()`

   que devuelve `true` si el cliente ya disfruta de un privilegio de tipo descuento, y `false` en caso contrario.
2. El número de años transcurridos desde su primera compra es igual o superior a 5 años.
3. En los últimos 2 años el cliente ha realizado un mínimo de dos pedidos.
4. En cada uno de los 2 últimos años la suma de los resultados de invocar al método `calcularCoste()` de cada uno de los pedidos de cada año es igual o superior a 500 Euros.

El descuento a aplicar será del 2% en todos los productos de los pedidos.

```
public void asignarPrivilegioDescuento(String nif)
            throws NoClienteException{
    Cliente cliente = this.clientes.get(nif) ;
    if(cliente==null){
        throw new NoClienteException(
                "No existe ningún cliente con el NIF " + nif) ;
    }
```

```
        // No hay privilegios descuento
        if(cliente.hayPrivilegioDescuento()){
            return ;
        }
        // Años desde primera compra >=5
        int anyoInicio = Fecha.getAnyo(cliente.getClienteDesde()) ;
        int anyoActual = Fecha.getAnyo(Fecha.getFechaActual()) ;
        if(anyoActual - anyoInicio<5){
            return ;
        }
        // 2 o más pedidos en los dos últimos años
        List<Pedido> anyoMenos1 = cliente.pedidosEnAnyo(anyoActual-1) ;
        List<Pedido> anyoMenos2 = cliente.pedidosEnAnyo(anyoActual-2) ;
        if(anyoMenos1.size()<2 || anyoMenos2.size()<2){
            return ;
        }
        if(cliente.facturadoEnAnyo(anyoActual-1)<500 ||
                cliente.facturadoEnAnyo(anyoActual-2)<500){
            return ;
        }
        Descuento descuento = new Descuento(cliente,2) ;
        cliente.addPrivilegio(descuento);
    }
En la clase Descuento hay que definir su constructor como sigue:
    public Descuento(Cliente cliente,double porcentaje) {
        super(cliente) ;
        this.porcentaje = porcentaje;
    }
Puesto que Privilegio tiene una asociación bidirecciona con Cliente cabe
pensar que el constructor de Privilegio será
public Privilegio(Cliente cliente) ;
```

5.2. (1 punto). `public void asignarPrivilegiosRegalo(String nif) throws NoClienteException`

Este método trata de crear y asignar privilegios de tipo regalo al cliente cuyo NIF coincide con el valor del argumento `nif`.

Si no existe ningún cliente con un NIF que coincida con el valor del argumento `nif`, lanza la excepción NoClienteException.

El método llevará a cabo la creación y asignación de privilegios de tipo regalo como se indica a continuación:

1. Solo se crearán privilegios de este tipo si el año de la fecha en la que se ejecuta este método es posterior al año de la fecha indicada en el atributo `ultimoPrivilegioRegaloEn` del cliente.
2. Se creará un privilegio de tipo regalo por cada pedido realizado el año de la fecha en que se ejecuta este método si el resultado de invocar al método `calcularCoste()` del pedido, detallado en la pregunta 3, es igual o superior a 750 Euros.
3. Se asigna al atributo `ultimoPrivilegioRegaloEn` el valor del año de la fecha en la que se ejecuta el método.

Cada privilegio creado deberá incluir como producto a regalar (esto es, deberá tener un coste de 0) el producto del que más unidades se han solicitado en el pedido en cuestión.

```
    public void asignarPrivilegiosRegalo(String nif)
                throws NoClienteException{
        Cliente cliente = this.clientes.get(nif) ;
        if(cliente==null){
            throw new NoClienteException(
                   "No existe ningún cliente con el NIF " + nif) ;
        }
        int anyoActual = Fecha.getAnyo(Fecha.getFechaActual()) ;
        if(anyoActual <= cliente.getUltimoPrivilegioRegaloEn()){
            return ;
        }
        List<Pedido> pedidos = cliente.pedidosEnAnyo(anyoActual) ;
        for(Pedido pedido: pedidos){
            if(pedido.calcularCoste()>= 750){
                Producto aClonar=
                            pedido.getProductoConMasUnidades();
                Producto p = new
                  Producto(aClonar.getId(),aClonar.getNombre(),0);
                Regalo regalo = new Regalo(cliente,p) ;
                cliente.addPrivilegio(regalo);
            }
        }
        cliente.setUltimoPrivilegioRegaloEn(anyoActual);
    }

// El constructor de Regalo:
    public Regalo(Cliente cliente,Producto productoARegalar) {
        super(cliente) ;
        this.productoARegalar = productoARegalar;
```

```
    }

// El constructor de Producto:
    public Producto(String id, String nombre, double precio) {
        this.id = id;
        this.precio = precio;
        this.nombre = nombre ;
    }
```

Ejercicio 6 **(1,25 puntos)**. Implementad en la clase Descuento el siguiente método

6.1. (1 punto). `public void aplica(Pedido p)`

Este método pone a 0 el valor del atributo `costeDespuesAplicarDescuento` y a continuación calcula el coste del pedido pasado en el argumento `p` después de aplicar el descuento indicado en el atributo porcentaje a todos los productos del pedido, y guarda el coste total calculado en el atributo `costeDespuesAplicarDescuento`.

```
    @Override
    public void aplica(Pedido pedido) {
        this.costeDespuesAplicarDescuento = 0  ;
        List<Linea> lineas = pedido.getLineas() ;
        for(Linea lin: lineas){
            this.costeDespuesAplicarDescuento +=
                    lin.getProducto().getPrecio()*
                    (1-(this.porcentaje/100))*lin.getCantidad() ;
        }
    }
/* Nota: también se acepta como válida la siguiente solución, que
invoqua al método calcularCoste() del objeto de clase Pedido
pasado como argumento y aplica el descuento al valor devuelto por
dicho método.
*/
    @Override
    public void aplica(Pedido pedido) {
        this. costeDespuesAplicarDescuento =
pedido.calcularCoste() * (1 - (this.porcentaje / 100));
    }
```

Implementad en la clase Regalo el siguiente método:

6.2. (0,25 puntos). `public void aplica(Pedido p)`

Este método añade al pedido una unidad del producto incluido en el privilegio regalo como

producto a regalar (a coste 0).

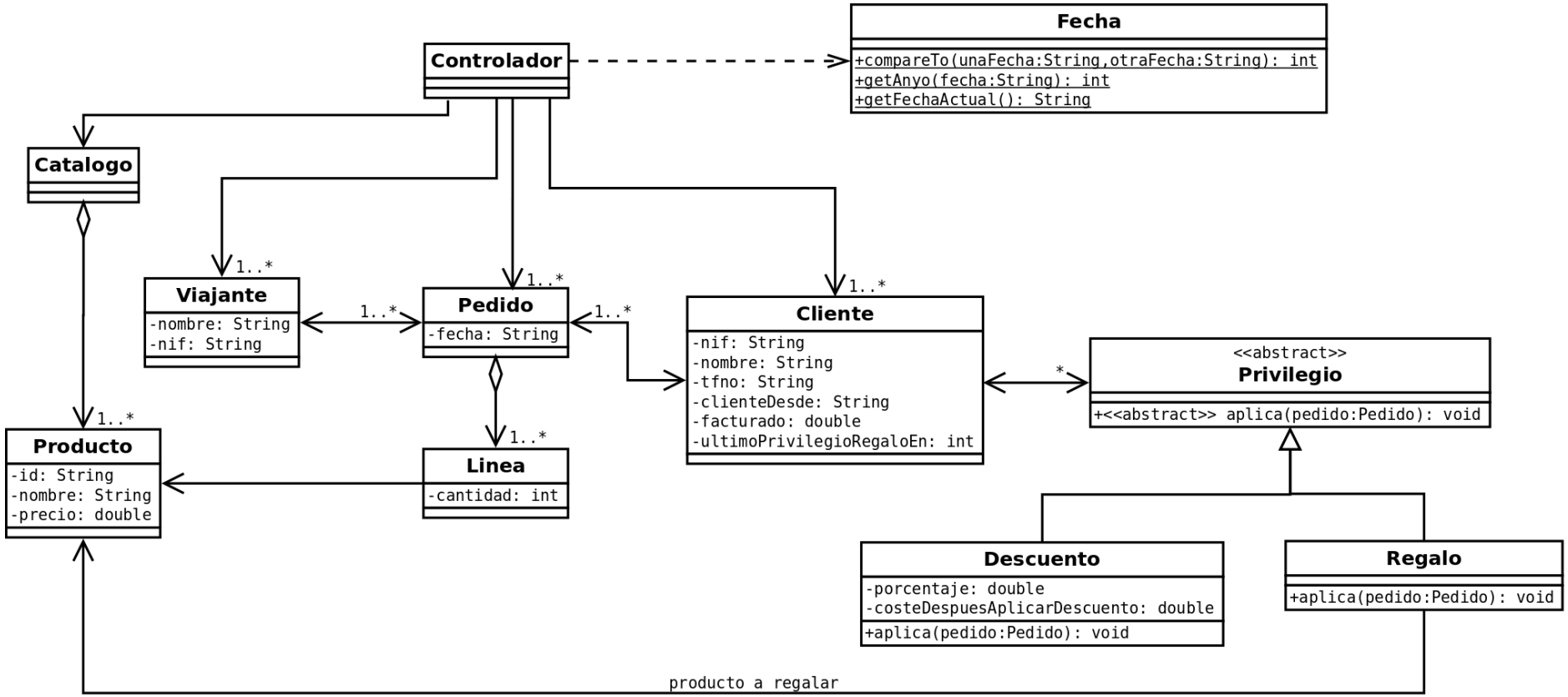
```
    @Override
    public void aplica(Pedido pedido) {
        pedido.addLinea(this.productoARegalar,1);
    }
```

Ejercicio 7 **(1 punto)**. Implementad en la clase Controlador los siguientes métodos

`public void procesaPedido(Pedido p)`

Este método debe procesar el pedido pasado a través del argumento `p`. El procesado consiste en la aplicación de todos los privilegios de los que disfruta el cliente. Finalmente, debe borrar de la lista de privilegios de los que disfruta el cliente aquellos que son privilegios de tipo regalo.

```
    public void procesaPedido(Pedido p){
        Cliente cliente = p.getCliente() ;
        List<Privilegio> privilegios = cliente.getPrivilegios() ;
        for(Privilegio priv: privilegios){
            priv.aplica(p);
        }
        Iterator<Privilegio> it = privilegios.iterator() ;
        while(it.hasNext()){
            Privilegio priv = it.next() ;
            if(priv instanceof Regalo){
                it.remove();
            }
        }
    }
```


Fecha
+compareTo(unaFecha:String,otraFecha:String): int
+getAnyo(fecha:String): int
+getFechaActual(): String
Controlador
Catalogo
Viajante
-nombre: String
-nif: String
Pedido
-fecha: String
Cliente
-nif: String
-nombre: String
-tfno: String
-clienteDesde: String
-facturado: double
-ultimoPrivilegioRegaloEn: int
<<abstract>>
Privilegio
+<<abstract>> aplica(pedido:Pedido): void
Producto
-id: String
-nombre: String
-precio: double
Linea
-cantidad: int
Descuento
-porcentaje: double
-costeDespuesAplicarDescuento: double
+aplica(pedido:Pedido): void
Regalo
+aplica(pedido:Pedido): void
1..*
*
producto a regalar