Curso 2019-2020. Programación Orientada a Objetos.

Examen final de reevaluación. 7 de julio de 2020.

Publicación de notas provisionales: 13/07/2020 (Atenea)

Alegaciones: Aplicación de alegaciones en la web de la ETSETB, hasta el 14/07/2020 a las 19:00h

Publicación de notas definitivas: 16/07/2020 (Atenea)

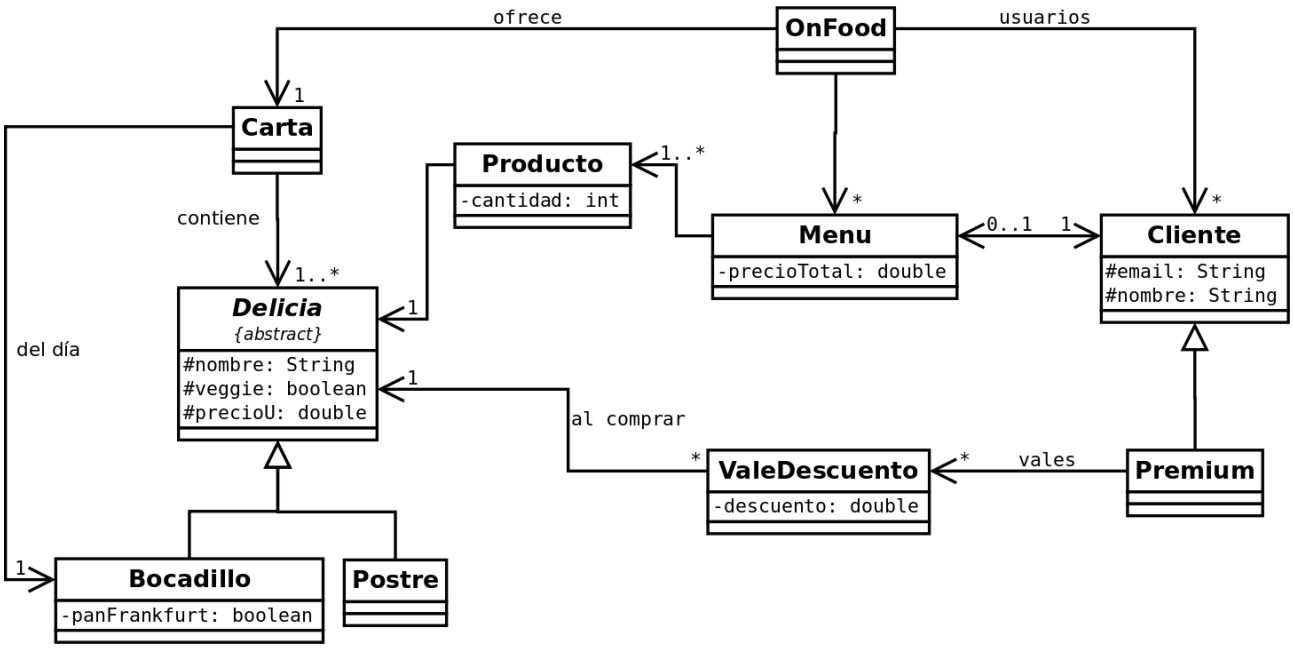
Un conocido restaurante nos ha pedido el desarrollo de una aplicación online llamada *OnFood*, que permita a sus clientes encargar comida a domicilio. La aplicación *OnFood* deberá gestionar la carta de delicias gastronómicas que ofrece el restaurante, especializado en bocadillos y postres caseros. De cada delicia se desea guardar su nombre (que será único de entre todas las delicias del restaurante), si es adecuada para dieta vegetariana y su precio/unidad. Además, en el caso de los bocadillos, se desea guardar también si estos se preparan con pan tierno tipo Frankfurt o no (sino, se preparan con pan de barra convencional). A diario, el restaurante elige un *bocadillo del día*, con un apetecible descuento de un 25% sobre su precio de venta habitual.

El restaurante desea diferenciar además entre clientes regulares y *Premium*, a los que ofrecerá descuentos en determinadas delicias gastronómicas del restaurante. De todo cliente, se desea guardar su dirección de correo electrónico (que servirá como identificador único) y su nombre. De los clientes *Premium* se deberá guardar además los vales descuento de los que disponen. Cada vale descuento especificará un descuento (en %) aplicable a todas las unidades de una cierta delicia gastronómica, cuando el cliente *Premium* la añada al menú encargado. Podéis asumir que la aplicación asegurará que un cliente *Premium* nunca dispondrá de más de un vale descuento para una misma delicia.

Los menús encargados se gestionarán como objetos de tipo Menu, que guardarán los productos encargados y su precio total. Cada producto representará una delicia concreta y el número de unidades encargadas de ella. Estos productos deberán ser guardados en un contenedor de tipo mapa, tal que queden identificados por la delicia encargada como clave.

Además, un cliente podrá encargar un único menú a la vez. Podéis asumir que la aplicación eliminará posteriormente aquellos menús servidos con anterioridad.

A continuación, se muestra el diagrama de clases UML de la aplicación *OnFood*:



NOTA 1: Al resolver las preguntas del examen podéis suponer implementados los métodos **getXXX() y setXXX()** que acceden a los atributos de las clases, aunque no aparezcan en el UML. **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**.

Ejercicio 1 **(1,5 puntos).** A partir del diagrama (incompleto) de clases UML mostrado en la última hoja de este enunciado, escribid para todas las clases mostradas en el diagrama UML, 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 OnFood {
    private Carta carta;
    private Map<String, Menu> menus;            //Key: email cliente
                                                //Se acepta también List<Menu>
    private Map<String, Cliente> clientes;      //Key: email cliente
}

public class Carta {
    private Map<String,Delicia> delicias;   //Key: nombre
    private Bocadillo delDia;
}

public abstract class Delicia {
    protected String nombre;
    protected boolean veggie;
    protected double precioU;
}

public class Bocadillo extends Delicia {
    private boolean panFrankfurt;
}

public class Postre extends Delicia {
}

public class Producto {
    private Delicia delicia;
    private int cantidad;
}

public class Menu {
    private Cliente cliente;
    private Map<Delicia,Producto> productos;    //Key: delicia encargada
    private double precioTotal;
}

public class Cliente {
```

```
    protected String email, nombre ;
    protected Menu menu;
}

public class Premium extends Cliente {
    private List<ValeDescuento> valesDescuento;
}

public class ValeDescuento {
    private Delicia delicia;
    private double descuento;
}
```

Ejercicio 2 **(2 puntos)**. Implementad el método `toString()` en las clases `Carta`, `Delicia`, `Bocadillo` y `Postre`.

A continuación se muestra un ejemplo de salida esperada de la aplicación cuando se invoca el método `toString()` de la clase `Carta` para mostrar la carta de delicias ofrecida por el restaurante. En este ejemplo, se ofrecen 4 bocadillos (Frankfurt, Hamburguesa, Lomo con queso y Veggie) y 2 postres (Flan vegano y Tarta de queso). El bocadillo del día con un 25% de descuento sobre su precio habitual es Frankfurt:

```
Carta de delicias:
==================

POSTRE:       Flan vegano; VEG; precio/unidad: 2.5 euros.
BOCADILLO:    Frankfurt; precio/unidad: 3.5 euros. Pan de frankfurt
POSTRE:       Tarta de queso; precio/unidad: 2.7 euros.
BOCADILLO:    Lomo con queso; precio/unidad: 4.0 euros. Pan de frankfurt
BOCADILLO:    Hamburguesa; precio/unidad: 3.3 euros.
BOCADILLO:    Veggie; VEG; precio/unidad: 3.0 euros.

Bocadillo del día: Frankfurt
```

```
public class Carta {

    @Override
    public String toString()
    {
        String s = "Carta de delicias:\n";
        s += "============================\n\n";
        for (Delicia del : this.delicias.values())
        {
            s += del.toString() + "\n";
        }

        if (this.delDia !=null)
            s += "\nBocadillo del día: " + this.delDia.getNombre();

        return s;
    }
}
```

```
public abstract class Delicia {

    @Override
    public String toString()
    {
        String s = this.nombre + ";";
        if (this.veggie)
            s += " VEG;";

        s += " precio/unidad: " + this.precioU + " euros.";
        return s;
    }
}

public class Bocadillo extends Delicia {

    @Override
    public String toString()
    {
        String s = "BOCADILLO: \t" + super.toString();

        if (this.panFrankfurt)
            s += " Pan de frankfurt ";

        return s;
    }
}

public class Postre extends Delicia {

    @Override
    public String toString()
    {
         return "POSTRE: \t" + super.toString();
    }
}
```

Ejercicio 3 **(1,5 puntos)**. Implementad en la clase `OnFood` el siguiente método:

```
public void crearMenu (String email) throws ClienteException,
MenuException;
```

Este método intenta iniciar el encargo de un nuevo menú para el cliente con la dirección de correo `email`, pasada como parámetro. Si no existe ningún cliente con esa dirección de correo, el método lanza una excepción de tipo `ClienteException`. Si el cliente existe, pero ya estaba encargando otro menú en ese mismo instante, el método lanza una excepción de tipo `MenuException`. En caso contrario, el método deberá crear un nuevo menú sin producto alguno para ese usuario y guardarlo.

La resolución de este ejercicio exige que implementéis también el método constructor de la clase `Menu`.

```
public class Menu
{
    public Menu(Cliente cliente)
    {
```

```
        this.cliente = cliente;
        this.productos = new HashMap<>();
        this.precioTotal = 0.0;
    }
}

public class OnFood
{
    public void crearMenu (String email) throws ClienteException, MenuException
    {
        Cliente c = this.clientes.get(email);
        if (c == null)
            throw new ClienteException();

        if (c.getMenu() != null)
            throw new MenuException();

        Menu menu = new Menu (c);
        this.menus.put(email, menu);    //this.menus.add(menu) si anteriormente
                                        //definido como List<Meu>
        c.setMenu(menu);
    }
}
```

Ejercicio 4 **(1,5 puntos)**. Implementad en la clase `Menu` el siguiente método:

`public void addDelicia(Delicia del, int unidades);`

Este método debe añadir al menú que recibe su invocación un número de unidades igual al parámetro `unidades` de la delicia `del`, también pasada como parámetro. Si dicha delicia no había sido aún encargada en ese menú, el método deberá crear un nuevo producto para esa delicia y con ese número de unidades. En caso contrario, simplemente deberá actualizar el número de unidades encargadas de la delicia. Finalmente, el método deberá actualizar el precio total del menú encargado.

La resolución de este ejercicio exige que implementéis también el método constructor de la clase `Producto`.

```
public class Producto
{
    public Producto (Delicia delicia, int cantidad)
    {
        this.delicia = delicia;
        this.cantidad = cantidad;
    }
}

public class Menu
{
    public void addDelicia(Delicia del, int unidades)
    {
        Producto p = this.productos.get(del);
        if (p == null)
            this.productos.put(del, new Producto(del,unidades));
        else
            p.setCantidad(p.getCantidad() + unidades);

        this.precioTotal = this.precioTotal + unidades * del.getPrecioU();
    }
}
```

Ejercicio 5 **(1,5 puntos)**. Implementad en la clase `Menu` el siguiente método:

```
public void aplicarValesDescuentoCliente (List<ValeDescuento> valesDesc);
```

Este método intenta aplicar todos los vales descuento contenidos en la lista pasada como parámetro al menú que recibe la invocación del método. Si existe en la lista un vale descuento para una determinada delicia, su descuento deberá ser aplicado a todas las unidades de esa delicia encargadas en el menú, actualizando así el precio total del menú. El método debe eliminar todos aquellos vales descuento aplicados satisfactoriamente de la lista `valesDesc` pasada como parámetro. Se consideran vales descuento no aplicados si el menú no había encargado la delicia para la cual pueden utilizarse.

```
public class Menu
{
    public void aplicarValesDescuentoCliente (List<ValeDescuento> valesDesc)
    {
        List<ValeDescuento> aplicados = new LinkedList<>();

        //Intentamos aplicar los vales descuento
        for (ValeDescuento vale : valesDesc)
        {
            if (this.productos.containsKey(vale.getDelicia()))
            {
                int unidades =
                        this.productos.get(vale.getDelicia()).getCantidad();

                this.precioTotal = this.precioTotal -
                        unidades*vale.getDelicia().getPrecioU()*
                        vale.getDescuento()/100;

                aplicados.add(vale);
            }
        }

        //Eliminamos vales descuento aplicados
        valesDesc.removeAll(aplicados);

        /* Alternativamente al uso de removeAll(), podrían eliminarse los vales
           descuento aplicados, uno a uno. Código también aceptado:

           for (ValeDescuento vale : aplicados)
               valesDesc.remove(vale);
        */
    }
}

/* Implementación alternativa del método utilizando un objeto de tipo Iterator y
su método remove(), también aceptada:
    public void aplicarValesDescuentoCliente (List<ValeDescuento> valesDesc)
    {
        Iterator<ValeDescuento> it = valesDesc.iterator();

        while (it.hasNext())
        {
            ValeDescuento vale = it.next();

            if (this.productos.containsKey(vale.getDelicia()))
            {
                int unidades =
                        this.productos.get(vale.getDelicia()).getCantidad();

                this.precioTotal = this.precioTotal -
                        unidades*vale.getDelicia().getPrecioU()*
                        vale.getDescuento()/100;
```

```
                it.remove();    //Eliminamos vale descuento aplicado
            }
        }
    }
*/
```

Ejercicio 6 **(2 puntos)**. Implementad en la clase `Menu` el siguiente método:

`public void generaFactura(Bocadillo delDia) throws FacturaException;`

Este método debe imprimir en un fichero de texto llamado `factura_NIF`, donde NIF indica el NIF del cliente que ha encargado el menú, la factura del menú que recibe la invocación del método, teniendo en cuenta el bocadillo del día pasado como parámetro. Si existe cualquier error durante la escritura del fichero, el método debe lanzar una excepción de tipo `FacturaException`.

La factura contendrá en su primera línea el NIF del cliente que ha encargado el menú. Después seguirá detallando el nombre y cantidad de cada delicia encargada, línea a línea. Acto seguido, mostrará en una nueva línea el precio total del menú sin aplicar descuentos. Después, se intentará aplicar el descuento del bocadillo del día (a todas las unidades encargadas de ese bocadillo) y todos los vales descuento de los que pueda disponer el cliente, si este es en realidad de tipo *Premium*. Finalmente, la factura mostrará en una nueva línea el precio total del menú con todos los posibles descuentos ya aplicados.

NOTA: DURANTE EL EXAMEN SE COMENTÓ QUE DONDE EL ENUNCIADO DECÍA NIF, DEBÍA DECIR email, PUESTO QUE EN EL DIAGRAMA UML NO APARECE NINGÚN ATRIBUTO nif PARA LA CLASE CLIENTE.

EN CONSECUENCIA, LA SOLUCIÓN QUE SE MUESTRA A CONTINUACIÓN GESTIONA LA DIRECCIÓN DE CORREO ELECTRÓNICO DEL CLIENTE EN LUGAR DE SU NIF.

```
public class Menu
{
    public void generaFactura(Bocadillo delDia) throws FacturaException
    {
        String filePath = "factura_" + this.cliente.getEmail();

        try
        {
            //Creamos PrintWriter para escritura
            FileOutputStream fos = new FileOutputStream(filePath, false);
            PrintWriter pw = new PrintWriter(fos);

            //Imprimimos email del cliente
            pw.println("EMAIL: " + this.cliente.getEmail());

            int unidadesBocadilloDia = 0;

            //Imprimimos nombre y cantidad encargada de cada delicia
            for (Producto p : this.productos.values())
            {
                pw.println(p.getDelicia().getNombre() + "; cantidad: " +
                        p.getCantidad());

                if (p.getDelicia().getNombre().equals(delDia.getNombre()))
                    unidadesBocadilloDia = p.getCantidad();
            }
```

```
            //Imprimimos precio total sin descuentos
            pw.println("Precio total: " + this.precioTotal);

            //Si bocadillo del día encargado, le aplicamos 25% de descuento
            this.precioTotal = this.precioTotal - unidadesBocadilloDia *
                    delDia.getPrecioU() * 0.25;

            //Si cliente es Premium, aplicamos posibles descuentos
            if (this.cliente instanceof Premium)
            {
                Premium p = (Premium)this.cliente;
                this.aplicarValesDescuentoCliente(p.getValesDescuento());
            }

            //Imprimimos precio total con descuentos
            pw.println("Precio total (con descuentos): " + this.precioTotal);

            //Cerramos PrintWriter
            pw.close();
        }
        catch (IOException e)
        {
            throw new FacturaException("Error escribiendo factura " + filePath);
        }
    }
}
```