PROGRAMACIÓN – 1º DAW

PRUEBA 1ª EVALUACIÓN curso 2019/2020

Nombre y Apellidos:	**Puntuación:**
SOLUCIÓN	

1. (Máx: 0,75ptos.) Implementar un método nuevoAlmacen que pida por pantalla al usuario los datos ya existentes en la clase `Almacen` y éste los introduzca por la entrada estándar, devolviendo un objeto completo. NO pedir datos sobre los Transportistas que trabajan en el nuevo Almacen que se esté creando ni tampoco sobre los Pedidos o Productos que se guardarán ahí en el futuro.

```
public static Almacen nuevoAlmacen() {
    Almacen ret = new Almacen();
    Scanner in = new Scanner(System.in);
    System.out.println("Introduzca el nombre para el nuevo almacen:");
    ret.setNombre(in.nextLine());
    System.out.println("Introduzca la capacidad para el nuevo:");
    ret.setCapacidad(in.nextInt());
    ret.setIdentificador(Utilidades.numAlmacenes + 1);
    return ret;
}
```

2. (Máx: 0,75ptos.) En la función principal, definir una variable de tipo colección de objetos `Almacen` que inicialmente contenga los 3 almacenes ya existentes en el sistema. Luego, pedir al usuario que introduzca varios nuevos almacenes (al menos uno) que se irán añadiendo a esa colección. Cuando el usuario termine de introducir almacenes, el programa principal mostrará de uno en uno los datos de cada objeto `Almacen` de la colección.

```
ArrayList<Almacen> almacenes = Almacen.convertir(Utilidades.ALMACENES);
        for (int i = 0; i < 3; i++) {
            System.out.println("Introduzca un nuevo almacen:");
            almacenes.add(Almacen.nuevoAlmacen());
        }
        System.out.println("Los almacenes guardados en el programa son:");
        for (Almacen a : almacenes) {
            System.out.println("Datos de almacen:" + a);
        }
```

3. (Máx: 1pto.) Modificar la clase `Cliente` para que pueda gestionarse toda la siguiente información:

   "Me llamo JuanCa Bobón y vivo en Avenida Real 50 de Madriz, tengo 80 años y mi profesión es Amodecasa. Para contactar conmigo, llamar al teléfono 915512345 o escribir un email a

juanqui_spain@gmail.com. Mi modo de pago preferido es por tarjeta ('T') y sí deseo subscribirme al boletín de noticias."

Realizar los cambios necesarios (constructor/es, getters/setters, etc.) en la clase `Cliente` sin modificar la información ya existente.

```
    private int edad;
    private String profesion;
    private char tipoPago;
    private boolean suscripcion = false;

    public Cliente(int nCliente, String nombre, String direccion, String email, String
telefono, int edad, String profesion, char tipoPago) {
        this.nCliente = nCliente;
        this.nombre = nombre;
        this.direccion = direccion;
        this.email = email;
        this.telefono = telefono;
        this.edad = edad;
        this.profesion = profesion;
        this.tipoPago = tipoPago;
    }

    public int getEdad() {
        return edad;
    }

    public void setEdad(int edad) {
        this.edad = edad;
    }

    public String getProfesion() {
        return profesion;
    }

    public void setProfesion(String profesion) {
        this.profesion = profesion;
    }

    public char getTipoPago() {
        return tipoPago;
    }

    public void setTipoPago(char tipoPago) {
        this.tipoPago = tipoPago;
    }

    public boolean isSuscripcion() {
        return suscripcion;
    }

    public void setSuscripcion(boolean suscripcion) {
        this.suscripcion = suscripcion;
    }
```

4. (Máx: 1,5pto.) Implementar un método que, dado un Producto, devuelva una lista de Clientes que hayan pedido ese Producto alguna vez. EXTRA: +0,5 si la lista devuelta no tiene duplicados de Clientes.

```
public ArrayList<Cliente> clientesPidieron(Producto p) {
        ArrayList<Cliente> ret = new ArrayList<Cliente>();
        int contLP = 0; //contador para recorrer las lineas de pedidos
        //Recorremos todos los clientes
        for (int i = 0; i < Utilidades.numClientes; i++) {
            Cliente c = Utilidades.CLIENTES[i];
            for (int j = contLP; j < Utilidades.numLineas; j++) {
                contLP++;
                LineaPedido lp = Utilidades.LINEAS[j];
                if (lp.getProducto().equals(p)) {
                    //Hay alguna linea que contiene el producto
                    if (lp.idPedido().getCliente().equals(c)) {
                        //El pedido pertenece al cliente sobre el que iteramos
                        if (!ret.contains(c)) //para evitar duplicados
                        {
                            ret.add(c);
                        }
                        break;
                    }
                }
            }
        }
        return ret;
    }
```

5. (Máx: 1pto.) Implementar un método que devuelva la lista de Productos a los que se les ha aplicado algún descuento alguna vez. EXTRA: +0,5 si la lista devuelta no tiene duplicados de Productos.

```
public ArrayList<Producto> productosPromocionadosAlgunaVez() {
        ArrayList<Producto> ret = new ArrayList<Producto>();
        for (int j = 0; j < Utilidades.numDescuentos; j++) {
            Descuento d = Utilidades.DESCUENTOS[j];
            for (int k = 0; k < d.getProductos().size(); k++) {
                if (!ret.contains(d.getProductos().get(k))) {
                    ret.add(d.getProductos().get(k));
                }
            }
        }
        return ret;
    }
```

6. (Máx: 1pto.) En la función principal, implementar un código para que se muestre el nombre, el código y las existencias de cada producto que se almacena en el “Polígono”.

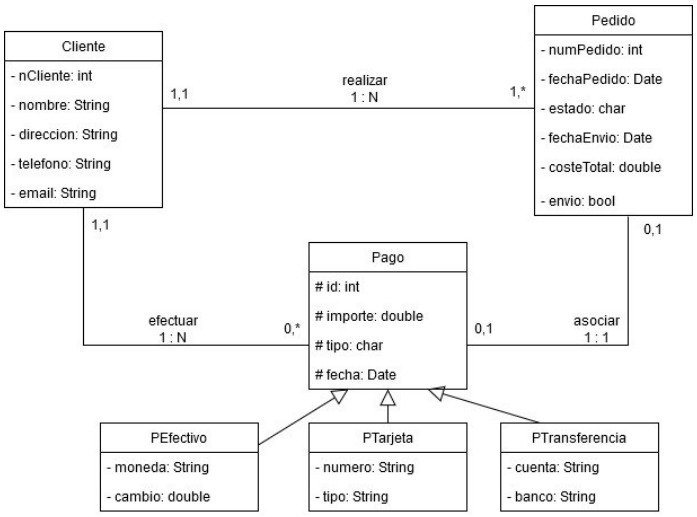
```
Almacen poligono = Almacen.buscarByNombre("Poligono");
        if (poligono != null) {
            System.out.println("Los productos del almacen \"Poligono\" son:");
            for (int cont = 0; cont < Utilidades.numProductos; cont++) {
                Producto p = Utilidades.PRODUCTOS[cont];
                if (p.getAlmacen().equals(poligono)) {
```

```
                    System.out.println(p);
                }
            }

        }
```

7. (Máx: 1pto.) Implementar un código para buscar un Empleado por su nombre. Se pide al usuario que introduzca una cadena de caracteres de al menos 3 caracteres y se buscará en la lista de empleados existentes, mostrando los datos de cada empleado cuyo nombre contenga esa cadena.

```
protected static Empleado buscarByNombre(String nombre) {

        for (Empleado a : Utilidades.EMPLEADOS) {

            if (a.getNombre().equals(nombre)) {

                return a;

            }

        }

        return null;

    }
```

8. (Máx: 1,5ptos.) Implementar una nueva clase `Pago` y sus subclases `PEfectivo`, `PTarjeta`, `PTransferencia`, para que encaje con la siguiente ampliación al diagrama de Clases:



Clase Pago.java

```
public class Pago {

    protected int identificador;
    protected char tipo;
    protected double importe;
    protected Date fecha;

    public Pago() {
```

```
    }

    public Pago(Pago p) {
        this.identificador = p.getIdentificador();
        this.tipo = p.getTipo();
        this.importe = p.getImporte();
        this.fecha = p.getFecha();
    }

    public Pago(int identificador, char tipo, double importe, Date fecha) {
        this.identificador = identificador;
        this.tipo = tipo;
        this.importe = importe;
        this.fecha = fecha;
    }

    protected int getIdentificador() {
        return identificador;
    }

    protected void setIdentificador(int identificador) {
        this.identificador = identificador;
    }

    protected char getTipo() {
        return tipo;
    }

    protected void setTipo(char tipo) {
        this.tipo = tipo;
    }

    protected double getImporte() {
        return importe;
    }

    protected void setImporte(double importe) {
        this.importe = importe;
    }

    protected Date getFecha() {
        return fecha;
    }

    protected void setFecha(Date fecha) {
        this.fecha = fecha;
    }

}
```

Clase PEfectivo.java

```
public class PEfectivo extends Pago {

    private String moneda;
    private double cambio;
```

```
    public PEfectivo(String moneda, double cambio, int identificador, char tipo, double
importe, Date fecha) {
        super(identificador, tipo, importe, fecha);
        this.moneda = moneda;
        this.cambio = cambio;
    }

    public PEfectivo(Pago p, double importe, Date fecha) {
        super(p);
        this.moneda = moneda;
        this.cambio = cambio;
    }

    public String getMoneda() {
        return moneda;
    }

    public void setMoneda(String moneda) {
        this.moneda = moneda;
    }

    public double getCambio() {
        return cambio;
    }

    public void setCambio(double cambio) {
        this.cambio = cambio;
    }

}
```

Clase PTtransferencia.java

```
public class PTransferencia extends Pago {

    private String cuenta;
    private String banco;

    public String getCuenta() {
        return cuenta;
    }

    public void setCuenta(String cuenta) {
        this.cuenta = cuenta;
    }

    public String getBanco() {
        return banco;
    }

    public void setBanco(String banco) {
        this.banco = banco;
    }

    public PTransferencia(Pago p, String cuenta, String banco) {
        super(p);
        this.cuenta = cuenta;
        this.banco = banco;
```

```
    }

    public PTransferencia(String cuenta, String banco, int identificador, char tipo,
double importe, Date fecha) {
        super(identificador, tipo, importe, fecha);
        this.cuenta = cuenta;
        this.banco = banco;
    }

}
```

Clase PTarjeta.java

```
public class PTarjeta extends Pago {

    private String numero;
    private String tipoTarjeta;

    public String getNumero() {
        return numero;
    }

    public void setNumero(String numero) {
        this.numero = numero;
    }

    /**
     * Si lo llamamos PTarjeta.tipoTarjeta tendremos un problema con
     * Pago.tipoTarjeta porque son campos diferentes, de clases distintas, de
     * diferente tipoTarjeta, pero con el mismo nombre La solución es renombrar
     * el campo, por ejemplo String PTarjeta.tipoTarjeta
     */
    public String getTipoTarjeta() {
        return tipoTarjeta;
    }

    public void setTipoTarjeta(String tipoTarjeta) {
        this.tipoTarjeta = tipoTarjeta;
    }

}
```

9. (Máx: 1,5ptos.) Añadir un nuevo campo a la clase `Cliente` de nombre "`saldo`" que medirá la cantidad de euros que tiene el cliente disponibles en el sistema, inicialmente al valor 0,00. Cuando un cliente realiza un pedido puede elegir cómo lo va a pagar: o bien en efectivo ('E') o bien con tarjeta ('T') o bien por transferencia bancaria ('B'), tanto si lo recoge personalmente como si se lo envían a casa. Si no realiza el pago en el momento del pedido, el valor del coste total de éste se decrementa en el saldo del cliente al finalizarlo. Los clientes pueden realizar Pagos en cualquier momento y, cuando lo hacen, o bien van asociados a un Pedido (es decir, lo pagan en el momento del pedido) o no (los clientes hacen pagos para saldar una deuda o para incrementar su saldo).

   CU: Realizar Pago

   Precondiciones: el cliente que realiza el pago ha de estar registrado en el sistema.

   Postcondiciones: se registra un nuevo pago en el sistema y se actualiza el saldo del cliente.

Pasos:

1. El cliente introduce el importe del pago en euros y el tipo de pago (‘E’/’T’/’B’)
2. El sistema comprueba que el importe es positivo y el tipo de pago válido.
3. El sistema crea un nuevo `Pago` con el importe, el tipo y la fecha del momento. Además:
    a. si el pago es en el momento de compra se establece el campo `Pago.pedido`
4. El sistema actualiza el saldo del cliente, incrementando su valor con la cantidad pagada.
5. Se notifica al cliente mediante un mensaje que el pago ha sido correcto.

Excepciones:

2. El importe o el tipo de pago no es válido. Se muestra mensaje al cliente con las opciones de valores válidos y éste debe volver a introducirlos.

Implementar el método `void Cliente.realizarPago(double importe, char tipo, Pedido p)` que gestione el caso de uso descrito.

```
public void realizarPago(double importe, char tipo, Pedido p) {
    Scanner in = new Scanner(System.in);
    if (importe < 0) {
        System.out.println("El importe del pago no es correcto.");
        return;
    }
    if (tipo != 'B' && tipo != 'E' && tipo != 'T') {
        System.out.println("El importe del pago no es correcto.");
        return;
    }
    Pago nuevo = new Pago();
    nuevo.setIdentificador(Utilidades.numPagos);
    nuevo.setFecha(Date.valueOf(LocalDate.now()));
    nuevo.setImporte(importe);
    nuevo.setTipo(tipo);

    switch (tipo) {
        case 'B':
            System.out.println("Introduzca el banco del pago por transferencia:");
            String banco = in.nextLine();
            System.out.println("Introduzca la cuenta para el pago por transferencia:");
            String cuenta = in.nextLine();
            PTransferencia nuevoPB = new PTransferencia(nuevo, cuenta, banco);
            if (p != null) {
                p.setPago(nuevoPB);
            }
            this.setSaldo(saldo + importe);
            break;
        case 'E':
            System.out.println("Introduzca la moneda del pago en efectivo:");
            String moneda = in.nextLine();
            System.out.println("Introduzca el cambio del pago en efectivo (x.xx):");
            double cambio = in.nextDouble();
            PEfectivo nuevoPE = new PEfectivo(nuevo, moneda, cambio);
            if (p != null) {
                p.setPago(nuevoPE);
            }
            this.setSaldo(saldo + importe);
            break;
        case 'T':
```

```
                System.out.println("Introduzca el tipo de la tarjeta para el pago por
tarjeta:");
                String tipoTarjeta = in.nextLine();
                System.out.println("Introduzca el numero para el pago por tarjeta:");
                String numero = in.nextLine();
                PTarjeta nuevoPT = new PTarjeta(nuevo, numero, tipoTarjeta);
                if (p != null) {
                    p.setPago(nuevoPT);
                }
                this.setSaldo(saldo + importe);
                break;
        }
        System.out.println("El pago se ha registrado con éxito.");

    }
```

Descripción del Sistema a modelar: “Gestión de un supermercado”

El sistema para el supermercado maneja los **pedidos** que los **clientes** realizan. Los clientes se registran en el sistema, indicando sus datos básicos: *nombre*, *dirección, email* y *teléfono*, y se les asigna un *número de cliente* que es único.

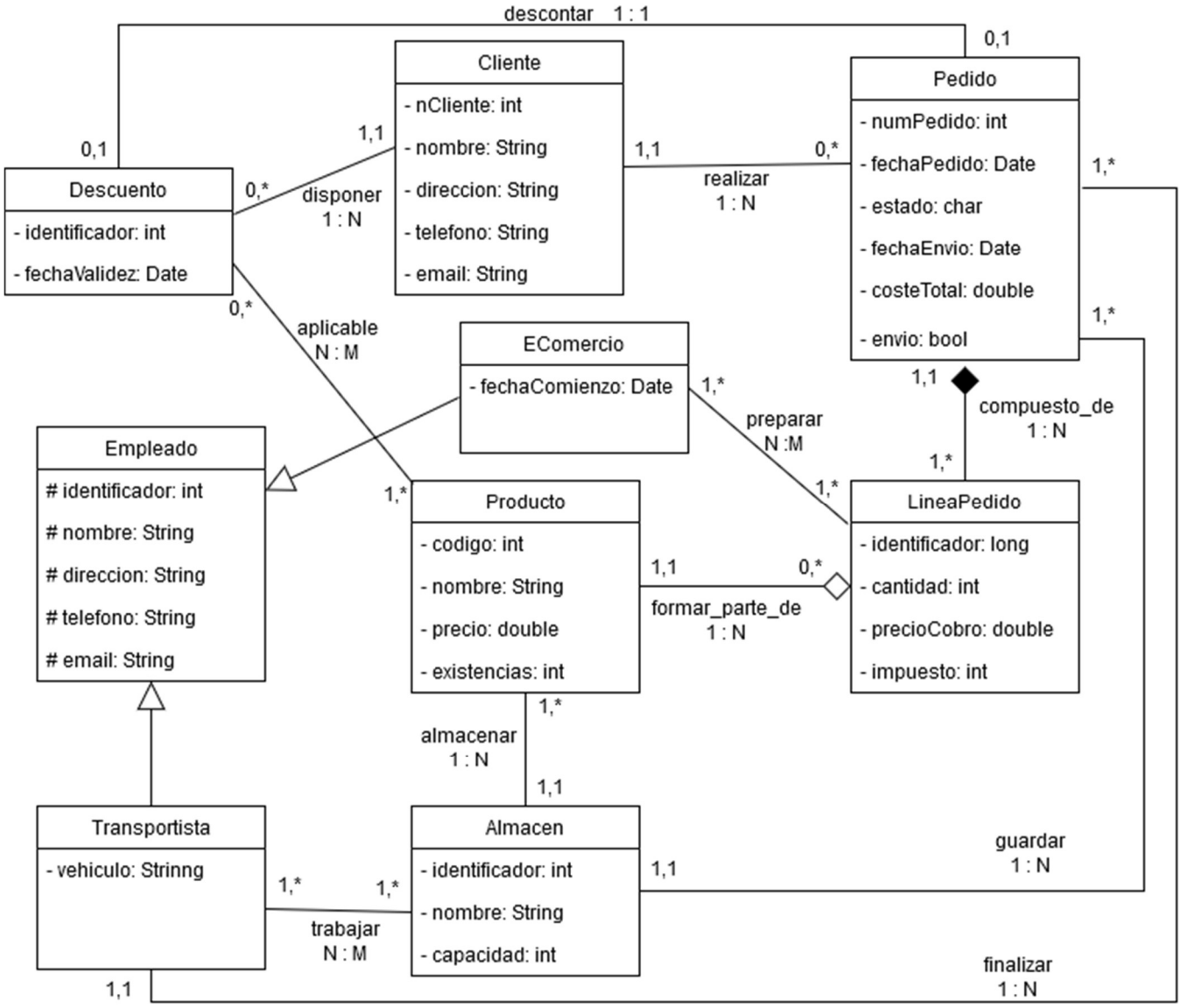
Los pedidos se identifican con un *número de pedido* único, se almacena la *fecha de pedido*, el *estado* en que se encuentra (codificado con una letra), la *fecha de envío* y el *coste total* del pedido en euros. Al realizar el pedido, en el paso final, el cliente marcará si el pedido será *enviado* a su dirección o si se recogerá en un **almacén**. Los pedidos constan de una o varias **líneas de pedido** y éstas están compuestas por un solo **producto**. Cada línea de pedido se registra con un *identificador* propio (así que habrá muchísimas líneas con el paso del tiempo) y se estructura en la forma: *cantidad* * *precio de cobro* + *impuesto*, donde se expresa la cantidad del producto de que se compone, el precio en euros que se cobra por unidad y el impuesto a aplicar (8%, 16% ó 24%). Todos los productos del supermercado tienen un *código* único que es un número y un *nombre*, así como un *precio unitario* expresando en euros.

El cliente, para realizar sus pedidos, va comprado productos. Para ello selecciona el producto que desea comprar del catálogo, marca la cantidad que quiere adquirir y pulsa “Añadir al carro”. El sistema comprueba las existencias y si quedan, añade una línea al pedido. Cuando el cliente termina de añadir líneas a su pedido, procede a marcar si quiere recoger personalmente el pedido en alguno de los almacenes o prefiere que se lo envíe un transportista. En ambos casos, un transportista requerirá una confirmación de recogida/recepción del pedido por parte del cliente. Los clientes disponen en algunas ocasiones de **descuentos** personales a aplicar en un pedido al precio de cobro de algunos productos. De los descuentos se tiene un *identificador* y una *fecha de validez*, sólo pueden ser *usados* una única vez y van relacionados a uno o a varios productos (así que dependen de éste/os), aunque se refleje en el sistema a través del precio de cobro de la línea correspondiente en el pedido. Hay productos y clientes que nunca tendrán descuento, pero algunos productos son muy promocionados.

Por otro lado, el sistema del supermercado almacena los datos de los **empleados**: *identificador* único, *nombre*, *dirección*, *teléfono* y *email*. Hay 2 tipos de empleados en el supermercado:

1. los **empleados de comercio**, que trabajan físicamente en el supermercado. De ellos se almacena la *fecha* en que comenzaron a trabajar en el supermercado. Entre uno solo o dos como mucho se encarga/n de preparar los pedidos: por cada línea de pedido, un empleado de ellos va al almacén que almacena el producto, lo toma (reduciendo el stock) y lo embala, conformando así un paquete completo que contiene todo el pedido.
2. los **transportistas**, de los que guardamos su *vehículo*, realizan varias funciones: gestionan los envíos de los pedidos a las direcciones de los clientes y confirman su recepción, también cuando se trata de recogida en persona en algún almacén. Pero también son los encargados de recibir los productos desde los proveedores y colocarlos en el almacén que les corresponde, actualizando el stock de existencias. Los almacenes van *identificados*, tienen un *nombre* y una *capacidad* expresada en metros cúbicos mediante un número entero. En cada almacén trabajan los mismos transportistas y los transportistas trabajan siempre en los mismos almacenes.

Diagrama de clases UML para la descripción:



Tablas:

Clientes

int nCliente	String nombre	String direccion	String email	String telefono
1	Luis	Gijon	luis@educantabria.es	942779900
2	Ana	Madrid	ana@gmail.com	912331188
3	Bruno	Barcelona	brunob@hotmail.es	932432245
4	Carla	Santander	carla@outlook.com	975073211
5	Ramona	Valencia	ramonnaval@gmail.com	942779900

Cliente-Pedido *realizar* guardar *finalizar* *descontar*

idCliente	idPedido	fechaPedido	estado	fechaEnvio	envio?	costeTotal	idAlmacen	idTransportista	idDescuento
1	1	12/10/2017	R	13/10/2017	NO	463,99	1	8	NULL
2	2	15/10/2017	R	15/10/2017	NO	10,15	1	9	NULL
3	3	20/10/2017	R	22/10/2017	SI	247,98	3	8	NULL
3	4	01/11/2017	R	02/10/2017	SI	813,83	1	10	1
4	5	12/11/2017	R	12/11/2017	NO	59,73	3	11	2
1	6	26/11/2017	R	26/11/2017	NO	39,74	3	9	NULL
5	7	07/12/2017	R	08/12/2017	SI	65,34	3	10	NULL
5	8	23/12/2017	C	NULL	SI	123,99	NULL	NULL	NULL

Productos *almacenar*

int codigo	String nombre	double precio	int existencias	idAlmacen
1	ordenador	399,99	4	3
2	bolsa de patatas	3,99	43	1
3	bote de caramelos	2,99	98	1
4	boligrafo	0,99	125	2
5	leche	0,59	78	1
6	consola	99,99	3	3
7	papel higienico	1,45	22	3
8	garrafa agua 5L	2,30	34	3
9	tableta cocholate	1,25	10	1

Pedido-Linea *compuesto_de* *forma_parte_de* *preparar*

idPedido	idLinea	idProducto	PrecioCobro	Cantidad	Impuesto	idEComercio
1	1	1	399,99	1	16	5
2	2	2	3,99	1	16	1
2	3	3	2,99	1	16	6
2	4	5	0,59	3	16	3
3	5	6	99,99	2	24	5
4	6	1	350,79	2	16	5
5	7	4	0,97	50	16	3
5	8	3	2,99	1	16	4
6	9	8	2,30	16	8	2
7	10	8	2,30	20	8	2
7	11	7	1,45	10	8	1
8	12	6	99,99	1	24	NULL

Descuento *disponer* *aplicable*

idDescuento	fechaValidez	idCliente	idProductos
1	31/12/2017	3	1, 6
2	31/12/2017	4	4
3	31/12/2017	1	2, 5, 8, 9

EComercio

int identificador	String nombre	String direccion	String telefono	String email	Date fechaComienzo
1	Raquel	Castro Urdiales	942653309	raqcastro86@gmail.com	20/08/2013
2	Marcos	Castro Urdiales	942321124	mzorth@gmail.com	08/09/2018
3	Laura	Samano	942630083	lausamano99@gmail.com	10/06/2019
4	Rocio	Laredo	942125124	roci222@hotmail.com	02/01/2015
5	Juan	Santander	683319233	jsantanderperez@gmail.com	28/04/2012
6	Veronica	Bilbao	623246233	vergarciarodriguez@gmail.com	12/11/2018
7	Maria	Castro Urdiales	942628882	marimarcastro@hotmail.es	15/01/2019

Transportista

int identificador	String nombre	String direccion	String telefono	String email	String vehiculo
8	Marga	Madrid	642773309	margarus87@gmail.com	Seat Ibiza
9	Jose	Castro Urdiales	672620300	josecar2000@gmail.com	Peugeot 307
10	Simon	Torrelavega	942871993	simonkantabria@hotmail.com	Audi A2
11	Sergio	Sevilla	924567188	sergisev1985@gmail.com	Renault Berlingo

Almacen

int identificador	String nombre	int capacidad
1	Sotano	100
2	Piso2	50
3	Poligono	250

trabajar

Almacen-Transportista

idAlmacen	idTransportista
1	8
1	9
1	10
2	8
3	8
3	9
3	10
3	11