PROGRAMACIÓN – 1º DAW

PRUEBA RECUPERACIÓN 1ª EVALUACIÓN curso 2019/2020

Nombre y Apellidos:	Puntuación:

1. (Máx: 1pto.) Implementar el constructor de copia para la clase `Cliente` y utilizarlo en la función principal para crear un nuevo objeto de esta clase a partir de los datos de otro objeto ya existente del fichero Utilidades.java.

```
Public Cliente(Cliente c){
  this.nCliente = c.getnCliente();
  this.nombre = c.getNombre();
  this.direccion = c.getDireccion();
  this.email = c.getEmail();
  this.telefono = c.getTelefono();
  this.edad = c.getEdad();
  this.profesion = c.getProfesion();
  this.tipoPago = c.getTipoPago();
  this.suscripcion = c.isSuscripcion();
 this.saldo = c.getSaldo();

}

Cliente c1 = new Cliente(Utilidades.CLIENTES[0]);
```

2. (Máx: 2ptos.) Implementar una nueva clase de nombre `Categoria` para los productos, en la que se almacena un identificador que es único, el código de la categoría expresado mediante una letra y una descripción de la categoría. Añadir los métodos de acceso a tales atributos, los 3 constructores recomendados y un método nuevaCategoria que pida por pantalla al usuario todos los datos y éste los introduzca por la entrada estándar, devolviendo un objeto completo.

```
public class Categoria {

    private long id;
    private char codigo;
    private String descripcion;

    public Categoria() {
    }

    public Categoria(long id, char codigo, String descripcion) {
        this.id = id;
        this.codigo = codigo;
        this.descripcion = descripcion;
    }

    public Categoria(Categoria c) {
        this.id = c.getId();
```

```
            this.codigo = c.getCodigo();
            this.descripcion = c.getDescripcion();
        }

        public long getId() {
            return id;
        }

        public void setId(long id) {
            this.id = id;
        }

        public char getCodigo() {
            return codigo;
        }

        public void setCodigo(char codigo) {
            this.codigo = codigo;
        }

        public String getDescripcion() {
            return descripcion;
        }

        public void setDescripcion(String descripcion) {
            this.descripcion = descripcion;
        }

        public static Categoria nuevaCategoria() {
            Categoria ret = new Categoria();
            Scanner in = new Scanner(System.in);
            System.out.println("Introduzca el codigo para la categoria: (una letra)");
            ret.setCodigo(Character.valueOf(in.next().charAt(0)));
            System.out.println("Introduzca la descripcion para la nueva categoria :");
            ret.setDescripcion(in.nextLine());
            ret.setId(Utilidades.numCategorias + 1);
            return ret;
        }

    }
```

Modificar también la clase `Producto` para incluir la única categoría a la que puede pertenecer un producto, sus métodos de acceso para ese nuevo dato e implementar un nuevo constructor que tenga como argumentos todos los datos, incluyendo la categoría del producto.

```
    private Categoria categoria;

    public Producto(int codigo, String nombre, double precio, int existencias, Almacen
almacen, Categoria cat) {
            this.codigo = codigo;
            this.nombre = nombre;
            this.precio = precio;
            this.existencias = existencias;
            this.almacen = almacen;
            this.categoria = cat;
        }
```

3. (Máx: 2pto.) Implementar una nueva clase de nombre `Comercial` que sea un nuevo tipo de Empleado y del que se quiere guardar, además de los datos propios de Empleado, la información necesaria para que pueda gestionarse todo lo extraído del siguiente comentario:

   "Los comerciales del supermercado nos movemos por distintas provincias. En mis 7 años de experiencia en el sector nunca antes hasta ahora había tenido un teléfono de contacto de total disposición, me parece una lata después de haberles indicado ya mi horario de disponibilidad. Mandaré un mail con una queja a mi jefe, aunque no es más que un empleado como yo."

   Implementar los 4 constructores siguientes: por defecto, de copia, con todos los argumentos propios de los comerciales más un objeto Empleado, con todos los argumentos propios de los comerciales más los generales de Empleado.
   Implementar también los getters/setters requeridos.

```
public class Comercial extends Empleado {

    private ArrayList<String> provicias = new ArrayList<String>();
    private int aniosExperiencia;
    private String telefonoContacto;
    private String disponibilidad;
    private Empleado jefe;

    public Comercial() {
        super();
    }

    public Comercial(Comercial c) {
        this.identificador = c.getIdentificador();
        this.nombre = c.getNombre();
        this.direccion = c.getDireccion();
        this.telefono = c.getTelefono();
        this.email = c.getEmail();
        this.aniosExperiencia = c.getAniosExperiencia();
        this.telefonoContacto = c.getTelefonoContacto();
        this.disponibilidad = c.getDisponibilidad();
        this.jefe = c.getJefe();
    }

    public Comercial(Empleado e, int aniosExperiencia, String telefonoContacto, String
disponibilidad, Empleado jefe) {
        super(e);
        this.aniosExperiencia = aniosExperiencia;
        this.telefonoContacto = telefonoContacto;
        this.disponibilidad = disponibilidad;
        this.jefe = jefe;
    }

     public Comercial(int identificador, String nombre, String direccion, String
telefono, String email, int aniosExperiencia, String telefonoContacto, String disponibilidad,
Empleado jefe) {
        super(identificador, nombre, direccion, telefono, email);
        this.aniosExperiencia = aniosExperiencia;
        this.telefonoContacto = telefonoContacto;
        this.disponibilidad = disponibilidad;
        this.jefe = jefe;
    }
```

```
    public ArrayList<String> getProvicias() {
        return provicias;
    }

    public void setProvicias(ArrayList<String> provicias) {
        this.provicias = provicias;
    }

    public int getAniosExperiencia() {
        return aniosExperiencia;
    }

    public void setAniosExperiencia(int aniosExperiencia) {
        this.aniosExperiencia = aniosExperiencia;
    }

    public String getTelefonoContacto() {
        return telefonoContacto;
    }

    public void setTelefonoContacto(String telefonoContacto) {
        this.telefonoContacto = telefonoContacto;
    }

    public String getDisponibilidad() {
        return disponibilidad;
    }

    public void setDisponibilidad(String disponibilidad) {
        this.disponibilidad = disponibilidad;
    }

    public Empleado getJefe() {
        return jefe;
    }

    public void setJefe(Empleado jefe) {
        this.jefe = jefe;
    }
}
```

4. (Máx: 1pto.) Implementar un método que devuelva la lista de Transportistas que trabajen en el almacén “Sotano”.

```
public ArrayList<Transportista> transportistasSotano() {
        ArrayList<Transportista> ret = new ArrayList<Transportista>();
        Almacen sotano = Almacen.buscarByNombre("Sotano");
        if (sotano != null) {
            for (Empleado e : Utilidades.EMPLEADOS) {
                if (e instanceof Transportista) {
                    for (Almacen a : ((Transportista) e).getAlmacenes()) {
                        if (a.equals(sotano)) {
                            if (!ret.contains(a)) {
                                ret.add((Transportista) e);
                            }
                        }
                    }
                }
```

```
                }
            }
            return ret;
        }
```

5. (Máx: 1pto.) Implementar un método que devuelva la lista de Clientes que aún no se han suscrito.

```
    public ArrayList<Cliente> clientesNoSuscritos() {
            ArrayList<Cliente> ret = new ArrayList<Cliente>();
            for (Cliente c : Utilidades.CLIENTES) {
                if (!c.isSuscripcion()) {
                    if (!ret.contains(c)) {
                        ret.add(c);
                    }
                }
            }
            return ret;
        }
```

6. (Máx: 1pto.) En la función principal, implementar un código para que se muestre el nombre, el código y las existencias de los productos que no han formado parte nunca de un descuento.

```
    ArrayList<Producto> productosEnAlgunDescuento = new ArrayList<Producto>();
            ArrayList<Producto> productosEnNingunDescuento =
Producto.convertir(Utilidades.PRODUCTOS);
            for(Descuento d: Utilidades.DESCUENTOS){
                for(Producto p: d.getProductos())
                    if(!productosEnAlgunDescuento.contains(p))
                        productosEnAlgunDescuento.add(p);
            }
            for(Producto p: productosEnAlgunDescuento){
                if(productosEnNingunDescuento.contains(p)){
                    productosEnNingunDescuento.remove(p);
                }
            }
            System.out.println("Los productos que no han formado parte nunca de un descuento
son:");
            if(productosEnNingunDescuento.isEmpty())
                System.out.println("No hay ninguno.");
            else{
                for(Producto p: productosEnNingunDescuento)
                    System.out.println(p);
            }
```

7. (Máx: 2ptos.) Añadir el código necesario para implementar el siguiente caso de uso
   CU: Finalizar Pedido
   Precondiciones: hay un pedido ya preparado y el cliente que lo realizó lo recibe a través de un transportista.
   Postcondiciones: el pedido queda finalizado por el transportista y el pedido deja de estar físicamente en manos del supermercado. En caso de pago en efectivo se registra un nuevo pago en el sistema y se actualiza el saldo del cliente.
   Pasos:
   1. a) El cliente llega al almacén donde se guarda su pedido y lo recibe un transportista. Éste introduce algún dato del cliente en el sistema (de entre los siguientes: número de cliente, nombre, email o teléfono) y accede a los datos del cliente y a los pedidos de ese cliente.

1. b) El cliente recibe en su domicilio al transportista que llega con su pedido. Éste introduce algún dato del cliente en el sistema (de entre los siguientes: número de cliente, nombre, email o teléfono) y accede a los datos del cliente y a los pedidos de ese cliente.
2. a) El transportista selecciona el código del pedido que reclama el cliente y lo recoge de las estanterías del almacén.
2. b) El transportista selecciona el código del pedido que va a entregar al cliente.
3. Si el pedido no fue previamente pagado o si existe alguna deuda de ese cliente, el cliente deberá realizar un nuevo Pago (ya implementado) en efectivo o por tarjeta y de importe igual o superior a la suma de su deuda más el coste total del pedido en curso o no recibirá su pedido. Cuando su saldo sea mayor o igual a 0.00 euros, el cliente recibe físicamente el paquete con su pedido, estableciendo el campo fecha de envío del pedido a la fecha actual.
4. El sistema establece el estado del pedido a 'R' (recibido) y se establece también el transportista del pedido con el transportista que lo finaliza. Se informa al transportista del éxito de estas operaciones y se imprimen 2 copias de los datos del pedido, una que firma el transportista y entrega al cliente y la otra la firma el cliente y la entrega al transportista.

Excepciones:

1. a) El sistema no encuentra el cliente con los datos. Se muestra mensaje al transportista y se mostrará la lista de clientes para seleccionar el que corresponda con los datos que el cliente le facilite.
1. b) El cliente no está en el domicilio. Se marca el estado del pedido como 'F' (entrega fallida), se establece la fecha de envío del pedido a la fecha actual y se acaba el caso de uso.
2. a) El pedido no está en el almacén porque se encuentra en otro. Se avisa al usuario, se le indica que cambie de almacén y se acaba el caso de uso sin modificaciones.
2. El sistema no encuentra el pedido. Se muestra la lista de pedidos y se selecciona el que corresponda.
3. No se realiza Pago o sigue habiendo deuda. Se marca el estado del pedido como 'F' (entrega fallida). Si además se trata de una entrega a domicilio se establece la fecha del pedido a la fecha actual y se acaba el caso de uso.

Se sugiere implementar todo el caso de uso descrito anteriormente mediante un único método `void finalizarPedido(Pedido p, Cliente c)` que lo gestione completamente desde la clase `Transportista`.

```
/**
     * *
     * Función que finaliza un Poledido p de un cliente c por el transportista
     * actual En caso de éxito, el pedido queda finalizado por el transportista
     * y el pedido deja de estar físicamente en manos del supermercado. En caso
     * de pago en efectivo se registra un nuevo pago en el sistema y se
     * actualiza el saldo del cliente.
     *
     * @param p El pedido que pretende finalizar el Transportista (pedido ya
     * preparado)
     * @param c El cliente que recibe el pedido (cliente que realizó el pedido
     * p)
     * @return true en caso de éxito o false en caso contrario
     */
    public boolean finalizarPedido(Pedido p, Cliente c) {
        Scanner in = new Scanner(System.in);
        //Paso1: Se comprueba que existe el Cliente c. Si no, se muestra la lista de
clientes y se selecciona el correcto
        int clienteConfirmado = -1;
```

```
            if (!Cliente.convertir(Utilidades.CLIENTES).contains(c)) {
                System.out.println("No se encuentra el cliente.");
                do {
                    System.out.println("Seleccione el cliente:");
                    for (int i = 0; i < Utilidades.numClientes; i++) {
                        System.out.println((i + 1) + ": " + Utilidades.CLIENTES[i]);
                        System.out.println("Pulse 0 para CANCELAR");
                    }
                    clienteConfirmado = in.nextInt();
                    if (clienteConfirmado == 0) {
                        return false;
                    }
                } while (clienteConfirmado < 1 || clienteConfirmado >
Utilidades.numClientes);
                c = Utilidades.CLIENTES[clienteConfirmado - 1];
            }
            //Paso2: Se comprueba que existe el Pedido p y que pertenece al cliente. Si no,
se muestra la lista de pedidos y se selecciona el correcto
            int pedidoConfirmado = -1;
            if (!Pedido.convertir(Utilidades.PEDIDOS).contains(p) || p.getCliente() != c) {
                System.out.println("No se encuentra el pedido.");
                do {
                    System.out.println("Seleccione el pedido:");
                    for (int i = 0; i < Utilidades.numPedidos; i++) {
                        System.out.println((i + 1) + ": " + Utilidades.PEDIDOS[i]);
                        System.out.println("Pulse 0 para CANCELAR");
                    }
                    pedidoConfirmado = in.nextInt();
                    if (pedidoConfirmado == 0) {
                        return false;
                    }
                } while (pedidoConfirmado < 1 || pedidoConfirmado > Utilidades.numPedidos);
                p = Utilidades.PEDIDOS[pedidoConfirmado - 1];
            }
            //Paso3: Si el pedido no se ha pagado o el cliente tiene deuda
            if (c.getSaldo() < 0 || p.getPago() == null) {
                System.out.println("Su pedido no ha sido pagado o tiene una deuda.");
                do {
                    double importe = 0;
                    char tipoPago = ' ';
                    do {
                        System.out.println("Debe ralizar un pago. Seleccione el tipo de pago
T=tarjeta o E=efectivo.");
                        tipoPago = in.next().charAt(0);
                        System.out.println("Indique el importe del pago en euros (x.xx):");
                        importe = in.nextDouble();
                        if (importe <= 0 || (tipoPago != 'T' && tipoPago != 'E')) {
                            System.out.println("Error: importe o tipo de pago incorrecto.");
                        }
                    } while (importe <= 0 || (tipoPago != 'T' && tipoPago != 'E'));
                    if (!c.realizarPago(importe, tipoPago, p)) {
                        p.setEstado('F');
                        if (p.isEnvio()) {
                            p.setFechaEnvio(Date.valueOf(LocalDate.now()));
                        }
                        return false;
                    }

                } while (c.getSaldo() < 0 || p.getPago() == null);
```

```
            }
            //El cliente no tiene deuda y el pedido ha sido pagado -> entrega física del
pedido al cliente
            p.setFechaEnvio(Date.valueOf(LocalDate.now()));
            //Paso 4: se establece el estado a 'Recibido' y el Transportista finaliza el
pedido exitosamente
            p.setEstado('R');
            p.setTransportista(this);
            return true;
        }
```

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