Práctica 3

Introducción a la Programación Orientada a Objetos con Java

Inicio: A partir del 24 de febrero.
Duración: 3 semanas.
Entrega: En Moodle, una hora antes del comienzo de la siguiente práctica según grupos (semana del 16 de marzo)
Peso de la práctica: 20%

El objetivo de esta práctica es introducir al alumno en la programación orientada a objetos con el lenguaje Java, pidiéndole que desarrolle de forma incremental varias clases en Java (incluyendo sus pruebas y documentación) que implementen diversos componentes software de una pequeña aplicación de ventas.
En el desarrollo de esta práctica se utilizarán principalmente los siguientes conceptos de Java:

- *tipos de datos primitivos, String, Array y tipos referencia* (objetos) definidos por el programador,
- *clases* sencillas definidas por el programador para implementar tipos abstractos de datos mediante *variables de instancia, variables de clase, métodos de instancia, métodos de clase y constructores*.
- *entrada/salida elemental* para lectura de archivos en formato texto y visualización de texto en la consola
- *herencia, sobrescritura de métodos*
- *iniciación a las colecciones*
- *buen estilo de programación y comentarios para documentación automática mediante javadoc.*

Apartado 0. Introducción

En esta práctica vamos a seguir desarrollando la aplicación de venta de electrodomésticos cuyo diseño se realizó en el **apartado 2 de la práctica anterior**. Inicialmente partiremos de la descripción dada en la práctica 2, y en sucesivos apartados de la práctica actual iremos progresivamente añadiendo nuevos requisitos de funcionalidad.

Apartado 1. Creación de objetos y cálculos básicos (3 puntos)

El tester que figura a continuación se encarga de crear unos electrodomésticos y crear con ellos diversas ventas para probar el cálculo del precio final a pagar por el cliente en cada venta. En este apartado debes adaptar tu diseño (diagrama de clases) para que sea compatible con este tester.

Por ejemplo, deberás usar los mismos nombres de clase que se utilizan en el tester, añadir un método `getTicket()` que quizá no tuvieses, etc. También notarás que los frigoríficos y las lavadoras pueden crearse con información opcional que no se mencionó en la práctica anterior: en concreto, los frigoríficos tienen un atributo que indica si son *no frost* y las lavadoras tienen dos atributos para *kilogramos de carga* y *revoluciones por minuto en centrifugado*.

Además de añadir estos atributos a tu diseño, deberás implementar todos los constructores y métodos de todas las clases para que la ejecución del tester con tu código produzca la salida esperada que se muestra abajo. Los nuevos atributos deben ocupar los últimos lugares en la lista de parámetros de los constructores, facilitándose así la creación de unos constructores con parámetros para toda la información incluida la opcional y otros constructores con parámetros sólo para la información obligatoria. En el tester de abajo puedes ver que los constructores de las lavadoras de marca Indesit y Superkin tienen distinto número de parámetros.

Recuerda que los detalles para calcular descuentos, portes y precio final se dieron en el apartado 2 de la práctica anterior. Asegúrate de que tus cálculos producen exactamente la misma salida y en el mismo formato de la salida esperada.

No te olvides de seguir la guía de estilo de programación Java disponible en Moodle, incluyendo comentarios, especialmente los usados para javadoc, en el código de todas las clases que escribas.

Tester apartado 1 (disponible en Moodle):

```
package ads.practica3;
/**
 * Primer tester de la práctica 3
 * @author Profesores ADS
 */
public class TesterTienda1 {
  public static void main(String[] args) {
    Electrodomestico tele1 = new Television("Loewe", "Art 48 4K Plata", 1699.00,
                                            ClaseEnergetica.A, 48);
    Electrodomestico tele2 = new Television("LG", "32LF592U", 271.92, ClaseEnergetica.C, 32);
    Electrodomestico lava1 = new Lavadora("Indesit", "XWE 91483 X", 329.0, ClaseEnergetica.A,
                                          new Dimension(59.5,60.5,85.0), 72, 9.5, 1500);
    Electrodomestico frigo1 = new Frigorifico("Bosch", "KGN39VW21", 599.0,
                                ClaseEnergetica.A, new Dimension(60,65,201), 83.0, true);

    Venta[] ventas = new Venta[6];
    ventas[0] = new Venta(tele1);
    ventas[1] = new VentaDomicilio(tele1);
    ventas[2] = new Venta(tele1,
                          new Lavadora("Superkin", "", 0.0, ClaseEnergetica.Desconocida,
                                  new Dimension(59.5, 60.5, 85.0), 100)); // sin carga ni rpm
    ventas[3] = new VentaDomicilio(lava1, frigo1);
    ventas[4] = new Venta(tele2,
                          new Television("Telefunken", "", 0.0, ClaseEnergetica.D, 32));
    ventas[5] = new VentaDomicilio(lava1, tele2);

    for (Venta venta : ventas)
        System.out.println(venta.getTicket());
    }
}
```

Salida esperada apartado 1:

```
-----------------------------------------
Producto vendido: Loewe Art 48 4K Plata, 1699.00 Euros
-----------------------------------------
Precio producto:         1699.00 Euros
Descuento entrega:          0.00 Euros
TOTAL:                   1699.00 Euros
-----------------------------------------
Producto vendido: Loewe Art 48 4K Plata, 1699.00 Euros
-----------------------------------------
Precio producto:         1699.00 Euros
Descuento entrega:          0.00 Euros
Coste porte:               43.00 Euros
TOTAL:                   1742.00 Euros
-----------------------------------------
Producto vendido: Loewe Art 48 4K Plata, 1699.00 Euros
-----------------------------------------
Precio producto:         1699.00 Euros
Descuento entrega:         10.00 Euros
TOTAL:                   1689.00 Euros
-----------------------------------------
Producto vendido: Indesit XWE 91483 X, 329.00 Euros
-----------------------------------------
Precio producto:          329.00 Euros
Descuento entrega:         25.00 Euros
Coste porte:               46.00 Euros
TOTAL:                    350.00 Euros
-----------------------------------------
Producto vendido: LG 32LF592U, 271.92 Euros
-----------------------------------------
Precio producto:          271.92 Euros
Descuento entrega:         40.00 Euros
TOTAL:                    231.92 Euros
-----------------------------------------
Producto vendido: Indesit XWE 91483 X, 329.00 Euros
-----------------------------------------
Precio producto:          329.00 Euros
Descuento entrega:         55.00 Euros
Coste porte:               46.00 Euros
TOTAL:                    320.00 Euros
```

Apartado 2. Lectura básica de archivo de texto (2 puntos)

En este apartado debes desarrollar una clase `LecturaElectrodomesticos` con un método `leer` que lea línea a línea un archivo de texto que contiene siempre todos los datos de electrodomésticos de prueba (incluidos los opcionales) y los cargue en una estructura de memoria que será devuelta por dicho método. Como se aprecia en el tester siguiente, dicha estructura devuelta por `leer` debe ser compatible con `List<Electrodomestico>`, por ejemplo, `ArrayList<Electrodomestico>`. Recuerda que para usar estas estructuras tu programa deberá incluir una línea **`import`** similar a la que hay al principio del tester.

También se aprecia en el tester que el método `leer()` recibe un parámetro que indica el nombre del archivo de texto a leer, cuyo contenido mostramos a continuación.

Archivo con datos de entrada para tester 2 (disponible en Moodle, archivo `productos.txt`):

```
Loewe=Art 48 4K Plata=1699=A=48
LG=32LF592U=271.92=C=32
Indesit=XWE 91483 X=329=A=59.5=60.5=85=72=9.5=1500
Bosch=KGN39VW21=599=A=60=65=201=83=NoFrost
Loewe=Art 48 4K Plata=1800=A=48
Indesit=XWE 91483 Z=329=A=59.5=60.5=85=72=10.0=1500
Bosch=KGN39VW21=599=A=59.5=60.5=85=72=10.0=1500
```

Cada línea del archivo consta de una serie de campos separados por un signo **=** (que podemos suponer que nunca aparecerá en el contenido de ningún campo). Los campos de cada línea describen todos los atributos de un electrodoméstico (ya sea frigorífico, lavadora o televisión), incluidos los que se añadieron en el apartado anterior.

Después de descomponer en campos cada línea leída, se debe crear un objeto del tipo de electrodoméstico que corresponda para añadirlo a la estructura de datos que se devolverá una vez los contenga todos. Es imprescindible que hagas un método independiente para crear cada tipo de electrodoméstico. Piensa bien en qué clase resulta más conveniente añadir dichos métodos. Al ejecutar el tester dado, su salida debe ser idéntica a la salida esperada que se indica a continuación de él.

Tester apartado 2 (disponible en Moodle)**:**

```java
package ads.practica3;

import java.util.List;

/**
 * Segundo tester el tercer apartado de la P2
 * DEBE EJECUTARSE CON UN ARGUMENTO EN LA LINEA DE COMANDOS
 *
 * @param args[0] El primer y único argumento será el nombre del archivo de texto a leer
 * @author Profesores ADS
 */
public class TesterTienda2 {
    public static void main(String[] args) {
        List<Electrodomestico> productos = LecturaElectrodomesticos.leer( args[0] );
        for (Electrodomestico e : productos) {
                System.out.println(">> " + e );
        }
    }
}
```

Salida esperada apartado 2:

```
>> Loewe Art 48 4K Plata, 1699.00 Euros
>> LG 32LF592U, 271.92 Euros
>> Indesit XWE 91483 X, 329.00 Euros
>> Bosch KGN39VW21, 599.00 Euros
>> Loewe Art 48 4K Plata, 1800.00 Euros
>> Indesit XWE 91483 Z, 329.00 Euros
>> Bosch KGN39VW21, 599.00 Euros
```

Apartado 3. Igualdad de objetos con `equals` y evitar duplicación (2 puntos)

En este apartado se utilizará el mismo tester que en el anterior, pero su salida esperada (mostrada abajo) será diferente a la anterior. En concreto, debes modificar tu clase `LecturaElectrodomesticos` para que al leer los electrodomésticos del archivo de texto se detecten los que sean iguales a otro electrodoméstico previamente procesado y añadido a la lista. En caso de que ya haya en la lista un electrodoméstico igual al que se esté leyendo del archivo se evitará añadir éste a la lista, conservando en ella el que se añadió antes. **Dos electrodomésticos se consideran iguales si y sólo si ambos son del mismo tipo de electrodoméstico y tienen idéntica marca e idéntico modelo**, independientemente de sus otros atributos. Aunque parezca raro que haya dos electrodomésticos de distinto tipo (p.ej., frigorífico y lavadora) con idéntica marca y modelo se debe hacer la comprobación necesaria para considerarlos desiguales por no ser del mismo tipo.

Nótese que en la salida esperada el duplicado no coincide en precio con el otro electrodoméstico de igual tipo, marca y modelo. En cambio, la última línea del archivo se toma como un electrodoméstico diferente por ser de distinto tipo al otro que coincide en marca y modelo.

El mecanismo que tiene Java para comprobar la igualdad de objetos se basa en el método `public boolean equals(Object obj)` de la clase `Object` que heredan todas las clases y que se puede sobrescribir en cualquier clase que desee definir su propia forma de igualdad entre objetos. Para ello se debe implementar en la clase correspondiente un método con exactamente la misma cabecera de `equals` (descrita arriba) pero precedida de `@Override` para indicar que se trata de una sobrescritura.

Archivo con datos de entrada para 3:
Mismo que para apartado 2

Tester apartado 3:
Mismo que tester 2

Salida esperada apartado 3:
```
Duplicado no añadido:
Loewe Art 48 4K Plata, 1800.00 Euros
>> Loewe Art 48 4K Plata, 1699.00 Euros
>> LG 32LF592U, 271.92 Euros
>> Indesit XWE 91483 X, 329.00 Euros
>> Bosch KGN39VW21, 599.00 Euros
>> Indesit XWE 91483 Z, 329.00 Euros
>> Bosch KGN39VW21, 599.00 Euros
```

Apartado 4. Memorización y anulación de ventas dentro de la clase `Venta` (2 puntos)

Como ves en el tester de este apartado, la primera instrucción consiste en volver a ejecutar el tester del apartado 1, y después se ejecutan algunos métodos nuevos que tendrás que desarrollar en este apartado y añadirlos a la clase `Venta`. El objetivo de estos métodos es gestionar globalmente la colección de todas las ventas creadas durante la ejecución del tester (concretamente, en el tester 1).

El primer método nuevo es un método de clase `resumenVentas` que construye y devuelve el string que constituye un resumen de ventas listo para ser impreso (ver código del tester y salida esperada) que contendrá, en orden cronológico, la línea final del ticket generado para cada venta (no anulada). Eso implica que en algún momento todas las ventas habrán tenido que ser almacenadas en el interior de la clase `Venta`. *(Nota: quizá no siempre sea ésta la mejor decisión de diseño pero por el momento te servirá para seguir avanzando.)* Si en el apartado 1, descompusiste el método `getTicket` en cuatro métodos, uno para cada línea componente del ticket, ahora te será muy sencillo reutilizar el método que imprimía la última línea del ticket; de lo contrario tendrás que repetir código (cosa no deseable).

El segundo método a desarrollar es el método de clase `anular` que elimina de la colección de ventas almacenadas la última venta almacenada, y la devuelve como resultado. Si este método se invoca de manera sucesiva puede eliminar, una a una, varias ventas en orden inverso a su creación. Y si no hubiese más ventas que anular, simplemente devolvería **`null`**. También tendrás que añadir otro método de clase que devuelva la última venta almacenada sin eliminarla. El tester utiliza estos métodos para anular dos ventas y posteriormente verificar que ya no aparecen en el resumen de ventas.

Por último, es necesario añadir otras dos formas del método `resumenVentas`; la primera con un parámetro numérico mostrará sólo ventas con *importe final no inferior al parámetro*, y la segunda con un parámetro `String` mostrará sólo ventas de *productos cuya marca contenga el texto dado* como parámetro.

Tester apartado 4 (disponible en Moodle)**:**

```
package ads.practica3;

/**
 * Tester el apartado 4 de la práctica 3
 * @author Profesores ADS
 */
public class TesterTienda4 {
  public static void main(String[] args) {
        // ejecutamos el tester1 para cargar electrodomesticos y ventas
        TesterTienda1.main(null);

        System.out.println( Venta.resumenVentas() );
        System.out.println( Venta.ultima().getTicket() );

        Venta anulada = Venta.anular(); // anulamos la última venta, se asigna a anulada
        System.out.println("Venta anulada:\n" + anulada.getTicket());

        Venta.anular();  // anulamos otra
        System.out.println( Venta.resumenVentas(500) ); // sin las dos ultimas ventas,
                                                        //  ni la de importe < 500
        System.out.println( Venta.resumenVentas("Indes") ); // solo ventas (no anuladas)
                                                            // de marca Indesit
  }
}
```

Salida esperada apartado 4: Primero se obtiene la salida del apartado 1 y a continuación lo siguiente.

```
RESUMEN DE VENTAS
TOTAL:                   1699.00 Euros
TOTAL:                   1742.00 Euros
TOTAL:                   1689.00 Euros
TOTAL:                    350.00 Euros
TOTAL:                    231.92 Euros
TOTAL:                    320.00 Euros

-----------------------------------------
Producto vendido: Indesit XWE 91483 X, 329.00 Euros
-----------------------------------------
Precio producto:          329.00 Euros
Descuento entrega:         55.00 Euros
Coste porte:               46.00 Euros
TOTAL:                    320.00 Euros

Venta anulada:
-----------------------------------------
Producto vendido: Indesit XWE 91483 X, 329.00 Euros
-----------------------------------------
Precio producto:          329.00 Euros
Descuento entrega:         55.00 Euros
Coste porte:               46.00 Euros
TOTAL:                    320.00 Euros

RESUMEN DE VENTAS
TOTAL:                   1699.00 Euros
TOTAL:                   1742.00 Euros
TOTAL:                   1689.00 Euros

RESUMEN DE VENTAS
TOTAL:                    350.00 Euros
```

Apartado 5. Ampliando el diseño original (1 punto)

La prueba de fuego de todo software es ver qué tal aguanta el mantenimiento, incluidos los requisitos adicionales. Por eso, en este apartado tienes que **añadir dos clases nuevas**: una para representar las *ventas a Canarias*, otra para representar las *televisiones curvas*.

Las ventas a Canarias nunca se hacen a domicilio con recogida de electrodoméstico viejo y sus portes se calculan siempre

como un 7% sobre el precio base del producto vendido. Por ejemplo, la venta de una TV Loewe Art 48 4K Plata, con precio base de 1699.00 Euros implica unos portes de 118.93 Euros.

Para las televisiones curvas debemos almacenar, además de las características comunes a todas las televisiones, su peso y dimensiones (como en las lavadoras y frigoríficos), ya que sus portes se calculan sumando a los portes ya descritos para las televisiones un suplemento de 25 Euros por metro cúbico.

Modifica tu código anterior, mejorándolo en todo lo que sea necesario para incorporar dicha funcionalidad nueva, y añade los datos de prueba y un tester nuevo para probar dicha funcionalidad. Todo ello, por supuesto, sin que ninguno de los testers anteriores deje de producir su salida deseada.

Durante ese proceso analiza tú mismo cuántas modificaciones estas realizando sobre tu código que podrían haberse evitado con un diseño mejor (más flexible).

Apartado 6 (opcional). (1 punto)

Amplía y modifica el diseño y codificación de las clases Java desarrolladas hasta el apartado anterior, para satisfacer nuevos requisitos relativos a la gestión de stocks (existencias en almacén) de los electrodomésticos.

En concreto, deben implementarse las siguientes funciones: añadir un nuevo modelo de electrodoméstico (televisión, lavadora o frigorífico) al catálogo de productos en almacén; mantener sus existencias actualizadas (cuando se realicen ventas, anulación de ventas, o se reciban nuevos ejemplares desde el fabricante); descatalogar un producto (eliminándolo del almacén); y mostrar un inventario del stock de electrodomésticos en el almacén, completo o por tipo de electrodoméstico.

Además de desarrollar y documentar las clases añadidas en este apartado deberás crear datos de prueba y tu propio tester para probar toda la funcionalidad adicional que se pide. **Si en este apartado modificas significativamente las clases desarrolladas desde el apartado 1 al 5, de forma que sus respectivos tester no generan la salida esperada, debes guardar y entregar dichas clases por separado antes de modificarlas en este apartado.**

Normas de Entrega:

- Se debe entregar un único fichero ZIP con todo lo solicitado abajo, que deberá llamarse de la siguiente manera: **P3_**GR<numero_grupo>_<nombre_estudiantes>.zip. Por ejemplo Marisa y Pedro, del grupo 2213, entregarían el fichero: **P3_**GR2213_MarisaPedro.zip.
- Se entregará la versión comprimida del directorio de nombre **P3_**GR<numero_grupo>_<nombre_estudiantes> (p.ej., **P3_**GR2213_MarisaPedro) que contenga:
    - un directorio **src** con todo el código Java en su **versión final del apartado 5**, y en caso de haber realizado la parte opcional, **otra versión final del apartado 6**, incluidos los datos de prueba y testers adicionales que hayas desarrollado en los apartados que lo requieren,
    - un directorio **doc** con la documentación generada
    - un directorio **txt** con todos los archivos utilizados y generados en las pruebas
    - un archivo PDF el **diagrama de clases** y una breve justificación de las decisiones que se hayan tomado en el desarrollo de la práctica, los problemas principales que se han abordado y cómo se han resuelto, así como los problemas pendientes de resolver.