MEMORIA PRÁCTICA 3

ANÁLISIS Y DISEÑO DE SOFTWARE

Laura de Paz Carbajo
Rubén García de la Fuente
Grupo 2201
19/03/2020

APARTADO 1

En este primer apartado implementamos la aplicación de venta de venta de electrodomésticos de la práctica anterior aunque tuvimos que hacer algunas modificaciones respecto a nuestro diagrama de clases como por ejemplo crear una super clase común a Figrorifico y Lavadora que contiene los atributos dimensión y peso. Tuvimos que añadir algún atributo más que se nos especificaba en la práctica. También añadimos el método abstracto precioPorte() en la clase Electrodomestico que no habíamos tenido en cuenta en la práctica anterior.

Añadimos el método getTicket() en Venta tal y como se nos indica en el enunciado. Para ello dividimos esta función en porciones según las líneas que imprime, por si en un futuro necesitáramos usarlas. Implementamos todo esto y finalmente pudimos ejecutar el tester1 y ver que el resultado era el esperado.

APARTADO 2

En este apartado creamos una nueva clase abstracta LeerElectrodoméstico en la cual implementamos una función leer() que lee productos de un txt y los devuelve en un ArrayList.

Ejecutamos el tester2 y vimos que funcionaba correctamente.

APARTADO 3

Sobreescribimos la función equals de Electrodomestico y la usamos para mejorar el resultado del apartado 2. Es decir, la usamos para comprobar si al leer electrodomésticos, estamos leyendo alguno por duplicado y, en tal caso, mostrar un mensaje por pantalla.

APARTADO 4

Creamos un nuevo atributo static al que llamamos ventas y que contiene todas las ventas realizadas hasta la fecha. Creamos varios métodos que nos ayudan a manejar este método, incluído resumenVentas(), que recoge la última línea del ticket de Venta, el método ultima() que devuelve la última venta realizada hasta la fecha y anular() que elimina la última de las ventas realizadas.

APARTADO 5

Con el objetivo de comprobar si nuestra aplicación aguanta el mantenimiento, creamos dos nuevas clases: TelevisionCurva y VentaCanarias.

La primera clase hereda claramente de Television, aunque también tiene los atributos dimensión y peso, por lo que podría heredar de ElectrodomesticoDimPeso. Como java no admite herencia múltiple, optamos por introducir estos dos atributos dentro de la clase TelevisionCurva y hacer que herede únicamente de Television ya que de esta clase hereda tanto atributos como métodos. Por último, sobreescribimos su método precioPorte(). Este se calcula sumando al precio del porte de las televisiones 25€ por metro cúbico. Como no se especifica si es por fracción de

metro cúbico o si éstos 25€ de pueden dividir proporcionalmente al volumen, se ha implementado como fracción de metro cúbico.
Por otro lado, la clase VentaCanarias hereda de VentaDomicilio aunque es necesario sobrescribir el método entregaDomicilio y, consecuentemente, el método geTicket() ya que los costes de los portes son distintos y, también, que no es posible entregar un electrodoméstico en una venta de este tipo. Por ello, a diferencia de en Venta y VentaDomicilio, VentaCanarias unicamente tiene un constructor.

APARTADO 6 (opcional)

En este apartado hemos creado una clase Almacen que contiene métodos genéricos para añadir, retirar o descatalogar productos y que además tiene dos atributos. El primero es un HashMap<Electrodomestico, Integer>, que contiene un par clave/valor indicando el electrodoméstico que se encuentra en el almacén y el número de unidades del mismo. La razón de crear un único mapa para Electrodomestico y no uno para cada una de las clases es que cuenta con la ventaja añadida de que no hay que modificar los métodos de añadir o retirar un producto si se crea un nuevo tipo de electrodoméstico. Aunque sí es cierto que a la hora de mostrar el inventario dificulta un poco la implementación del código, pero nos ha parecido la solución más idónea.
Hemos incluido un array de ventas, que indican las ventas que se realizan exclusivamente desde este almacén. Para indicar que una venta se realiza desde el almacén se invoca al método nuevaVenta(Venta v), este método intenta retirar del almacén el producto del cual se intenta hacer la venta, si este producto no está en el almacén anulamos la venta (a través del método anular()) e igualamos el objeto a null (para que no se pueda repetir esa venta) indicándose además con un mensaje.
Explicar por último que el método anularVenta() anula la última venta del almacén y lo elimina del array de ventas.

DIAGRAMA DE CLASES