**MACOWINS**

**Requerimientos:**

Que los usuarios puedan conocer el precio de prendas y sus tipos.

Que se pueda filtrar las ventas por fecha y obtener la sumatoria de ese día.

**Pseudocodigo:**

**%%**

Class Venta:

Int Código

Fecha fecha

List prendaL

Int cantidad

Double precioTotal

addPrenda(Prenda prenda)

prendaL.add(prenda)

calcularPrecioBase()

return prendaL.sum(prenda->prenda.precio)

mostrarProductos()

prendaL.forEach(prenda->mostrarprecioTipo(prenda))

fechaCMP(Fecha buscada)

return buscada==fecha

%%

Class VentaConTarjeta extends Venta:

Int cuotas

Int coeficienteFijo

calcularPrecioConTarjeta ()

return cuotas\*coeficientefijo+ calcularPrecioBase ()\*0.01

%%

Class registro:

list listaVentas

addLista(Venta venta);

listaVentas.add(venta)

sumatoriaDeVentasEnFecha(listaVentas, Fecha fecha);

return listaVentas.filter(venta->venta.fechaCMP(buscada))

.sum(venta->venta.calcular)

%%

Class prenda:

Double precio

Enum tipo

Enum estado

mostrarprecioTipo()

%%

Class nueva extends prenda:

calculoPrecio()

return precio

%%

Class promocion extends prenda:

Int valorDescuento;

calculoPrecio();

return precio-valorDescuento

%%

Class liquidacion extends prenda:

calculoPrecio();

return precio \*0.50

%%

**Diagrama de clase:**

Diagrama

Descripción generada automáticamente

**Prosa:**

El requerimiento de obtener el total de un día en especifico implica que debe existir métodos que logren este resultado. Para esto se agrego a la clase registro que guarda todas las ventas en una lista con la cual estos métodos necesitaran. El primer método es sumatoriaDeVentasEnFecha que calcula el total de una fecha especifica filtrando las ventas de ese día y sumando los preciosTotales. De esta forma se retorna el dinero ganado en ese día.

Una venta esta compuesta por métodos que comparan fecha y mostrarProductos, esto es un requisito que se pidió. EL mostrarProductos implica mostrar los precios y tipos de una lista de prendas de forma organizada

Existirá una clase ventaContarjeta diseñada para los pagos con tarjeta. Esta hereda de venta adicionando 2 parámetros y un método calculoBase. Todo esto se usará para la nueva forma de cálculo de calcularPrecioTotal.

Claramente la clase prenda debe incluir getters que devuelvan el precio y tipo. Se incluyo 3 nuevas clases que heredan de prenda siendo estas nueva, promoción y liquidación, las cuales modifican el método de calcularPrecio para devolver el precio que se calcula de forma diferente a la original.

**ALTERNATIVAS**

Una alternativa es que no existan clases nueva, promoción y liquidación. Dejándolo como un enum para que el calculo se haga en calcular precio mediante un switch o if. Esto se puede molestar al momento de querer controlar el valor de descuento de manera que se tendría que modificar el código para que funcione. En cambio, si es una clase aparte es mas fácil y se puede lograr mediante una interfaz gráfica.

Otra era al comienzo donde pensé en hacer clases para sacos, pantalones y camisas, pero ninguna tiene un comportamiento único. Pero no se sabe a futuro si el usuario solicite nuevas variables y funcionalidades.

Otra alternativa son el uso de interfaces, en vez de heredar de prenda las clases nueva, promoción y liquidación implementan interfaces. En los dos casos funcionan y son validas.

Entiendo que las interfaces dan la posibilidad de poder implementar muchas en una clase en comparación de herencia que solo es una. Como lo que se requiere no es de una complejidad grande donde una clase deba usar varias interfaces, no opte por aplicarlo.