Actividad

“Macowins”

Diseño de Sistemas

Docente:

- Gaston Oscar Prieto

Alumnos:

- Aizcorbe Daniel
- Blanco Lucas
- Candia Thomas
- Maqueda Pablo
- Parente Daniel

Fecha de entrega:

- 7/4/2022

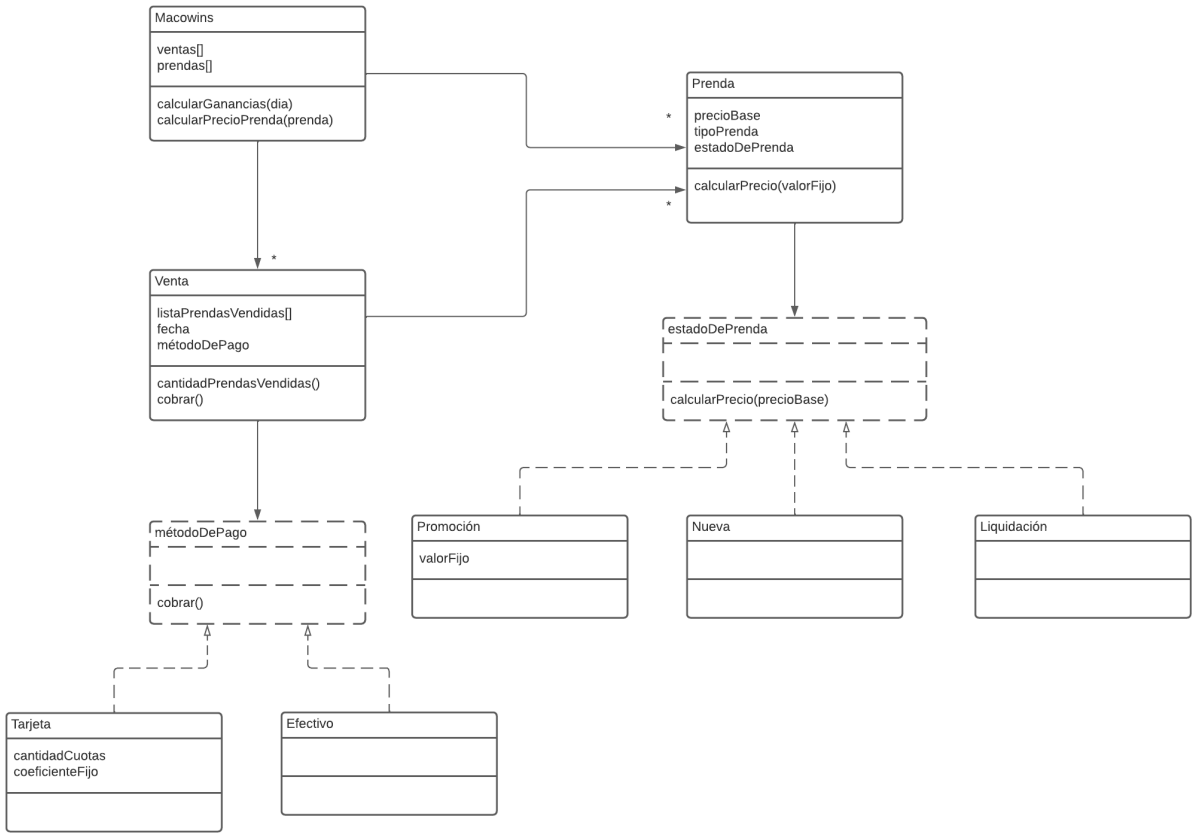
Enunciado:

https://docs.google.com/document/d/1mjWKl9YH9Bb39iIUl1bQj_xhx_-CjCAMpcAXRqKhVjU/edit

Requerimientos:

- Funcionales:
  - Calcular el precio de venta de una prenda.
  - Consultar el tipo de cada prenda.
  - Registrar ventas.
  - Consultar ganancias de un determinado día
- No Funcionales:
  - No se encontraron requerimientos no funcionales.

Nuestra solución es principalmente basada en composición:



ALCANCE DEL SISTEMA

Como el enunciado no habla de “clientes”, asumimos que la parte “cliente realiza orden de compra, validamos inventario, validamos si se puede pagar, hacemos la venta y realizamos actualización de stock” está por fuera de nuestro sistema.

Solución (pseudocódigo):

Para el primer requerimiento solicitado por el cliente "*calcular el precio de venta de una prenda":*
La composición fue parte fundamental de nuestra solución, ya que nos permite tener la flexibilidad de que un objeto pueda cambiar su estado (de promoción a liquidación por ejemplo) o si se diera el caso que un nuevo estado surja, sería fácilmente implementable, solamente habría que crear una nueva clase que entienda los mensajes básicos de un "**Estado de Prenda".**
Por consecuencia, el método encargado de saber el precio de una prenda (en base a su estado) solamente debía preguntarle al estado de la prenda:

```
73    method calcularPrecio() {
74        return estadoPrenda.calcularPrecio(precioBase)
75    }
76
```

y cada estado se encargaría de calcularlo en función de lo que el cliente haya decidido. En este caso:

- *Nueva: en este caso no modifican el precio base.*
- *Promoción: Le resta un valor fijo decidido por el usuario.*
- *Liquidación: Es un 50% del valor del producto.*

En el diagrama de clases, al principio pensamos en una clase abstracta y que los 3 estados mencionados en los requerimientos fuesen subclases, pero por simplicidad, solo dejamos a "Promocion" como subclase, ya que los otros no tenían sentido como clases porque no se necesitaba instanciar más de una vez. En el diagrama los dejamos indicados como subclases porque nos pareció semánticamente más claro así.

El cliente solicitó poder "*consultar el tipo de cada prenda"* para lo cual simplemente añadimos un getter() a un *string constante (tipoDePrenda)*:

```
    method tipoDePrenda() = tipoDePrenda
```

Decidimos que sea un *string* y no *clase* porque no cumplía ningún rol además de informativo.

Aclaración: El tercer requerimiento "*registrar ventas"* nos generó cuestionamientos al tratar de representar el diagrama de clases al tratarse de un objeto el que almacena las ventas *"object macowins"*.
Cada vez que se cobra una venta se añade al objeto *macowins*

```
33      method cobrarse() {
34          macowins.cobrarVenta(self)
35      }
```

Para el cuarto y último requerimiento el cliente nos solicitó “*consultar las ganancias de un determinado día”* para lo cual es necesario el requerimiento anterior (que hayan ventas registradas)

```
6       method determinarGanancias(dia) {
7           // (control del día)
8           const ventasDelDia = filtrado por dia seleccionado
9           return suma de ventas del dia filtrado
10      }
```