##### **Concepto de evaluación:**

*Leé completo y con cuidado el enunciado. Pensá bien la estrategia de resolución antes de comenzar el desarrollo de lo que te solicitan. El objetivo de este examen es evaluar la correcta aplicación de los conceptos y técnicas vistos hasta el momento:*

* *Correcta implementación de constructores.*
* *Modularización reutilizable y mantenible con uso de métodos con correcta parametrización y correcto encapsulamiento, publicando setters y getters sólo cuando corresponda.*
* *Manejo de clases, colecciones, y arreglos uni y bidimensionales.*
* *Posicionamiento directo e indirecto.*
* *Excepciones*
* *Herencia, Polimorfismo, Interfaces*
* *Pilas, Colas y Listas Ordenadas*
* *Importación y Exportación de proyectos Java desde Eclipse.*

### **Contexto:**

Un cliente nos pide realizar un simulador de un supermercado.

En él se debe poder procesar sus compras, registrar a los clientes y emitir un listado de las compras.

Primero los clientes se deben registrar para poder ingresar al supermercado.

Se le pedirá al cliente nombre, edad e ingresar dinero a su cuenta para poder comprar.

Además pueden haber 3 tipos de clientes:

* *Cobre***:** tienen el 10% de descuento en las compras a partir de $200
* *Plata***:** tienen el 20% de descuento en las compras a partir de $300
* *Oro***:** tienen el 50% de descuento en todas las compras.

El proceso de compra es el siguiente:

1. Cada vez que se agrega un producto al carrito del cliente este último se pone arriba del anterior.
2. El cliente debe dirigirse a la fila y esperar a ser atendido.
3. Una vez llegado a la caja se debe procesar todos los productos del carrito para saber el precio de cada producto.
4. Se verifica si al cliente se le aplica el descuento.
5. Luego se verifica el saldo del cliente para saber si está en condiciones de realizar la compra, en caso negativo se rechaza.

De cada producto se sabe el precio y su nombre.

De cada compra se sabe el ID, precio (con descuento), si se aplicó descuento o no y que cliente realizó la compra.

Una vez terminado con todas las compras nos piden que al final del día se emita un listado.

### 

### **Introducción técnica:**

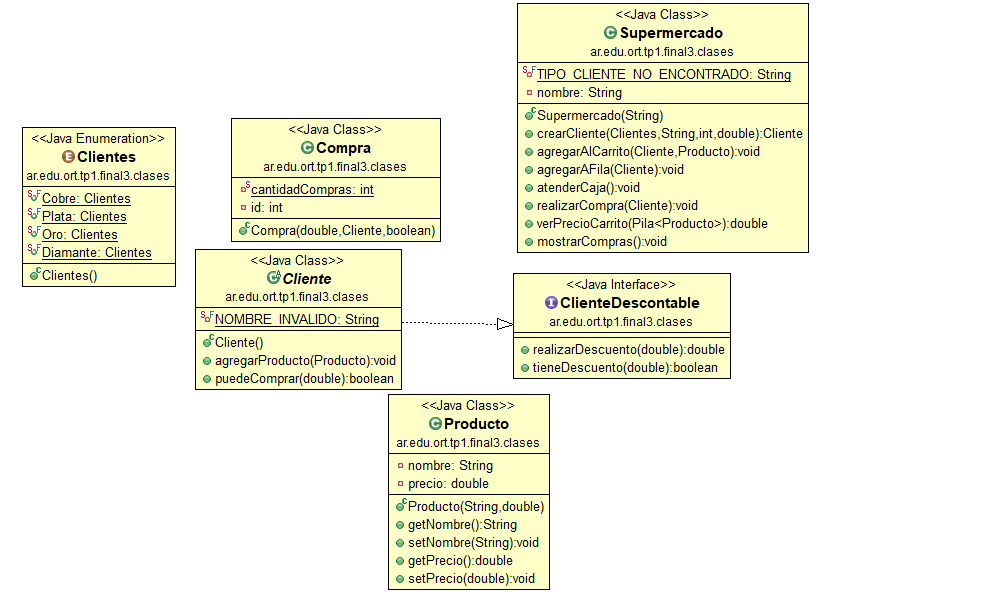
- Un cliente no puede ser menor de edad, tener saldo negativo ni tener nombre null.

- Un cliente no puede comprar más de 2 productos iguales, al momento de querer agregarlo al carrito no se le permitirá.

- El listado que se emite al final debe estar ordenado por precio de compra de menor a mayor.

-El tipo de cliente diamante no está disponible en esta versión

**El diagrama de clase (incompleto) es el siguiente:**

****

La clase Final (Main) no debe ser editada.

***IMPORTANTE!!!***

Cada vez que se pida completar un método no se puede modificar la firma del mismo

La salida debe ser la siguiente

Final 3 TP1

El cliente no puede ser menor de edad

El saldo no puede ser negativo

El nombre no puede ser nulo ni vacio

El tipo de cliente que ha ingresado no se encuentra disponible

La cantidad limite de este producto fue alcanzada

El saldo no es suficiente para realizar esta compra

-------------------------

Compra [id=4, precio=20.0, cliente=Kevin, descuento=false]

Compra [id=2, precio=25.0, cliente=Shaggy, descuento=true]

Compra [id=3, precio=30.0, cliente=Brian, descuento=false]

Compra [id=0, precio=50.0, cliente=Donal, descuento=false]

Compra [id=5, precio=250.0, cliente=Patricio, descuento=true]

Compra [id=1, precio=404.0, cliente=Scooby, descuento=true]

### **Problemas a resolver:**

* 1. Completar la clase Cliente con sus respectivos atributos y métodos y con cada una de sus validaciones, en caso de que no se cumpla con los requisitos se debe arrojar un ERROR personalizado para cada uno de ellos.
  2. Se debe resolver la variación de los tipos de clientes de una manera escalable y flexible en caso de que en un futuro se quiera agregar o eliminar otro tipo.
  3. Completar el método crearCliente de la clase Supermercado.

\* Donde el parámetro “tipo” corresponde al tipo de cliente que se quiere crear (cobre, plata u oro).

\* Si el tipo de cliente no se encuentra disponible arrojar un ERROR.

\* Debe poder manejar ERRORES.

\* En caso de que no se haya podido crear al cliente devolver null.

1. 1. Completar el método agregarProducto de la clase Cliente.

\* donde se debe agregar un producto al carrito.

\* en caso de que no se cumpla con el requisito se debe arrojar un ERROR personalizado.

* 1. Completar el método agregarAlCarrito de la clase Supermercado.

\* Donde se debe agregar un producto al carrito del cliente.

\* Si el Cliente es null no se agrega el producto al carrito

\* Debe poder manejar ERRORES.

* 1. Completar el método agregarAFila de la clase Supermercado.

\* Se debe agregar un cliente a la fila del supermercado.

\* Si el Cliente es null no se agrega el cliente a la fila

* 1. Completar el método verPrecioCarrito de la clase Supermercado.

\* Donde se le pasa un carrito y devuelve el precio total (La suma de cada uno de sus productos).

\* Los productos deben quedar en el mismo orden que fueron ingresados al carrito.

* 1. Completar el método puedeComprar de la clase Cliente.

\*Donde se le pasa un precio e indica si este puede o no pagarlo.

* 1. Completar el método realizarCompra de la clase Supermercado.

\* Donde se le pasa un cliente y genera una compra con los productos de su carrito previamente cargado.

\*Se debe verificar si se puede aplicar el descuento correspondiente.

\* En caso de que el dinero del cliente no alcance se debe cancelar la compra y arrojar un ERROR

\* De no haber inconvenientes el proceso finaliza con la compra de dicho cliente agregandola en la estructura correspondiente

* 1. Completar el método atenderCaja de la clase Supermercado.

\* Donde cada cliente realice su compra.

\* Debe poder manejar ERRORES en caso necesario.

* 1. Completar el método mostrarTicket de la clase Compra.

\* donde se muestra su id, precio, nombre de cliente, y si tuvo descuento

* 1. Completar el método mostrarCompras de la clase Supermercado.

\* Donde se emite el listado de todas las compras realizadas, de menor a mayor precio.