# Técnicas de Diseño
75.10

**Trabajo Práctico N°2**

1er cuatrimestre 2013

Corrige: Carlos Curotto

**Grupo N° 15:**

**Albertario, Hernán (87812 - hernandres64@hotmail.com)**

**Hemmingsen, Lucas (76187 - lucas@lhca.com.ar)**

**Jalil Maluf, Yamil (79040 – yamiljm@gmail.com)**

**75.10 Técnicas de diseño**

**Trabajo práctico grupal N° 2**
**Hyper Market - Descripción del Negocio**

El grupo Hyper Market es una cadena de supermercados que vende productos de todo tipo y cuenta con sucursales en todo el país. El negocio de esta cadena es la venta minorista de estos productos al público en general.
El mercado actual se ha tornado muy competitivo por lo que las cadenas de supermercados necesitan sacar ofertas y promociones de todo tipo y color constantemente.
La cadena cuenta con una administración central, donde los responsables comerciales y de marketing son una verdadera usina generadora de ofertas y promociones de todo tipo y color (Ej.: todos los productos de farmacia con 10% de descuento, o si llevas 1 coca te llevas otra gratis, si es Lunes y pagas con tarjeta de debito del banco Nacion tenes un 5%, si compras productos de la vinoteca la segunda unidad tiene un 75% de descuento, pero quedan excluidas las marcas Chandon y Bosca, etc.). Un producto no puede aplicar a mas de una promo.
La cadena ha decidido el desarrollo de un nuevo sistema que le permita gestionar dinámicamente las ofertas que deben aplicar en las distintas sucursales, de acuerdo a las decisiones tomadas por los responsables comerciales y de marketing desde la administración central. De esta forma, las sucursales aplicarían las ofertas y promociones en tiempo real sobre las ventas que realicen, de acuerdo a lo que se define en la administración central. Al momento en que el comercial activa una oferta para una sucursal, esta sucursal debe comenzar a aplicarla lo más pronto posible.
Los cajeros de las sucursales operan independientemente de los sistemas y repositorios de la administración central, ya que los cajeros no acceden al sistema central para efectuar las ventas. Existe no obstante, una conexión permanente que sirve para intercambiar información entre la central y las sucursales.

***Restricciones***
1) Trabajo Práctico **Grupal** implementado en java o C#
2) Debe entregarse las hojas del TP, bien abrochadas (sin carpetas, ni folios), con enunciado del tp y carátula con los datos del alumno (padrón, nombre, email).
3) Se debe enviar en un mail con título TP2-GRUPO-N el TP realizado en un zip/war de nombre TP2_GRUPON_zip, incluyendo código fuente, instructivos y documentación, a la dirección de mail: **tp2@tecnicasdedisenio.com.ar** con los datos del grupo en el cuerpo del email (nro de grupo, integrantes: padrón, nombre, email).

***Criterios de Corrección***
_ Documentación Entregada
_ Diseño del modelo
_ Diseño del código
_ Test Unitarios

Se tendrán en cuenta también la completitud del tp, la correctitud, distribución de responsabilidades, aplicación y uso de criterios y principios de buen diseño.

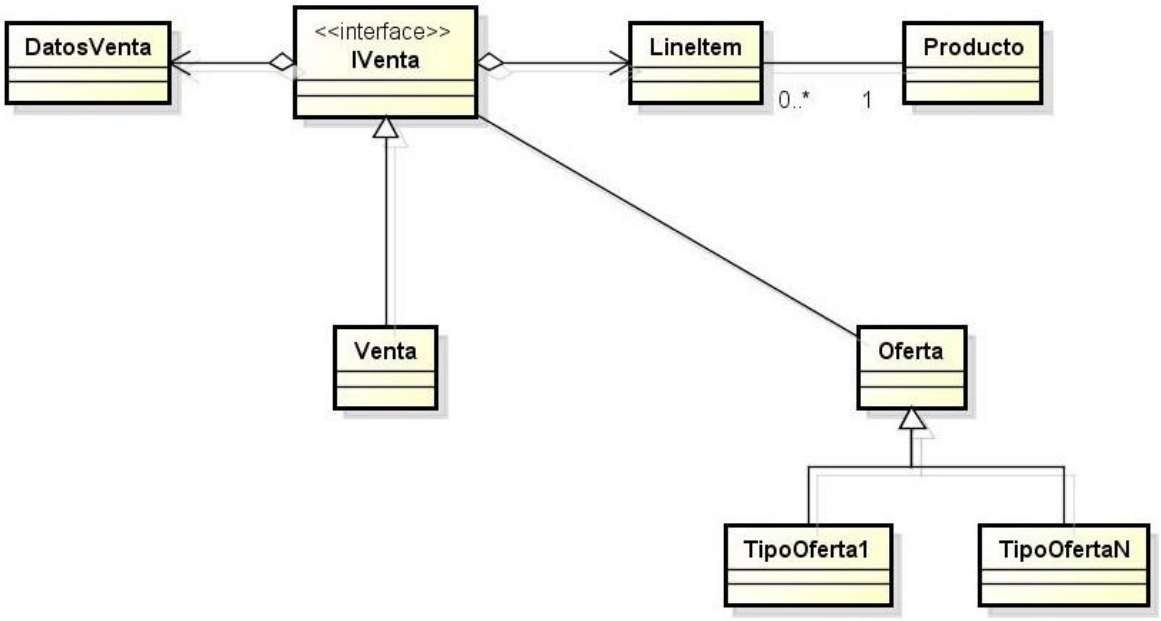
**Primera Entrega**
_ Se debe modelar la creación y administración de ofertas, con la mayor flexibilidad posible para las mismas
_ Se debe modelar la caja con las funcionalidades de abrir caja, iniciar compra, agregar productos, visualizar total y descuentos aplicados, indicar medio de pago, confirmar compra, cerrar caja. Visualizar total ventas de la caja, total descuentos, total monto en caja para cada medio de pago.
_ Se debe tener un buen set de pruebas unitarias sobre algunas ofertas a definir por el cliente y sobre la caja.
_ Se deberá armar una aplicación de **consola de texto, o UI simplificada** que provea toda la funcionalidad de la caja, la cual permite abrir la caja, iniciar una venta, agregar los productos, visualizar el total y los descuentos aplicados, confirmar compra y cerrar la caja.
_ No hay restricciones en el acceso a datos, el mismo puede ser totalmente en memoria o utilizando algún medio persistente. En dicho caso la única restricción es que no se podrá usar base de datos relacional. XML o Base de datos orientada a objetos son alternativas válidas.

***Calendario***
**Jue 09/05 Presentación del TP**
**Jue 30/05 Primera Entrega**

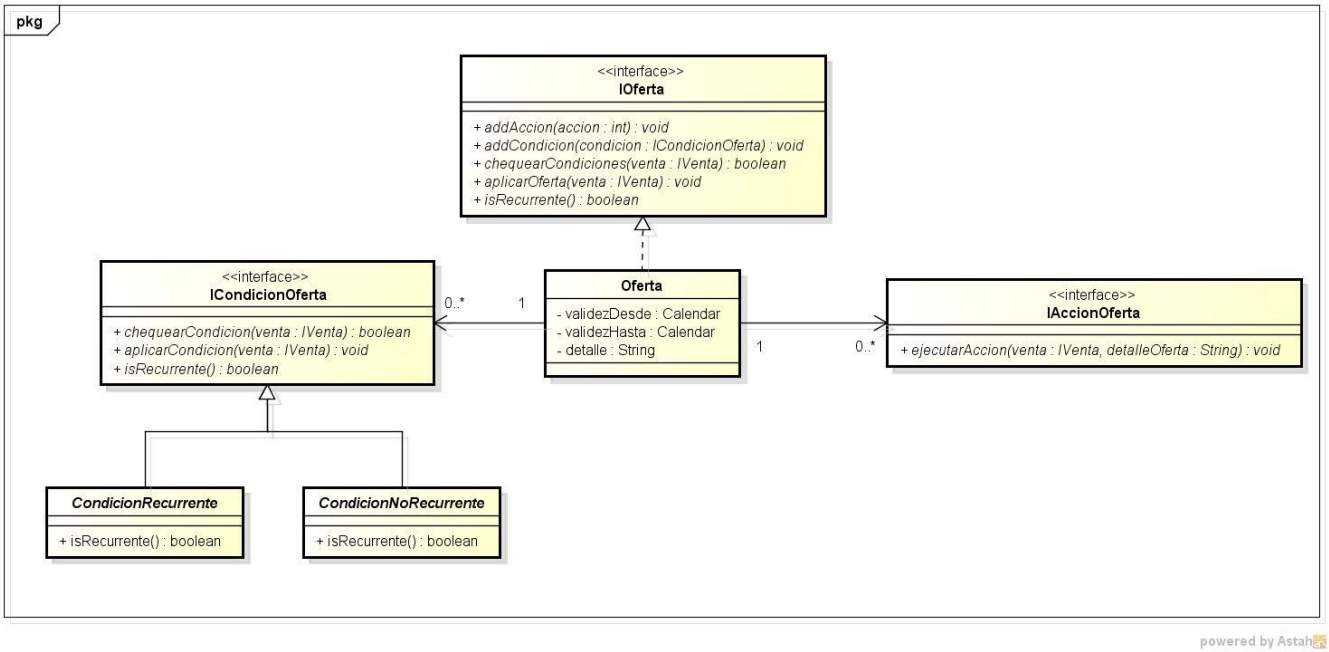
Los TPs que no realicen la entrega en fecha, quedan desaprobados.
Durante las clases intermedias se realizaran consultas del grupo al ayudante a fin de validar el avance del mismo. Durante el mismo se puede preguntar a los integrantes temas puntuales de los realizados para ir siendo evaluados.

## Resolución del Trabajo Práctico.

Inicialmente, luego de haber leído las especificaciones del programa que debía realizarse hicimos un análisis de los requerimientos y utilizando los patrones de análisis armamos un primer bosquejo de lo que se necesitaba. El diagrama fue el siguiente:

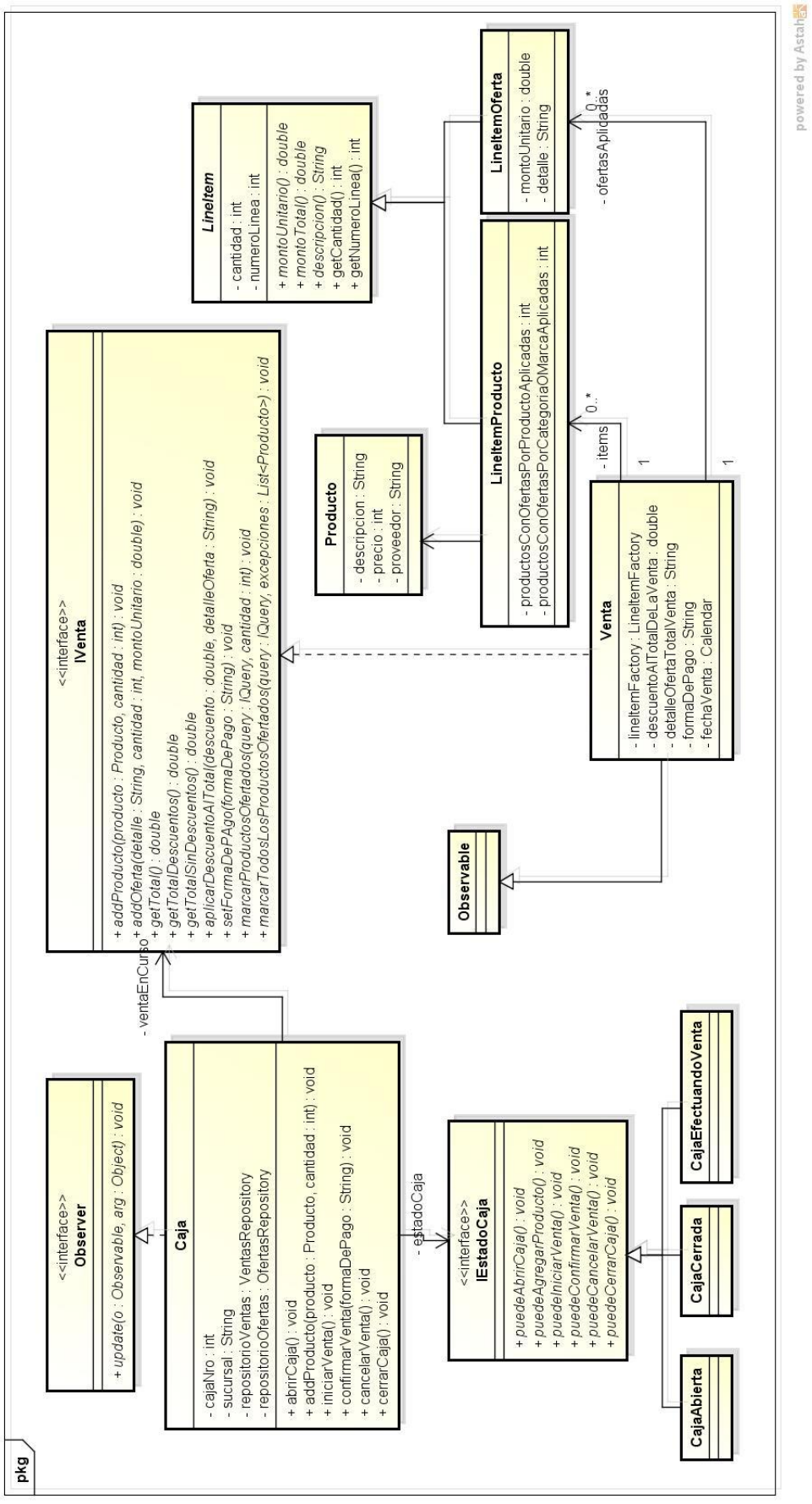



Una vez realizado esto, comenzamos a ver cómo implementar las ofertas para que estas fueran lo más genéricas posible. Luego de analizar algunas opciones, optamos por tratar a las ofertas como una serie de condiciones que debían cumplirse para que la oferta se aplique y una serie de acciones que se producían al aplicarla.




Las condiciones las separamos en dos grupos, las recurrentes y las no recurrentes. Con condiciones recurrentes nos referimos a aquellas en que si la oferta es aplicable sobre la venta, una vez marcados los productos sobre los que se aplicó la oferta debe ser chequeada

nuevamente para verificar si hay otro grupo de productos sobre los cuales aplicar esa oferta.
Una oferta se transforma en recurrente teniendo por lo menos una condición recurrente. Una oferta es no recurrente si está formada solo por condiciones no recurrentes. Ejemplo de condición recurrente es la clase CondiciónLlevaNProducto. Ejemplos de condiciones no recurrentes son CondiciónCategoria o CondicionDiasSemana.
Con respecto a las acciones, estas se aplican sobre la venta. Algunas acciones se aplican sobre productos específicos de la venta, otras se aplican sobre una categoría o una marca de producto (sobre un producto se puede aplicar solo una de estas dos) y acciones que se aplican sobre la venta completa.
Creamos varios repositorios para abstraernos de toda la parte de acceso a datos. Creamos un repositorio de ventas, otro de ofertas y otro de productos. Estos son accedidos por la caja para realizar todas las operaciones.
La clase venta está compuesta por line ítems, estos pueden ser de dos tipos, line ítems de productos, que modelan a uno o varios productos de un mismo producto siendo computados por la caja, o line ítems de oferta, que son generados cuando una oferta cumple las condiciones para ser aplicada sobre la venta.
La clase caja posee todas las operaciones pertinentes a una caja de supermercado y es la interface con la cual el cliente de la aplicación interactuaría. En el caso de utilizarse un patrón de arquitectura Model-View-Controller (MVC) la vista y el controlador interactuarían con la clase caja. Hubiese sido correcto haber segregado una interfaz de Caja para que el cliente de la aplicación programe contra la interfaz de Caja en vez de sobre la clase misma.
El proceso de chequeo de las ofertas sobre la venta es el siguiente: al agregar un nuevo producto desde la caja a la venta se recorre todo el repositorio de ofertas para comprobar si hay alguna oferta que cumpla las condiciones requeridas. Para cada oferta para la cual se cumplan las condiciones, se le aplica la oferta a la venta realizando las acciones correspondientes y se marcan los productos que corresponden a esa oferta, para que no pueda esa misma oferta u otra aplicarle otra oferta del mismo tipo (esto es para el caso de ofertas de producto u ofertas por marca o categoría). En el caso de ofertas que en sus acciones aplican un descuento sobre el total de la compra se guarda el descuento dentro de la clase Venta solo si es un descuento mayor al guardado anteriormente. Este tipo de descuento se aplica recién al final de la venta al efectuar el cobro.


pkg
<<interface>>
Observer
+ update(o : Observable, arg : Object) : void
Caja
- cajaNro : int
- sucursal : String
- repositorioVentas : VentasRepository
- repositorioOfertas : OfertasRepository
+ abrirCaja() : void
+ addProducto(producto : Producto, cantidad : int) : void
+ iniciarVenta() : void
+ confirmarVenta(formaDePago : String) : void
+ cancelarVenta() : void
+ cerrarCaja() : void
- ventaEnCurso
- estadoCaja
<<interface>>
IEstadoCaja
+ puedeAbrirCaja() : void
+ puedeAgregarProducto() : void
+ puedeIniciarVenta() : void
+ puedeConfirmarVenta() : void
+ puedeCancelarVenta() : void
+ puedeCerrarCaja() : void
CajaAbierta
CajaCerrada
CajaEfectuandoVenta
<<interface>>
IVenta
+ addProducto(producto : Producto, cantidad : int) : void
+ addOferta(detalle : String, cantidad : int, montoUnitario : double) : void
+ getTotal() : double
+ getTotalDescuentos() : double
+ getTotalSinDescuentos() : double
+ aplicarDescuentoAlTotal(descuento : double, detalleOferta : String) : void
+ setFormaDePAgo(formaDePago : String) : void
+ marcarProductosOfertados(query : IQuery, cantidad : int) : void
+ marcarTodosLosProductosOfertados(query : IQuery, excepciones : List<Producto>) : void
Observable
Venta
- lineItemFactory : LineItemFactory
- descuentoAlTotalDeLaVenta : double
- detalleOfertaTotalVenta : String
- formaDePago : String
- fechaVenta : Calendar
- items
0..*
1
LineItem
- cantidad : int
- numeroLinea : int
+ montoUnitario() : double
+ montoTotal() : double
+ descripcion() : String
+ getCantidad() : int
+ getNumeroLinea() : int
Producto
- descripcion : String
- precio : int
- proveedor : String
LineItemProducto
- productosConOfertasPorProductoAplicadas : int
- productosConOfertasPorCategoriaOMarcaAplicadas : int
LineItemOferta
- montoUnitario : double
- detalle : String
- ofertasAplicadas
powered by Astah

**Patrones utilizados**

Observer: Este patrón se utilizó entre la caja (observer) y la venta en curso (observable). Cuando a la venta se agrega un line ítem de oferta o producto, esta se encarga de avisarle a la caja que tiene un nuevo line ítem (para que la caja pueda imprimirlo).

Factory Method: Se utilizó en dos ocasiones. La primera como un Factory de LineItems, que se encargaba de crear tanto LineItems de producto como de ofertas y le asignaba el número de línea correspondiente a la venta. También implementamos un Factory para las condiciones de las ofertas y otro para las acciones de las ofertas.

Builder: Utilizamos este patrón para crear un constructor de ofertas. A una oferta mediante el builder se le va agregando condiciones y acciones y luego se obtiene la oferta final.

Command: utilizamos el patrón command para implementar las acciones de las ofertas.

State: se utilizó para implementar el estado de la caja y de esta manera poder saber que operaciones eran válidas para cada estado de la caja.

Strategy: Utilizamos una variación de este patrón (en vez de aplicarlo sobre una clase lo aplicamos sobre un método) en los métodos marcarProductosOfertados y marcarTodosLosProductosOfertados de la clase Venta para poder marcar los productos con ofertas aplicadas de algún tipo de la venta en curso mediante diferentes estrategias (las estrategias están dadas por las clases que implementan IQuery) y de esta manera poder marcar productos por marca, categoría o producto con un solo método.

Problemas encontrados durante el desarrollo

En nuestro diseño inicial las condiciones y acciones de las ofertas cuando necesitaban obtener algún tipo de información de la venta en curso se lo pedían a la clase venta ya que consideramos que Venta tenía que ser la encargada de proveerle la información necesaria a las ofertas.

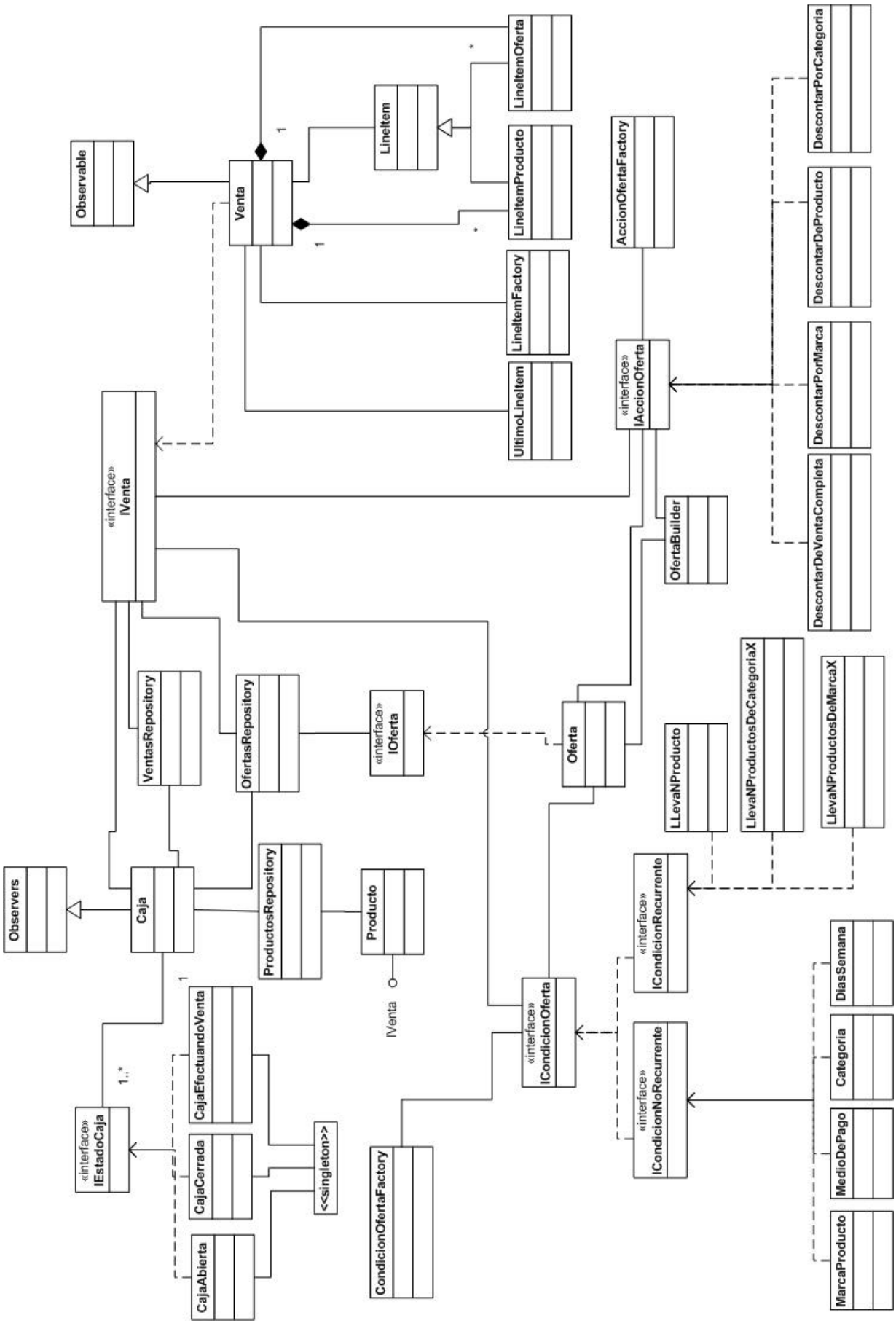
El mayor problema que encontramos con el diseño inicial que nos habíamos planteado fue que notamos que ante algún nuevo tipo de condición o acción de una oferta que creábamos teníamos que crear algún método nuevo en la clase Venta para poder obtener la información requerida. Llegó un momento en que consideramos que debíamos realizar algún cambio ya que por un lado no nos parecía correcto que ante una clase de condición o acción de oferta nueva tuviésemos que crear algún método nuevo en la clase Venta y además esta clase se estaba extendiendo demasiado.

Fue entonces que decidimos que Venta le pase a las ofertas una lista no modificable de los productos para que directamente las condiciones de la oferta puedan chequear ellas mismas si se cumplían o no.

Aún nos quedaba resolver como marcar los productos comprados ya ofertados por categoría, producto o marca sin tener en Venta un método para cada uno. Ahí fue cuando se nos ocurrió implementar el patrón Strategy anteriormente expuesto para de esta manera tener en Venta solo dos métodos para marcar los productos ya ofertados.

De esta manera pudimos clausurar Venta ante la creación de nuevos tipos de condiciones y acciones de oferta.

## Diagrama de clases del modelo utilizado



## Test unitarios

A continuación se muestran algunos de los test unitarios implementados

Test de Producto

```
@Test
	public void cantidadDeProductosNuncaMenorACero() {
		ProductosRepository productoRepo = new ProductosRepository();
		productoRepo.removeProductoById(productoMock.getId());
		assertEquals(productoRepo.getProductos().size(),0);
	}

	@Test
	public void disminuirCantidadDeProductosEnUnoAlQuitarProducto() {

		ProductosRepository productoRepo = new ProductosRepository();
		productoRepo.addProducto(productoMock);
		productoRepo.removeProductoById(productoMock.getId());
		assertEquals(productoRepo.getProductos().size(),0);
	}

	@Test
	public void obtenerProductoPorDetalle()
	{
		ProductosRepository productoRepo = new ProductosRepository();
		productoRepo.addProducto(productoMock);

	assertEquals(productoMock,productoRepo.getProductoByDetalle(productoMock.getDescrip
cion()));
	}

	@Test
	public void obtenerProductoPorId()
	{
		ProductosRepository productoRepo = new ProductosRepository();
		productoRepo.addProducto(productoMock);

	assertEquals(productoMock,productoRepo.getProductoById(productoMock.getId()));
	}
```

## Test VentasRepository

```
@Test
	public void obtenerMediosDePagoTodosDistintos(){

		VentasRepository ventasRepositorio = new VentasRepository();
		ventasRepositorio.add(venta1Mock);
		ventasRepositorio.add(venta2Mock);

		ArrayList<String> mediosDePagoCalculados =
		ventasRepositorio.getMediosDePago();

		ArrayList<String> mediosDePagosEsperados = new ArrayList<String>();
		mediosDePagosEsperados.add("Visa - Banco Itau");
		mediosDePagosEsperados.add("Visa - Banco Santander");

		assertEquals(mediosDePagosEsperados, mediosDePagoCalculados);


	}

	@Test
	public void obtenerMediosDePagoHabiendoRepetidos(){

		VentasRepository ventasRepositorio = new VentasRepository();
		ventasRepositorio.add(venta1Mock);
		ventasRepositorio.add(venta2Mock);
		ventasRepositorio.add(venta3Mock);

		ArrayList<String> mediosDePagoCalculados =
		ventasRepositorio.getMediosDePago();

		ArrayList<String> mediosDePagosEsperados = new ArrayList<String>();
		mediosDePagosEsperados.add("Visa - Banco Itau");
		mediosDePagosEsperados.add("Visa - Banco Santander");

		assertEquals(mediosDePagosEsperados, mediosDePagoCalculados);
	}
```

## Test Venta

```
@Test
	public void marcarProductosComoOfertadosConExcepciones(){

		Venta venta = new Venta(Calendar.getInstance(), 1);
		int cantidadaUnidades = 10;
		IQuery query = mock(IQuery.class);

		when(query.query((Producto) any())).thenReturn(true);
		Producto producto1 = mock(Producto.class);
		Producto producto2 = mock(Producto.class);
		Producto producto3 = mock(Producto.class);
		when(producto1.getId()).thenReturn(1);
		when(producto2.getId()).thenReturn(2);
		when(producto3.getId()).thenReturn(3);
		venta.addProducto(producto1, cantidadaUnidades);
		venta.addProducto(producto2, cantidadaUnidades);
		venta.addProducto(producto3, cantidadaUnidades);
		ArrayList<Producto> productosQueNoAplican = new ArrayList<Producto>();
		productosQueNoAplican.add(producto3);

		venta.marcarTodosLosProductosOfertados(query, productosQueNoAplican );

		List<LineItemProducto> lineItems = venta.getProductos();
```

```
		int cantidadProductosConOferta =
lineItems.get(0).getProductosConOfertasPorCategoriaOMarcaAplicadas();
		cantidadProductosConOferta +=
lineItems.get(1).getProductosConOfertasPorCategoriaOMarcaAplicadas();

		assertEquals(cantidadProductosConOferta, cantidadaUnidades * 2);


	}
```

## Test de Caja

```
	@Test
	public void noSePuedeAgregarUnProductoEnUnaCajaCerrada() {
		boolean lanzoExcepcion = false;
		Producto p = new Producto("Telefono", 12, 12, "Electronica", "Nokia");
		try {
			caja.addProducto(p, 2);
		}
		catch (OperacionCajaInvalidaException e) {
			lanzoExcepcion = true;
		}
		assertTrue(lanzoExcepcion);
	}

	@Test
	public void noSePuedeCancelarUnaVentaEnUnaCajaCerrada() {
		boolean lanzoExcepcion = false;
		try {
			caja.cancelarVenta();
		}
		catch (OperacionCajaInvalidaException e) {
			lanzoExcepcion = true;
		}
		assertTrue(lanzoExcepcion);
	}

	@Test
	public void sePuedeAbrirUnaCajaEnUnaCajaCerrada() {
		boolean lanzoExcepcion = false;
		try {
			caja.abrirCaja();
		}
		catch (OperacionCajaInvalidaException e) {
			lanzoExcepcion = true;
		}
		assertFalse(lanzoExcepcion);
	}

	@Test
	public void noSePuedeConfirmarUnaVentaEnUnaCajaAbierta() {
		boolean lanzoExcepcion = false;
		caja.abrirCaja();
		try {
			caja.confirmarVenta("Efectivo");
		}
		catch (OperacionCajaInvalidaException e) {
			lanzoExcepcion = true;
		}
		assertTrue(lanzoExcepcion);
	}
```

```
@Test
public void sePuedeCerrarUnaCajaAbierta() {
	boolean lanzoExcepcion = false;
	caja.abrirCaja();
	try {
		caja.cerrarCaja();
	}
	catch (OperacionCajaInvalidaException e) {
		lanzoExcepcion = true;
	}
	assertFalse(lanzoExcepcion);
}

@Test
public void sePuedeIniciarUnaCompraEnUnaCajaAbierta() {
	boolean lanzoExcepcion = false;
	caja.abrirCaja();
	try {
		caja.iniciarVenta();
	}
	catch (OperacionCajaInvalidaException e) {
		lanzoExcepcion = true;
	}
	assertFalse(lanzoExcepcion);
}
```

## Test de Venta y Oferta (Integración)

```
private Calendar ofertaDesde;
	private Calendar ofertaHasta;
	private AccionOfertaFactory accionOfertaFactory;
	private CondicionOfertaFactory condicionOfertaFactory;
	private OfertaBuilder ofertaBuilder;
	private OfertasRepository ofertasRepositorio;

	@Before
	public void doBefore()
	{
		ofertaDesde = Calendar.getInstance();
		ofertaDesde.add(Calendar.MONTH, -1);
		ofertaHasta = Calendar.getInstance();
		ofertaHasta.add(Calendar.MONTH, 1);

		accionOfertaFactory = new AccionOfertaFactory();
		condicionOfertaFactory = new CondicionOfertaFactory();
		ofertaBuilder = new OfertaBuilder();
		ofertasRepositorio = new OfertasRepository();
	}

	@Test
	public void aplicarDescuentoPorCategoria() {
		String categoria = "categoriaTest";
		double descuento = 0.1;
		double precio = 10;

		Producto productoMock = mock(Producto.class);
		when(productoMock.getId()).thenReturn(1);
		when(productoMock.getPrecio()).thenReturn(precio);
		when(productoMock.getCategoria()).thenReturn(categoria);
		when(productoMock.getDescripcion()).thenReturn("Coca Cola");

		ofertaBuilder.crearNuevaOferta(ofertaDesde, ofertaHasta,
		categoria);

		ofertaBuilder.agregarCondicion(condicionOfertaFactory.Condicion
		Categoria(categoria));

		ofertaBuilder.agregarAccion(accionOfertaFactory.AccionDescontar
		DeVentaCompleta(descuento));

		ofertasRepositorio.addOferta(ofertaBuilder.getOferta());

		Venta venta = new Venta(Calendar.getInstance(), 1);
		venta.addProducto(productoMock, 1);

		ofertasRepositorio.aplicarOfertas(venta);

		venta.cobrar();

		assertEquals(productoMock.getPrecio() * 0.9,
		venta.getTotal(),1);
	}

	@Test
	//Testea una venta con descuento levando n prods iguales, pagas M < N
	public void ventaConOfertaLlevaNpagaM()
	{
```

```
        double precio = 10;

        //Creo un producto con precio $10
        Producto productoMock = mock(Producto.class);
        when(productoMock.getId()).thenReturn(1);
        when(productoMock.getPrecio()).thenReturn(precio);
        when(productoMock.getMarca()).thenReturn("Coca Cola Company");
        when(productoMock.getDescripcion()).thenReturn("Coca Cola");

        //Creo una nueva oferta
        ofertaBuilder.crearNuevaOferta(ofertaDesde, ofertaHasta, "2x1
        Coca Cola");
        //Le agrego la condicion de que si lleva dos de un producto.
        ofertaBuilder.agregarCondicion(condicionOfertaFactory.Condicion
        LlevaNProducto(productoMock, 2));
        //Le descuenta una unidad.
        ofertaBuilder.agregarAccion(accionOfertaFactory.AccionDescontar
        DeProducto(productoMock, 1, 1));

        //Cargo la oferta al repositorio de ofertas.
        ofertasRepositorio.addOferta(ofertaBuilder.getOferta());

        Venta venta = new Venta(Calendar.getInstance(), 1);

        //le cargo 2 unidades del producto de la promo 2X1
        venta.addProducto(productoMock, 2);

        ofertasRepositorio.aplicarOfertas(venta);

        venta.cobrar();

        //Dado que hay una promocion 2X1 para un producto, cuando lleva
        ese prodcuto, entonces paga solo 1 de los 2.
        //El precio del produco es $10, lleva 2, esta en promo. Paga 10
        assertEquals(venta.getTotal(), precio,1);
}

@Test
public void ventaConOfertaLlevandoNyMpagaMenosElZ()
{
        double precioCoca = 10;
        double precioSprite = 20;
        double precioFanta = 30;

        Producto productoCocaMock = mock(Producto.class);
        when(productoCocaMock.getId()).thenReturn(1);
        when(productoCocaMock.getPrecio()).thenReturn(precioCoca);
        when(productoCocaMock.getDescripcion()).thenReturn("Coca
        Cola");

        Producto productoSpriteMock = mock(Producto.class);
        when(productoSpriteMock.getId()).thenReturn(1);
        when(productoSpriteMock.getPrecio()).thenReturn(precioSprite);
        when(productoSpriteMock.getDescripcion()).thenReturn("Sprite");

        Producto productoFantaMock = mock(Producto.class);
        when(productoFantaMock.getId()).thenReturn(1);
        when(productoFantaMock.getPrecio()).thenReturn(precioFanta);
        when(productoFantaMock.getDescripcion()).thenReturn("Fanta");

        //Creo una nueva oferta
```

```
		ofertaBuilder.crearNuevaOferta(ofertaDesde, ofertaHasta, "Coca +
		Sprite = Fanta al 50%");
		//Le agrego la condicion de que si lleva el producto Coca Cola y el
		Producto Sprite
		ofertaBuilder.agregarCondicion(condicionOfertaFactory.CondicionLlevaN
		Producto(productoCocaMock, 1));

		ofertaBuilder.agregarCondicion(condicionOfertaFactory.CondicionLlevaN
		Producto(productoSpriteMock, 1));

		ofertaBuilder.agregarCondicion(condicionOfertaFactory.CondicionLlevaN
		Producto(productoFantaMock, 1));

		//Le descuenta la mitad a la Fanta.
		ofertaBuilder.agregarAccion(accionOfertaFactory.AccionDescontarDeProd
		ucto(productoFantaMock, 1, 0.5));

		//Cargo la oferta al repositorio de ofertas.
		ofertasRepositorio.addOferta(ofertaBuilder.getOferta());

		//Creo la venta
		Venta venta = new Venta(Calendar.getInstance(), 1);
		//Y le cargo 1 unidades de cada uno de los prods de la promo
		venta.addProducto(productoCocaMock, 1);
		venta.addProducto(productoSpriteMock,1);
		venta.addProducto(productoFantaMock, 1);

		ofertasRepositorio.aplicarOfertas(venta);

		venta.cobrar();
		//Dado que hay una promocion Cocal + Sprite = Fanta al 50%, cuando
		lleva esos 3 productos prodcuto, entonces paga la fanta al 50%
		assertEquals(venta.getTotal(),
		precioCoca+precioSprite+(precioFanta/2),1);

}

		@Test
		//Testea una venta con descuento por marca: Llevando algun producto
		de una marca dada, hay un descuento de algun porcentaje.
		public void ventaConOfertaPorMarca()
		{
			String marca = "Coca Cola Company";

			double precioCoca = 10;
			double precioSprite = 20;
			double precioFanta = 30;

			Producto productoCocaMock = mock(Producto.class);
			when(productoCocaMock.getId()).thenReturn(1);
			when(productoCocaMock.getPrecio()).thenReturn(precioCoca);
			when(productoCocaMock.getMarca()).thenReturn(marca);

			Producto productoSpriteMock = mock(Producto.class);
			when(productoSpriteMock.getId()).thenReturn(1);
			when(productoSpriteMock.getPrecio()).thenReturn(precioSprite);
			when(productoSpriteMock.getMarca()).thenReturn(marca);

			Producto productoFantaMock = mock(Producto.class);
			when(productoFantaMock.getId()).thenReturn(1);
			when(productoFantaMock.getPrecio()).thenReturn(precioFanta);
```

```
		when(productoFantaMock.getMarca()).thenReturn(marca);

		//Creo una nueva oferta
		ofertaBuilder.crearNuevaOferta(ofertaDesde, ofertaHasta, "Marca
		Coca Cola con 50% de descuento");
		//Le agrego la condicion de que si lleva dos de un producto.
		ofertaBuilder.agregarCondicion(condicionOfertaFactory.Condicion
		MarcaProducto(marca));
		//Le descuenta la mitad a los productos de la marca.
		ofertaBuilder.agregarAccion(accionOfertaFactory.AccionDescontar
		PorMarca(marca, null, 0.5));

		//Cargo la oferta al repositorio de ofertas.
		ofertasRepositorio.addOferta(ofertaBuilder.getOferta());

		//Creo la venta
		Venta venta = new Venta(Calendar.getInstance(), 1);
		//Y le cargo 3 productos de la marca.
		venta.addProducto(productoCocaMock, 1);
		venta.addProducto(productoSpriteMock,1);
		venta.addProducto(productoFantaMock, 1);

		ofertasRepositorio.aplicarOfertas(venta);

		//Dado que hay una promocion tal que hace el 50% de desc a
		todos los prods de la marca Coca Cola,
		//cuando lleva 3 productos de $10, $20 y $30
		//Entonces solo debe cobrar la mitdad de cada uno: $30
		assertEquals(venta.getTotal(),
		precioCoca/2+precioSprite/2+precioFanta/2,1);
}

@Test
//Testea una venta con oferta de pagando con un medio de pago dado
(Efectivo en este caso) hay un descuento de algun porcentaje.
public void ventaConOfertaPorMedioDePago()
{
		double precioCoca = 10;
		double precioSprite = 20;
		double precioFanta = 30;
		String medioPago = "Efectivo";
		double descuento = 0.1;

		//
		Producto productoCocaMock = mock(Producto.class);
		when(productoCocaMock.getId()).thenReturn(1);
		when(productoCocaMock.getPrecio()).thenReturn(precioCoca);

		Producto productoSpriteMock = mock(Producto.class);
		when(productoSpriteMock.getId()).thenReturn(1);
		when(productoSpriteMock.getPrecio()).thenReturn(precioSprite);

		Producto productoFantaMock = mock(Producto.class);
		when(productoFantaMock.getId()).thenReturn(1);
		when(productoFantaMock.getPrecio()).thenReturn(precioFanta);

		//Creo una nueva oferta
		ofertaBuilder.crearNuevaOferta(ofertaDesde, ofertaHasta, "10%
		Pago en Efectivo");
		//Le agrego la condicion de que si paga con el medio de la
		oferta.
```

```
        ofertaBuilder.agregarCondicion(condicionOfertaFactory.Condicion
        MedioDePago(medioPago));
        //Le descuenta el porcentaje de la oferta.
        ofertaBuilder.agregarAccion(accionOfertaFactory.AccionDescontar
        DeVentaCompleta(descuento));

        //Cargo la oferta al repositorio de ofertas.
        ofertasRepositorio.addOferta(ofertaBuilder.getOferta());

        //Creo la venta
        Venta venta = new Venta(Calendar.getInstance(), 1);
        //Y le cargo 3 productos de la marca.
        venta.addProducto(productoCocaMock, 1);
        venta.addProducto(productoSpriteMock,1);
        venta.addProducto(productoFantaMock, 1);
        venta.setFormaDePago(medioPago);

        ofertasRepositorio.aplicarOfertas(venta);
        venta.cobrar();
        //Dado que hay una promocion tal que hace el 10% de desc a
        todos los prods si se paga con efectivo,
        //cuando lleva 3 productos de $10, $20 y $30
        //Entonces debe cobrar $54
        assertEquals((precioCoca+precioSprite+precioFanta)* (1-
        descuento), venta.getTotal(),1);

}
```