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

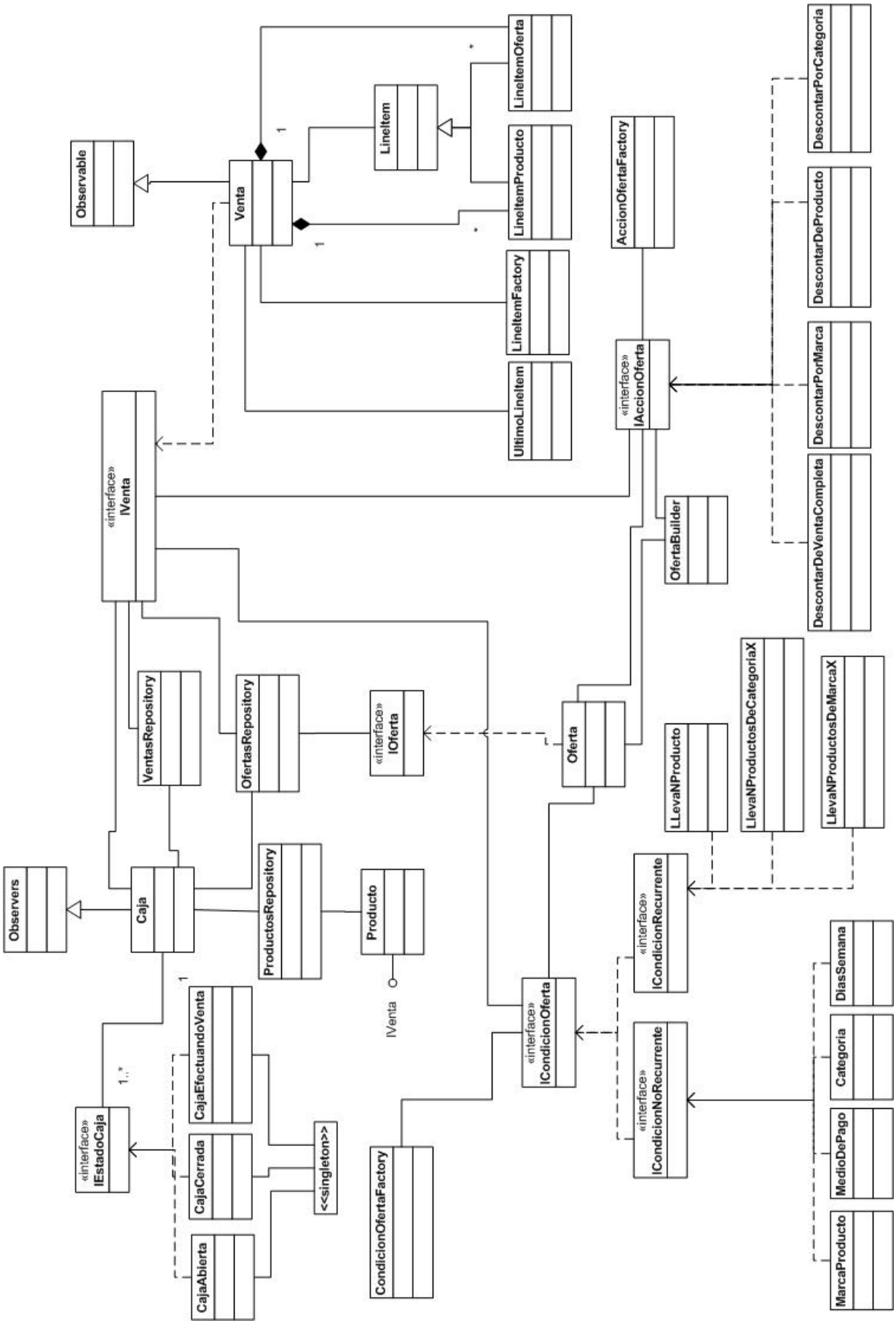
Se tendrán en cuenta también la completitud del tp, la correctitud, distribución de responsabilidades, aplicación y uso de criterios y principios de buen diseño.

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.

Diagrama de clases del modelo utilizado



Test unitarios

A continuación se muestran algunos de los test unitarios implementados

Test de Producto

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

	@Test
	public void disminuirCantidadDeProductosEnUnoAlQuitarProductoTest() {

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

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

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

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

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

Test VentasRepository

```
@Test
	public void obtenerMediosDePagoTodosDistintosTest(){

		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 obtenerMediosDePagoHabiendoRepetidosTest(){

		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 marcarProductosComoOfertadosConExcepcionesTest(){

		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 integración entre Oferta y Venta

```
@Test
	public void aplicarDescuentoPorCategoriaTest() {
		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
				.CondicionCategoria(categoria));

		ofertaBuilder.agregarAccion(accionOfertaFactory
				.AccionDescontarDeVentaCompleta(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 ventaConOfertaLlevaNpagaMTest() {
	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
			.CondicionLlevaNProducto(productoMock, 2));
	// Le descuenta una unidad.
	ofertaBuilder.agregarAccion(accionOfertaFactory
			.AccionDescontarDeProducto(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 ventaConOfertaLlevandoNyMpagaMenosElZTest() {
	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
                        .CondicionLlevaNProducto(productoCocaMock, 1));
        ofertaBuilder.agregarCondicion(condicionOfertaFactory
                        .CondicionLlevaNProducto(productoSpriteMock, 1));
        ofertaBuilder.agregarCondicion(condicionOfertaFactory
                        .CondicionLlevaNProducto(productoFantaMock, 1));
        // Le descuenta la mitad a la Fanta.
        ofertaBuilder.agregarAccion(accionOfertaFactory
                        .AccionDescontarDeProducto(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 ventaConOfertaPorMarcaTest() {
        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
                        .CondicionMarcaProducto(marca));
        // Le descuenta la mitad a los productos de la marca.
        ofertaBuilder.agregarAccion(accionOfertaFactory
                        .AccionDescontarPorMarca(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 ventaConOfertaPorMedioDePagoTest() {
        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
                        .CondicionMedioDePago(medioPago));
        // Le descuenta el porcentaje de la oferta.
        ofertaBuilder.agregarAccion(accionOfertaFactory
                        .AccionDescontarDeVentaCompleta(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);

    }
```