

## ***Solución Ejercicio 1 - Práctica 4 (Diseño de pruebas de caja negra)***

**Entrada1** : Cliente = (nif, estado, deuda)

Clases válidas : CV1 = (nif\_registrado, normal, 0)

CV2 = (nif\_registrado, moroso, <= 1000)

Clases NO válidas : CNV1 = (nif\_NO\_registrado, cualquier\_estado, 0)

CNV2 = NULL

CNV3 = (nif\_registrado, moroso, > 1000)

**Entrada2** : lista de artículos (código)

Clases válidas : AV1 = lista con artículos en BD y sin códigos repetidos

AV2 = lista con artículos en BD y con códigos repetidos

Clases NO válidas : ANV1 = NULL

ANV2 = lista con algún artículo que no está en BD

ANV3 = Lista vacía

**Entrada3** : Base de datos

Clase válida : BV1 = Acceso Ok (no genera error de acceso)

Clase NO válida : BNV1 = genera error: "Error al recuperar datos del artículo"

**Salidas**: Objeto TicketTO (cliente, líneas, precioTotal) o excepción lanzada (BOException)

Clase válida : SV1 = objeto TicketTO

cliente = mismo cliente de entrada

líneas = lista <lineaVentaTO> con los mismos artículos que la lista de entrada

artículo = cada artículo de entrada

unidades = número de veces que aparece el artículo en la lista

precioLinea = unidades \* precioArtículo

precioTotal = Suma de precioLinea de todos los artículos de la lista

Clases NO válidas : SNV1 = BOException: "El cliente no puede realizar la compra"

SNV2 = BOException: "El artículo no está en la BD"

SNV3 = BOException: "Error al recuperar datos del artículo"

SNV4 = ??? No se especifica la salida cuando la lista de artículos es NULL o vacía

<b>Id</b>	<b>Combinaciones de Clases</b>
1	CV1-AV1-BV1-SV1
2	CV2-AV2-BV1-SV1
3	CNV1-AV1-BV1-SNV1
4	CNV2-AV1-BV1-SNV1
5	CNV3-AV1-BV1-SNV1
6	CV1-ANV1-BV1-SNV4
7	CV1-ANV2-BV1-SNV2
8	CV1-AV1-BNV1-SNV3
9	CV1-ANV3-BV1-SNV4

## Casos de prueba:

Suponemos que la base de datos contiene los siguientes Artículos:

código    precio

Ipad        2€

Imac        3€

Suponemos que el nif 00000000T está registrado y el nif 11111111H no lo está.

Id	Datos Entrada					Resultado Esperado		
	Cliente			Lista Artículos	Acceso BD	TicketTO o BOException		
	Nif	Estado	Deuda			Cliente	líneas	precio Total
1	00000000T	normal	0	["Ipad", "Imac"]	ok	(00000000T , normal, 0)	{{["Ipad", 2],1, 2} [["Imac",3] 1, 3]}	5
2	00000000T	moroso	500	["Ipad", "Imac", "Ipad"]	ok	(00000000T , moroso, 500)	{{["Ipad", 2],2, 4} [["Imac", 3], 1, 3]}	7
3	11111111H	normal	0	["Ipad", "Imac"]	ok	<b>BOException</b> con mensaje "El cliente no puede realizar la compra"		
4	NULL			["Ipad", "Imac"]	ok	<b>BOException</b> con mensaje "El cliente no puede realizar la compra"		
5	00000000T	moroso	2000	["Ipad", "Imac"]	ok	<b>BOException</b> con mensaje "El cliente no puede realizar la compra"		
6	00000000T	normal	0	NULL	ok	???		
7	00000000T	normal	0	["tablet", "Ipad", "Imac"]	ok	<b>BOException</b> con mensaje "El artículo no está en la BD"		
8	00000000T	normal	0	["Ipad", "Imac"]	fallo	<b>BOException</b> con mensaje "Error al recuperar datos del artículo"		
9	00000000T	normal	0	[ ]	ok	???		

### ***Solución Ejercicio 3 - Práctica 4 (Diseño de pruebas de caja negra)***

**Entidad a modelar:** terminal de ventas de un supermercado. Sobre esta entidad queremos gestionar el proceso de venta de productos en un supermercado.

Estados del sistema: El terminal de ventas estará en uno de los siguientes posibles estados

- Autenticando al encargado del terminal
- En espera
- Escaneando productos
- Procesando el pago
- Saliendo del sistema

Eventos del sistema: las entradas que pueden provocar cambios de estado son:

- Password(cod): el encargado introduce un password con valor “cod”
- Cliente(prods): el cliente coloca sobre la cinta transportadora una lista de productos
- Escanear(prod): escaneamos un producto cuyo código es “prod”
- Pagar(codTarjeta): procedemos a pagar con la tarjeta con código “codTarjeta”
- AnularCompra
- SalirSistema

Guardas del sistema:

- cod\_invalido: el código introducido por el encargado es inválido
- fallo\_escáner: al escanear un producto el escáner falla
- quedanProd: quedan productos por escanear
- ultimoProd: estamos escaneando el último producto
- tarj\_valida: el código de la tarjeta es un código válido
- finTurno: el turno del cajero ha finalizado

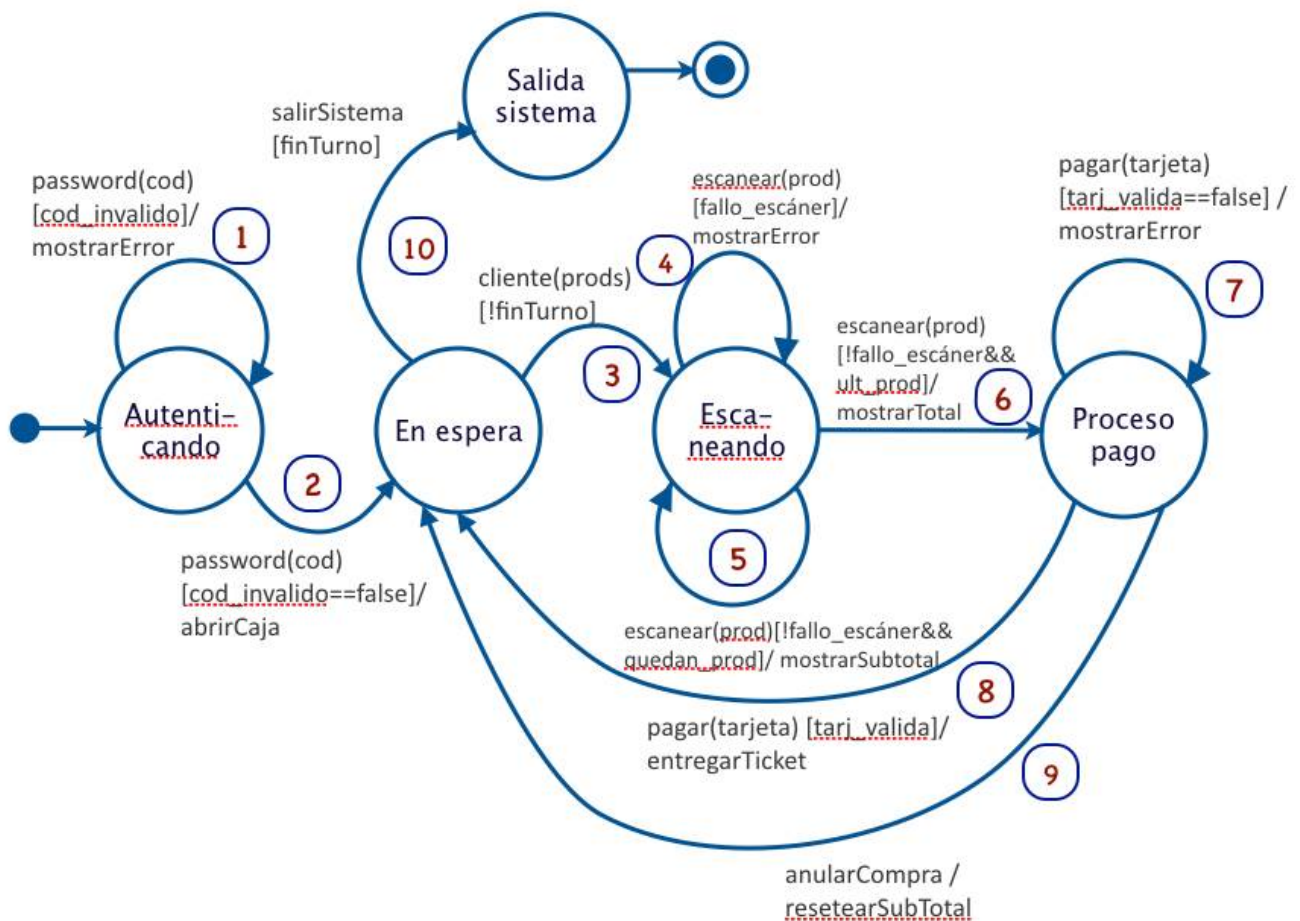
Acciones del sistema:

- mostrarError: se le informa al usuario de que la entrada es errónea
- abrirCaja: se activa el uso de la caja
- mostrarSubtotal: se informa del precio acumulado de los productos escaneados
- mostrarTotal: se muestra el precio total de todos los productos escaneados, es igual al valor del subtotal, cuando ya se ha escaneado el último producto
- entregarTicket: se le entrega el ticket de compra al cliente
- tarj\_valida: el código de la tarjeta es un código válido
- resetearSubtotal: se pone a cero el valor del subtotal acumulado

Diagrama de transición de estados:

Para diseñar los casos de prueba, **asumimos que:**

- el password que debe introducir el encargado es “1234”
- el producto con código 100, tiene un precio de 10
- el código de tarjeta válida es el “222”
- el ticket de compra tiene el formato: ((producto1,unidades1), ..., (productoN, unidadesN), total))



#### Casos de prueba:

- Transiciones cubiertas: 1, 2, 3, 4, 6, 9, 10

• password(1111)  
 • password(1234)  
 • cliente(100)  
 • escanear(100) && fallo\_escáner  
 • escanear(100) && !fallo\_escáner  
 • anular\_compra  
 • salirSistema

Datos de entrada

- Resultado esperado: mostrarError(password 1111), abrirCaja, mostrarError(fallo\_escanner), mostrarTotal(10 euros), resetearSubtotal, Compra anulada (Salida sin compra)

- Transiciones cubiertas: 2, 3, 5, 5, 6, 7, 8, 10

• password(1234)  
 • cliente(100,100)  
 • escanear(100)&&!fallo\_escáner&&quedan\_prod  
 • escanear(100)&&!fallo\_escáner&&ult\_prod  
 • pagar(111)  
 • pagar(222)  
 • salirSistema

Datos de entrada

- Resultado esperado: abrirCaja, mostrarSubtotal(10 euros,), mostrarTotal(20 euros), entregarTicket((100,2), 20)