PL2 Ingeniería del Software

Ejercicio 3

Cristina Martínez Toledo: 09109126E
Roberto Seco Volkava: 09854422A

Índice

Diseño de alto nivel mediante ingeniería inversa..... 3
Código original..... 3
Resultados de pruebas de caja blanca..... 6
Resultados de pruebas de caja negra..... 7

Diseño de alto nivel mediante ingeniería inversa

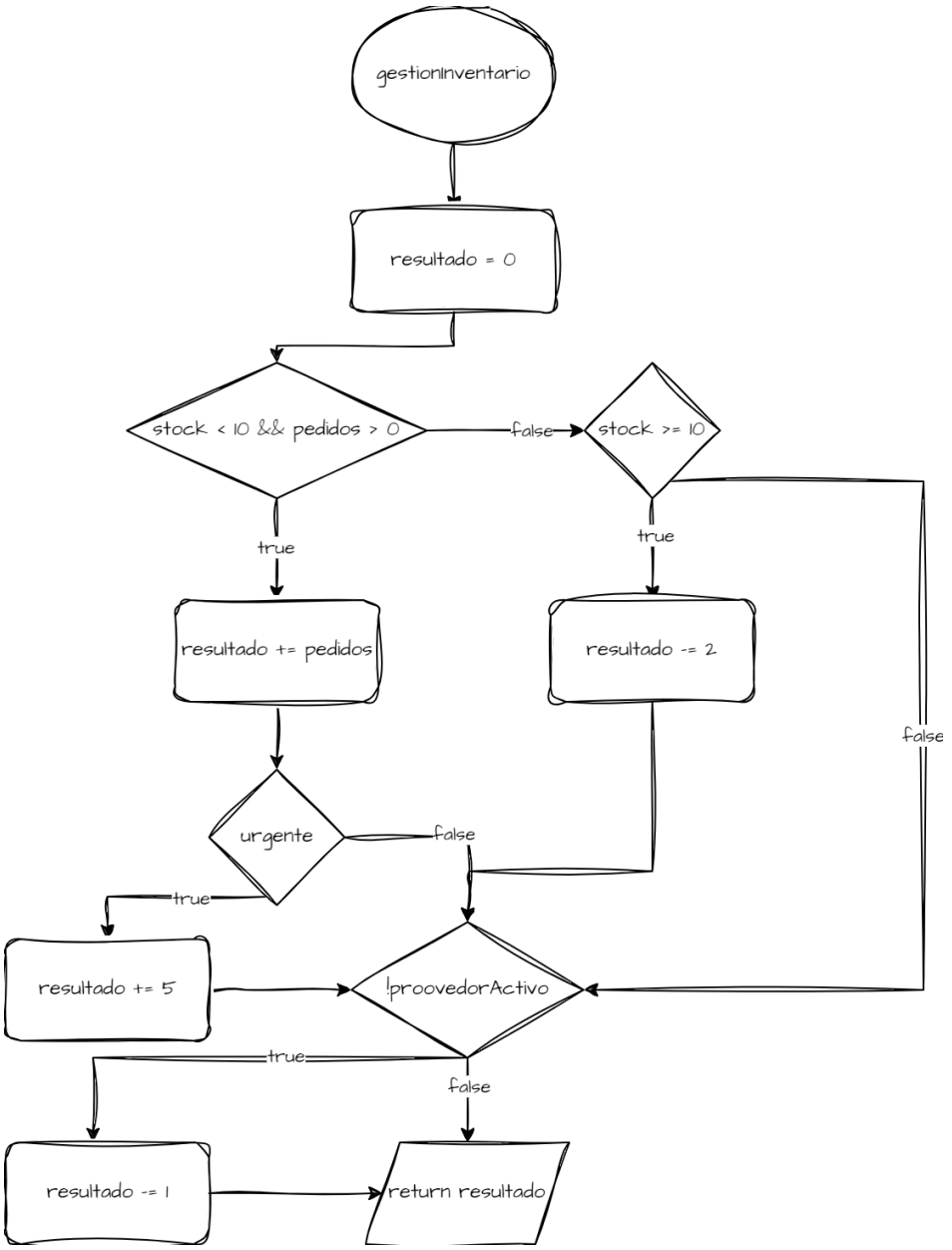
Código original

```java
public int gestionInventario(int stock, int pedidos, boolean urgente, boolean proveedorActivo) {
    int resultado = 0;
    if (stock < 10 && pedidos > 0) {
        resultado += pedidos;
        if (urgente) {
            resultado += 5; // margen de urgencia
        }
    } else if (stock >= 10) {
        resultado -= 2; // pequeña penalización por sobrealmacenaje
    }
    if (!proveedorActivo) {
        resultado = -1; // error en la gestión
    }
    return resultado;
}
```

Primero analizaremos los parámetros de entrada.

1. int stock: representa la cantidad de productos que tenemos en nuestro inventario
2. int pedidos: representa la cantidad de pedidos realizados sobre nuestro stock
3. boolean urgente: indica si el pedido es urgente o no
4. boolean proveedorActivo: indica si el proveedor está activo o no

A partir de este código se puede extraer un diagrama que permite representar el flujo de ejecución del método. De esta manera se facilita comprender qué hace el código y para qué sirve.



Sabiendo esto, podemos determinar que el código corresponde a un método de gestión de inventarios. Primero se inicializa una variable resultado, que será la que posteriormente se devuelve.

Si el stock es menor que 10 y hay pedidos entonces se incrementa el resultado en tantos pedidos como hayan. Si además es urgente se le añade un margen de urgencia. Si la condición no se cumpliese, bien porque no hay pedidos, o porque hay suficiente stock se deberá comprobar si esto fue por suficiente stock o por falta de pedidos.

Si era porque teníamos un exceso de stock entonces se aplica la penalización. Si no, es porque estábamos ante un caso de falta de pedidos.

A continuación se debe comprobar si el proveedor no está activo. Si no lo está, se pone el resultado a -1 para alertar de un error. Finalmente se devuelve el valor de la variable resultado.

Para resumir un poco esta explicación podríamos decir que el método comprueba si estamos teniendo un sobre almacenaje, lo cuál no nos interesa, o si la empresa está haciendo las cosas bien, devolviendo una especie de puntaje sobre las operaciones

Resultados de pruebas de caja blanca

En las pruebas de caja blanca nos interesa analizar el control de flujo del código, por lo que deberemos usar diferentes casos que cubran las combinaciones de if-else que se presentan.
En este caso, hemos utilizado el método del camino básico, por lo que, partiendo del grafo que define el código, se generarán casos de prueba para cada uno de los caminos básicos, seleccionando las entradas adecuadas para asegurar que todos se prueben. Según el grafo ya obtenido en el ejercicio 2, se puede ver que habría 8 caminos posibles, aunque, como el resultado siempre será -1 si proveedorActivo tiene valor false, se podría reducir a solamente 5 caminos básicos. Debido a esto, las pruebas realizadas son:

Número	Stock	Pedidos	Urgente	proovedorActivo	Resultado	Razonamiento
1	5	3	true	true	8	Entra en el primer y el segundo if, por lo que suma pedidos y margen
2	5	3	false	true	3	Solo entra en el primer if, por lo que no se añade el margen
3	15	0	false	true	-2	Entra en el “elif”; se penaliza
4	2	0	false	true	0	No entra en ninguna de las condiciones
5	8	1	true	false	-1	Entra en el último if (Proveedor inactivo)

Resultados de pruebas de caja negra

Para las pruebas de caja negra nos interesa ver para una entrada si la salida es la apropiada, sin centrarnos en el flujo intermedio.
En este caso, hemos empleado pruebas de clases de equivalencia. En estas, debido al código dado, no hemos añadido pruebas no válidas

Clases de equivalencia:
1. Para stock
    -s1: stock < 10: válida
    -s2: stock >= 10: válida

Podría haber una tercera inválida, s3, con stock < 0, si asumiéramos que no debe haber stock negativo, pero según el código tal cual no la añadiremos

2. pedidos
    -p1: pedidos > 0: válida
    -p2: pedidos <= 0: válida

De nuevo, se podría hacer una tercera inválida, p3, si pedidos < 0, si asumimos que no puede haber pedidos negativos, pero no la añadiremos

3. urgente
    -u1: urgente == true: válida
    -u2: urgente == false: válida

4. proveedorActivo
    -pr1: proveedorActivo == true: válida
    -pr2: proveedorActivo == false: válida

Para crear los casos de prueba, se deben estudiar todas las combinaciones posibles (2^4=16):

Caso	Stock	Pedidos	Urgente	Prov.Activo	Resultado
1	s1 (<10)	p1 (>0)	u1 (true)	pr1 (true)	15
2	s1 (<10)	p1 (>0)	u1 (true)	pr2 (false)	-1
3	s1 (<10)	p1 (>0)	u2 (false)	pr1 (true)	10
4	s1 (<10)	p1 (>0)	u2 (false)	pr2 (false)	-1
5	s1 (<10)	p2 (<=0)	u1 (true)	pr1 (true)	0
6	s1 (<10)	p2 (<=0)	u1 (true)	pr2 (false)	-1
7	s1 (<10)	p2 (<=0)	u2 (false)	pr1 (true)	0
8	s1 (<10)	p2 (<=0)	u2 (false)	pr2 (false)	-1

9	s2 (>=10)	p1 (>0)	u1 (true)	pr1 (true)	-2
10	s2 (>=10)	p1 (>0)	u1 (true)	pr2 (false)	-1
11	s2 (>=10)	p1 (>0)	u2 (false)	pr1 (true)	-2
12	s2 (>=10)	p1 (>0)	u2 (false)	pr2 (false)	-1
13	s2 (>=10)	p2 (<=0)	u1 (true)	pr1 (true)	-2
14	s2 (>=10)	p2 (<=0)	u1 (true)	pr2 (false)	-1
15	s2 (>=10)	p2 (<=0)	u2 (false)	pr1 (true)	-2
16	s2 (>=10)	p2 (<=0)	u2 (false)	pr2 (false)	-1