UD3.EXAME Práctico

DAM1-Contornos de Desenvolvemento 2024-25
26/03/2025

- Puedes utilizar apuntes y materiales que consideres pero deberás realizar los programas individualmente. En caso contrario se retirará el examen.
- Indica la autoría del código incluyendo un comentario con tu nombre y apellidos.

1. **Descarga,** se é o caso, o código fonte do exercicio do **repositorio** (ou onde che indique o profesor). Recoméndase configuralo nun novo proxecto Java. Precisarás a librería de probas de Junit5.
2. Resposta no propio documento editable, amplía as táboas se é necesario.
3. **Entrega:** O documento coas respostas en PDF e os ficheiros de código fonte que xeraras (Clases de probas solicitadas)
4. **Tiempo máximo:** 1:30 horas

1. Sistema de Bonificacións en Compras Online (6)

Dado o seguinte **enunciado** e a **implementación** levada a cabo pol@ programador@, aplica a técnica de Proba do Camiño Básico e realiza as seguintes tarefas:

1. Representa o **grafo de fluxo**
2. Calcula a **complexidade ciclomática** (de McCabe)
3. Detalla os **camiños independentes** e elabora os **casos de proba**
4. Implementa unha clase de proba en Junit5.
    a. Engade un comentario de Autoría na clase de probas.
5. Executa as probas, analiza os resultados, identifica posibles erros no código e como correxilos.
    a. Engade captura do resultado da execución das probas amosando cobertura.

> **Nota:** Para probar valores decimais utiliza o seguinte método:
> `assertEquals(valor esperado, valor real, tolerancia)`
> onde `tolerancia` é a marxe de erro tolerada na proba por cuestións de precisión. Podes usar o valor **`0.01`**

Enunciado

Unha empresa aplica bonificacións aos clientes segundo o total da súa compra e se pertencen ao programa de fidelización. As regras son:

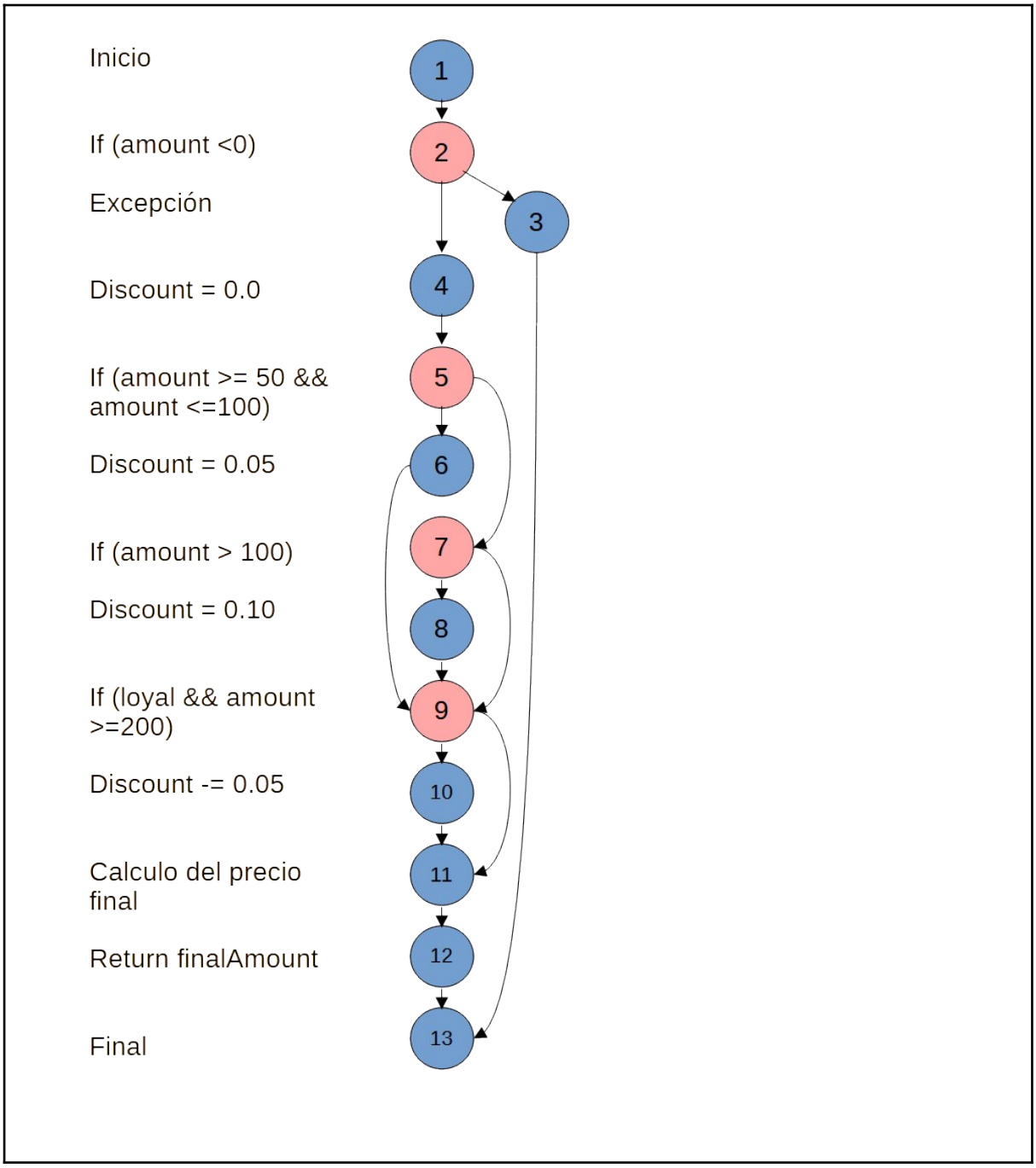
1. O importe da compra non pode ser inferior a 0€ (se ocorre, debe lanzar unha excepción).
2. Se o importe da compra é menor de 50€, non hai desconto.
3. Se está entre 50€ e 100€, aplícase un 5% de desconto.
4. Se iguala ou supera os 100€, aplícase un 10% de desconto.
5. Se o cliente pertence ao programa de fidelización e a compra iguala ou supera os 200€, aplícase un desconto extra do 5%.

Implementación

```
public class DiscountSystem {
    public static double applyDiscount(double amount, boolean isLoyalCustomer) { // 1
        if (amount < 0) { // 2
            throw new IllegalArgumentException("O importe non pode ser negativo"); // 3
        }

        double discount = 0.0; // 4

        if (amount >= 50 && amount <= 100) { // 5
            discount = 0.05; // 6
        } else if (amount > 100) { // 7
            discount = 0.10; // 8
        }

        if (isLoyalCustomer && amount >= 200) { // 9
            discount -= 0.05; // 10
        }

        double finalAmount = amount - (amount * discount); // 11

        return finalAmount; // 12
    }
} // 13
```

Solución

Grafo de fluxo



Complexidade ciclomática

V(G) = 4 nodos predicados + 1 = 5

Camiños independentes

1. **1,2,3,13**
2. **1,2,4,5,6,9,10,11,12,13**
3. **1,2,4,5,6,9,11,12,13**
4. **1,2,4,5,7,8,9,10,11,12,13**
5. **1,2,4,5,7,8,9,11,12,13**

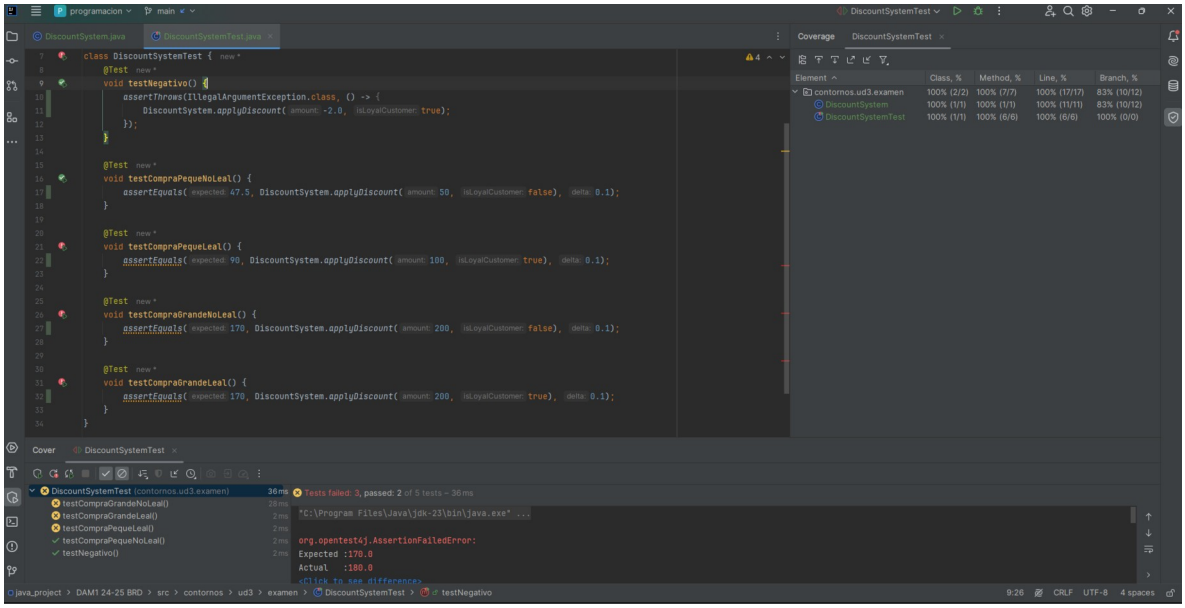
Casos de proba

Entrada	Valor Esperado
-2.0, true	"O importe non pode ser negativo"
50.0, false	47,5
100.0, true	90
200.0, false	170
200.0, true	170

Clase de Probas en Junit5

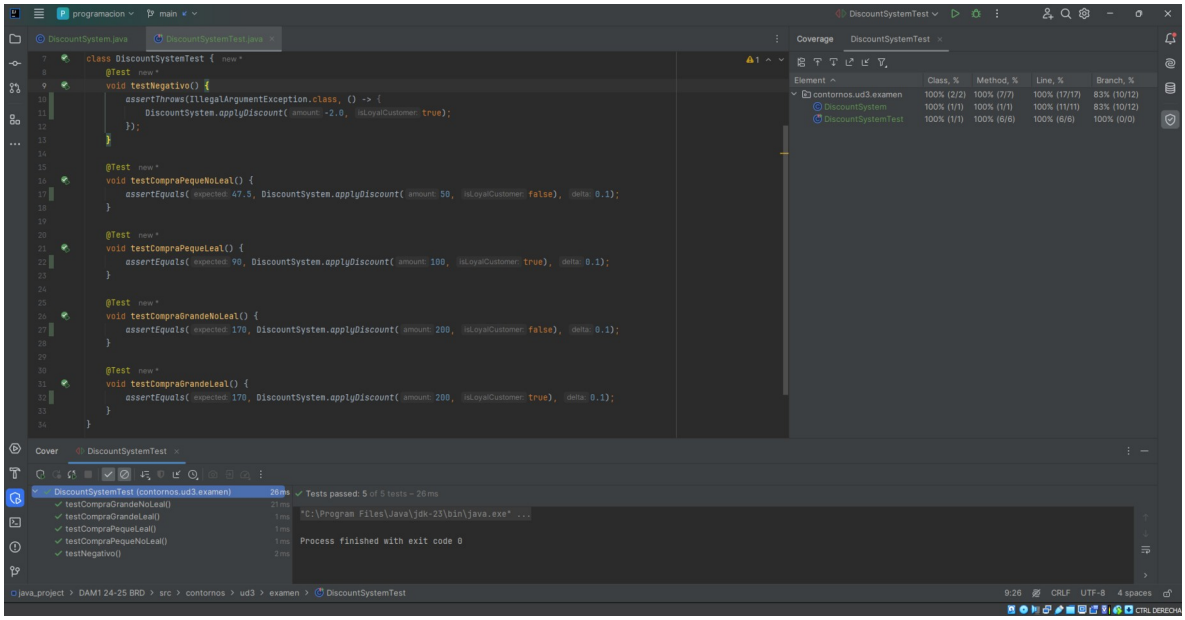
```
class DiscountSystemTest {
    @Test
    void testNegativo() {
        assertThrows(IllegalArgumentException.class, () -> {
            DiscountSystem.applyDiscount(-2.0, true);
        });
    }

    @Test
    void testCompraPequeNoLeal() {
        assertEquals(47.5, DiscountSystem.applyDiscount(50, false), 0.1);
    }

    @Test
    void testCompraPequeLeal() {
        assertEquals(90, DiscountSystem.applyDiscount(100, true), 0.1);
    }

    @Test
    void testCompraGrandeNoLeal() {
        assertEquals(170, DiscountSystem.applyDiscount(200, false), 0.1);
    }

    @Test
    void testCompraGrandeLeal() {
        assertEquals(170, DiscountSystem.applyDiscount(200, true), 0.1);
    }
}
```

Captura/s do resultado de executar as probas



Erro/s atopados

```
public class DiscountSystem {
    public static double applyDiscount(double amount, boolean isLoyalCustomer) { // 1
        if (amount < 0) { // 2
            throw new IllegalArgumentException("O importe non pode ser negativo"); // 3
        }

        double discount = 0.0; // 4

        if (amount >= 50 && amount <= 100) { // 5
            discount = 0.05; // 6
        } else if (amount >= 100) { // 7 ERROR: El enunciado nos dice mayor o igual a 100 euros
            discount = 0.10; // 8
        }

        if (isLoyalCustomer || amount >= 200) { // 9 ERROR: Se comprueba que se cumplan las dos, no una de las dos
            discount += 0.05; // 10 ERROR: Se está restando el descuento en vez de sumarse
        }

        double finalAmount = amount - (amount * discount); // 11

        return finalAmount; // 12
    }
} // 13
```


programacion
main
DiscountSystemTest
DiscountSystem.java
DiscountSystemTest.java
class DiscountSystemTest {
@Test
void testNegativo() {
assertThrows(IllegalArgumentException.class, () -> {
DiscountSystem.applyDiscount(amount: -2.0, isLoyalCustomer: true);
});
}
@Test
void testCompraPequeNoLeal() {
assertEquals(expected: 47.5, DiscountSystem.applyDiscount(amount: 50, isLoyalCustomer: false), delta: 0.1);
}
@Test
void testCompraPequeLeal() {
assertEquals(expected: 90, DiscountSystem.applyDiscount(amount: 100, isLoyalCustomer: true), delta: 0.1);
}
@Test
void testCompraGrandeNoLeal() {
assertEquals(expected: 170, DiscountSystem.applyDiscount(amount: 200, isLoyalCustomer: false), delta: 0.1);
}
@Test
void testCompraGrandeLeal() {
assertEquals(expected: 170, DiscountSystem.applyDiscount(amount: 200, isLoyalCustomer: true), delta: 0.1);
}
}
Coverage
DiscountSystemTest
Element
Class, %
Method, %
Line, %
Branch, %
contornos.ud3.examen
100% (2/2)
100% (7/7)
100% (17/17)
83% (10/12)
DiscountSystem
100% (1/1)
100% (1/1)
100% (11/11)
83% (10/12)
DiscountSystemTest
100% (1/1)
100% (6/6)
100% (6/6)
100% (0/0)
Cover
DiscountSystemTest
DiscountSystemTest (contornos.ud3.examen) 26 ms
testCompraGrandeNoLeal() 21 ms
testCompraGrandeLeal() 1 ms
testCompraPequeLeal() 1 ms
testCompraPequeNoLeal() 1 ms
testNegativo() 2 ms
Tests passed: 5 of 5 tests – 26 ms
"C:\Program Files\Java\jdk-23\bin\java.exe" ...
Process finished with exit code 0
java_project › DAM1 24-25 BRD › src › contornos › ud3 › examen › DiscountSystemTest
9:26
CRLF
UTF-8
4 spaces
CTRL DERECHA

2. Validación de correos electrónicos (4)

Dado o seguinte **enunciado** e a **implementación** levada a cabo pol@ programador@:

1. Crea unha táboa de clases de equivalencia
2. Xera casos de proba correspondentes indicando as clases de equivalencia cubiertas en cada caso.
3. Implementa unha clase de proba en Junit5.
    a. Engade un comentario de Autoría na clase de probas.
4. Executa as probas e amosa o resultado.

Enunciado:

Crea unha clase EmailValidator con un método isValid(String email). O correo é válido se:

- É distinto de null
- Contén exactamente un símbolo @.
- O dominio (parte despois do @) ten polo menos un punto (.).
- O nome de usuario (parte antes do @) non está baleiro.

Implementación:

```
public class EmailValidator {
    public static boolean isValid(String email) {
        if (email == null || !email.contains("@")) {
            return false;
        }
        String[] parts = email.split("@");
        if (parts.length != 2 || parts[0].isEmpty()) {
            return false;
        }
        return parts[1].contains(".");
    }
}
```

Solución

Táboa de Clases de Equivalencia

Condición de Entrada	Clases Válidas	Clases Non Válidas
email	No es null [1]	Es null [2]
	Tiene un solo “@” [3]	Tiene dos @ [4] No tiene @ [5]
	El dominio tiene uno o más punto [6]	El dominio NO tiene un punto [7]
	El nombre de usuario no está vacío [8]	El nombre de usuario está vacío [9]

Condición de Entrada	Clases Válidas	Clases Non Válidas
Email no es null	"test@gmail.com” [1]	“” [2]
Email tiene un solo @	"test@gmail.com” [3]	"test@@gmail.com" [4] "testgmail.com" [5]
O dominio ten polo menos un punto	"test@gmail.com” [6]	"test@gmailcom” [7]
O nome de usuario non está baleiro	"test@gmail.com” [8]	"@gmail.com” [9]

Casos de proba con clases de equivalencia válidas

Entrada1	Clases incluidas
Email = "test@gmail.com”	[1] [3] [6] [8]

Casos de proba con clases de equivalencia non válidas

Entrada1	Clases incluidas
Email = “”	[2] [5] [7] [9]
Email = "test@@gmail.com"	[1] [4] [6] [8]
Email = "testgmail.com"	[1] [5] [6] [8]
Email = "test@gmailcom”	[1] [3] [7] [8]
Email = "@gmail.com”	[1] [3] [6] [9]

Clase de Probas en Junit5

```
class EmailValidatorTest {
    @Test
    void esValido() {
        assertEquals(true,EmailValidator.isValid("test@gmail.com"));
    }

    @Test
    void esNullOrEmpty() {
        assertEquals(false,EmailValidator.isValid(""));
    }

    @Test
    void tieneDosArrobas() {
        assertEquals(false,EmailValidator.isValid("test@@gmail.com"));
    }

    @Test
    void noTieneArrobas() {
        assertEquals(false,EmailValidator.isValid("testgmail.com"));
    }

    @Test
    void noTienePunto() {
        assertEquals(false,EmailValidator.isValid("test@gmailcom"));
    }

    @Test
    void tieneUser() {
        assertEquals(false,EmailValidator.isValid("@gmail.com"));
    }

}
```

Captura do resultado de executar as probas