ENTREGABLE 3 ENTORNOS DE DESARROLLO

ENUNCIADO 1:

1.- Descripción:

Se han realizado pruebas unitarias sobre los métodos del programa de gestión de notas utilizando pruebas de caja blanca, caja negra y JUnit5. Todas las pruebas se han realizado satisfactoriamente sin errores.

2.- Métodos probados:

- calcularMedia
- encontrarNotaMaxima
- contarAprobados
- actualizarNota

3.- Criterios de cobertura:

a) Método calcularMedia:

```
public static double calcularMedia(double[] notas) {
        if (notas == null || notas.length == 0) {
                return 0;
        }
        double suma = 0;
        for (int i = 0; i < notas.length; i++) {
                suma += notas[i];
        }
        return suma / notas.length;
}
```

Se ha cubierto el if en caso de que el array de notas esté vacío o en caso de que sea null (no exista).
También se ha cubierto el bucle for modificando el array de notas, dado que dicho bucle depende de la amplitud de ese array. Se han realizado pruebas con 1 número en el array y con 10.
En las pruebas de caja blanca se hicieron pruebas siguiendo todos los caminos posibles, modificando el array. En las pruebas de caja negra se llamó al método desde otra clase directamente probando diferentes casos y en JUnit se realizó un @Test para cada caso tipo.

b) Método encontrarNotaMaxima

```
public static double encontrarNotaMaxima(double[] notas) {
	if (notas == null || notas.length == 0) {
		return 0;
	}
	double max = notas[0];
	for (int i = 1; i < notas.length; i++) {
		if (notas[i] > max) {
			max = notas[i];
		}
	}
	return max;
}
```

Se ha cubierto el if en caso de que el array de notas esté vacío o en caso de que sea null (no exista).
También se ha cubierto el bucle for modificando el array de notas, dado que dicho bucle depende de la amplitud de ese array. Se han realizado pruebas con 1 número en el array, siendo ese número el valor límite y con 10.
En las pruebas de caja blanca se hicieron pruebas siguiendo todos los caminos posibles, modificando el array. En las pruebas de caja negra se llamó al método desde otra clase directamente probando diferentes casos y en JUnit se realizó un @Test para cada caso tipo.

c) Método contarAprobados

```
public static int contarAprobados(double[] notas) {
	if (notas == null) {
		return 0;
	}
	int contador = 0;
	for (int i = 0; i < notas.length; i++) {
		if (notas[i] >= 5.0) {
			contador++;
		}
	}
	return contador;
}
```

Se ha cubierto el if en caso de que el array no exista, sea null.
También se ha cubierto el bucle for modificando el array de notas, dado que dicho bucle depende de la amplitud de ese array. Se han realizado pruebas con 1 número en el array, en los casos de que el número sea el valor límite, sea por debajo del mismo o por encima y con un array de 10 números con aprobados y suspensos.

En las pruebas de caja blanca se hicieron pruebas siguiendo todos los caminos posibles, modificando el array. En las pruebas de caja negra se llamó al método desde otra clase directamente probando diferentes casos y en JUnit se realizó un @Test para cada caso tipo.

d) Método actualizarNota

```
public static void actualizarNota(double[] notas, int indice, double nuevaNota) {
        if (notas == null) {
                return;
        }
        if (indice >= 0 && indice < notas.length) {
                if (nuevaNota < 0) {
                        notas[indice] = 0;
                } else if (nuevaNota > 10) {
                        notas[indice] = 10;
                } else {
                        notas[indice] = nuevaNota;
                }
        }
}
```

Se ha cubierto el if en caso de que el array no exista, sea null.
También se ha cubierto el siguiente if en caso de que el índice sea <0 o que dicho índice sea mayor que la longitud del array. También en los casos en los que la nota a actualizar sea < 0, 0, 10 y >10. Así como una modificación de la nota dentro de los valores.
En las pruebas de caja blanca se hicieron pruebas siguiendo todos los caminos posibles, modificando el array. En las pruebas de caja negra se llamó al método desde otra clase directamente probando diferentes casos y en JUnit se realizó un @Test para cada caso tipo.

ENUNCIADO 2:

1.- Descripción:

Se han realizado pruebas unitarias sobre los métodos del programa de gestión de pedidos de una tienda utilizando pruebas de caja blanca, caja negra y JUnit5. Todas las pruebas se han realizado satisfactoriamente sin errores.

2.- Métodos probados:

- calcularTotalPedido
- aplicarDescuento
- contarProductosCaros
- actualizarPrecio

3.- Criterios de cobertura:

a) Método calcularTotalPedido

```
public static double calcularTotalPedido(double[] precios) {
        if (precios == null) {
                return 0;
        }
        double total = 0;
        for (int i = 0; i < precios.length; i++) {
                total += precios[i];
        }
        return total;
}
```

Se ha cubierto el if en caso de que el array de precios sea null (no exista).
También se ha cubierto el bucle for modificando el array de notas, dado que dicho bucle depende de la amplitud de ese array. Se han realizado pruebas con 1 número en el array y con varios.
En las pruebas de caja blanca se hicieron pruebas siguiendo todos los caminos posibles, modificando el array. En las pruebas de caja negra se llamó al método desde otra clase directamente probando diferentes casos y en JUnit se realizó un @Test para cada caso tipo.

b) Método aplicarDescuento

```
public static double aplicarDescuento(double total) {
        if (total < 0) {
                return 0;
        }
        double descuento = 0;
        if (total >= 100) {
                descuento = 0.10;
        } else if (total >= 50) {
                descuento = 0.05;
        }
        return total - (total * descuento);
}
```

Se ha cubierto el if en caso de que la variable total sea menor de 0.
También se ha cubierto el resto de if en caso de que la variable sea <50 (no aplica descuento), sea 50 o >50 y <100 (se aplica descuento de 0,05) y en caso de que sea 100 o >100 (se aplica descuento de 0,10).
En las pruebas de caja blanca se hicieron pruebas siguiendo todos los caminos posibles, modificando la variable. En las pruebas de caja negra se llamó al método desde otra clase directamente probando diferentes casos y en JUnit se realizó un @Test para cada caso tipo.

c) Método contarProductosCaros

```
public static int contarProductosCaros(double[] precios, double umbral) {
        if (precios == null) {
                return 0;
        }
        int contador = 0;
        for (int i = 0; i < precios.length; i++) {
                if (precios[i] > umbral) {
                        contador++;
                }
        }
        return contador;
}
```

Se ha cubierto el if en caso de que el array precios sea null (no exista).
También se ha cubierto el bucle for modificando el array de precios, dado que dicho bucle depende de la amplitud de ese array. Se han realizado pruebas con 1 número en el array y con varios. Incluso se probó el bucle modificando el umbral de manera que sea menor que el precio, evaluando así el if dentro del bucle.
En las pruebas de caja blanca se hicieron pruebas siguiendo todos los caminos posibles, modificando el array. En las pruebas de caja negra se llamó al método

desde otra clase directamente probando diferentes casos y en JUnit se realizó un @Test para cada caso tipo.

d) Método actualizarPrecio

```
public static void actualizarPrecio(double[] precios, int indice, double nuevoPrecio) {
        if (precios == null) {
                return;
        }
        if (indice >= 0 && indice < precios.length) {
                if (nuevoPrecio < 0) {
                        precios[indice] = 0;
                } else {
                        precios[indice] = nuevoPrecio;
                }
        }
}
```

Se ha cubierto el if en caso de que el array no exista, sea null.
También se ha cubierto el siguiente if en caso de que el índice sea <0 o que dicho índice sea mayor que la longitud del array. También en los casos en los que lel precio a actualizar sea < 0, 0 y un número válido.
En las pruebas de caja blanca se hicieron pruebas siguiendo todos los caminos posibles, modificando el array. En las pruebas de caja negra se llamó al método desde otra clase directamente probando diferentes casos y en JUnit se realizó un @Test para cada caso tipo.

ENUNCIADO 2:

1.- Descripción:

Se han realizado pruebas unitarias sobre los métodos del programa de reserva de asientos utilizando pruebas de caja blanca, caja negra y JUnit5. Todas las pruebas se han realizado satisfactoriamente sin errores.

2.- Métodos probados:

- contarLibres
- contarOcupados
- estaCompleta
- reservarAsiento
- cancelarReserva

3.- Criterios de cobertura:

a) Método contarLibres

```
public static int contarLibres(boolean[] fila) {
        if (fila == null) {
                return 0;
        }
        int libres = 0;
        for (int i = 0; i < fila.length; i++) {
                if (!fila[i]) {
                        libres++;
                }
        }
        return libres;
}
```

Se ha cubierto el if en caso de que el array de fila sea null (no exista).
También se ha cubierto el bucle for modificando el array de fila, dado que dicho bucle depende de la amplitud de ese array. Se han realizado pruebas con el array vacio, con un valor true, uno false y con múltiples valores.
En las pruebas de caja blanca se hicieron pruebas siguiendo todos los caminos posibles, modificando el array. En las pruebas de caja negra se llamó al método desde otra clase directamente probando diferentes casos y en JUnit se realizó un @Test para cada caso tipo.

b) contarOcupados

```
public static int contarOcupados(boolean[] fila) {
	if (fila == null) {
		return 0;
	}
	int ocupados = 0;
	for (int i = 0; i < fila.length; i++) {
		if (fila[i]) {
			ocupados++;
		}
	}
	return ocupados;
}
```

Se ha cubierto el if en caso de que el array de fila sea null (no exista).
También se ha cubierto el bucle for modificando el array de fila, dado que dicho bucle depende de la amplitud de ese array. Se han realizado pruebas con el array vacio, con un valor true, uno false y con múltiples valores.
En las pruebas de caja blanca se hicieron pruebas siguiendo todos los caminos posibles, modificando el array. En las pruebas de caja negra se llamó al método desde otra clase directamente probando diferentes casos y en JUnit se realizó un @Test para cada caso tipo.

c) estaCompleta

```
public static boolean estaCompleta(boolean[] fila) {
	if (fila == null || fila.length == 0) {
		return false;
	}
	for (int i = 0; i < fila.length; i++) {
		if (!fila[i]) {
			return false;
		}
	}
	return true;
}
```

Se ha cubierto el if en caso de que el array de fila sea null (no exista) o en caso de que esté vacío.
También se ha cubierto el bucle for modificando el array de fila, dado que dicho bucle depende de la amplitud de ese array. Se han realizado pruebas con el array vacio, con un valor true, uno false y con múltiples valores variados y con la fila completa.
En las pruebas de caja blanca se hicieron pruebas siguiendo todos los caminos posibles, modificando el array. En las pruebas de caja negra se llamó al método

desde otra clase directamente probando diferentes casos y en JUnit se realizó un @Test para cada caso tipo.

d) reservarAsiento

```
public static void reservarAsiento(boolean[] fila, int indice) {
        if (fila == null) {
                return;
        }
        if (indice >= 0 && indice < fila.length) {
                if (!fila[indice]) {
                        fila[indice] = true;
                }
        }
}
```

Se ha cubierto el if en caso de que el array de fila sea null (no exista).
También se ha cubierto el resto de if en caso de que el índice sea menor de 0, o sea mayor que la longitud del array. Finalmente se ha probado el if en caso de que el asiento a reservar esté en estado true y en caso de que esté en false.
En las pruebas de caja blanca se hicieron pruebas siguiendo todos los caminos posibles, modificando el array. En las pruebas de caja negra se llamó al método desde otra clase directamente probando diferentes casos y en JUnit se realizó un @Test para cada caso tipo.

e) cancelarReserva

```
public static void cancelarReserva(boolean[] fila, int indice) {
        if (fila == null) {
                return;
        }
        if (indice >= 0 && indice < fila.length) {
                if (fila[indice]) {
                        fila[indice] = false;
                }
        }
}
```

Se ha cubierto el if en caso de que el array de fila sea null (no exista).
También se ha cubierto el resto de if en caso de que el índice sea menor de 0, o sea mayor que la longitud del array. Finalmente se ha probado el if en caso de que el asiento a reservar esté en estado true y en caso de que esté en false.
En las pruebas de caja blanca se hicieron pruebas siguiendo todos los caminos posibles, modificando el array. En las pruebas de caja negra se llamó al método desde otra clase directamente probando diferentes casos y en JUnit se realizó un @Test para cada caso tipo.