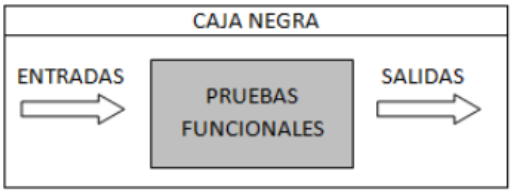
ÍNDICE

1. Introducción

El propósito de las **pruebas de caja negra o funcionales** es comprobar si las salidas que devuelve la aplicación son las esperadas en función de los parámetros de entrada.



Este tipo de prueba, no consideraría, en ningún caso, el código desarrollado, ni el algoritmo, ni la eficiencia, ni si hay partes del código innecesarias, etc. Estos aspectos son comprobados en las pruebas de caja blanca.

Dentro de las pruebas funcionales, podemos indicar los siguientes tipos:

- **Particiones equivalentes**: la idea de este tipo de pruebas funcionales, es considerar el menor número posible de casos de pruebas, para ello, cada caso de prueba tiene que abarcar el mayor número posible de entradas diferentes. Lo que se pretende, es crear un conjunto de clases de equivalencia, donde la prueba de un valor representativo de la misma, en cuanto a la verificación de errores, sería extrapolable al que se conseguiría probando cualquier valor de la clase.

- **Análisis de valores límite**: en este caso, a la hora de implementar un caso de prueba, se van a elegir como valores de entrada aquellos que se encuentra en el límite de las clases de equivalencia.

- **Pruebas aleatorias**: consiste en generar entradas aleatorias para la aplicación que hay que probar. Se suelen utilizar generadores de prueba, que son capaces de crear un volumen de casos de prueba al azar, con los que será alimentada la aplicación.

- **Conjetura de errores**: trata de generar casos de prueba que la experiencia ha demostrado generan típicamente errores. En valores numéricos, un buen ejemplo es comprobar si funciona correctamente con el valor 0, ya que si es utilizado como denominador en alguna división podría generar un error en nuestro programa.

2. Caso práctico de Caja Negra

La técnica para determinar los casos de prueba de caja negra se realiza completando los siguientes pasos:

1. Determinar las clases de equivalencia.
2. Determinar un análisis de valores límite.
3. Conjetura de errores.
4. Generar los casos de uso necesarios para probar las clases validas y no validas.
5. Establecer los datos de entrada y los resultados esperados.
6. Lanzar una ejecución del programa por cada caso de uso y comparar los resultados obtenidos con los esperados para determinar la corrección del código.

2.1. Determinar las clases de equivalencia

La técnica de clases de equivalencia es un tipo de **prueba funcional**, donde en cada caso de prueba se agrupa el mayor número de entradas posibles. A partir de aquí, se asume que la prueba de un valor representativo de cada clase, permite suponer que el resultado que se obtiene con él, será el mismo que con cualquier otro valor de la clase.

Los pasos a seguir para identificar las clases de equivalencia son:

1. Identificar las condiciones de las entradas del programa, es decir, restricciones de formato o contenido de los datos de entrada.
2. A partir de ellas, identificar clases de equivalencia que pueden ser:
   - De datos válidos.
   - De datos no válidos o erróneos.

Existen algunas **reglas heurísticas** que ayudan a identificar las clases:

Tipo de dato	Ejemplo	Clases equivalencia
Rango de valores de entrada. Crear una clase válida y dos clases no válidas.	La edad de acceso a un evento está comprendida entre 18 y 100 años.	**Clase válida:**
		Valor entre 18 - 100
		Clases no válidas:
		Menor de 18. Mayor de 100.

Número finito y consecutivo de valores. Creará una clase válida y dos no válidas.	Una encuesta puede ser valorada con los valores 0, 1, 2, 3.	**Clase válida:**
		Cualquiera de los valores 0,1,2,3
		Clases no válidas:
		Menor de 0. Mayor de 3.
Condición **verdadero/falso**.	Una persona tiene la condición de ser mayor de edad.	**Clase válida:**
		Edad >=18
		Clases no válidas:
		Edad<18
Conjunto de valores admitidos. Se identifica una clase válida por cada valor y una no válida.	Una opción de menú puede aceptar los valores 'A' para altas, 'B' para bajas y 'S' para salir del programa.	**Clases validas:**
		Opción 'A' Opción 'B' Opción 'S'
		Clase no válida:
		Opción 'J'

En cualquier caso, si se sospecha que ciertos elementos de una clase no se tratan igual que el resto de la misma, deben dividirse en clases menores.

2.2. Análisis de los valores límite.

La experiencia indica que los casos de prueba que exploran las condiciones límite de un programa producen un mejor resultado para detectar defectos.

El AVL (Análisis de valores límite) es una técnica de diseño de casos de prueba que complementa a la de particiones de equivalencia.

La principal diferencia se encuentra en el tratamiento que tienen las clases de equivalencia de rango de valores y de número finito y consecutivo de valores. Ahora la prueba se realizará sobre los valores límite de los rangos.

Ejemplo	Clase de equivalencia	Valores límite
La edad de acceso a un evento está comprendida entre 18 y 100 años.	**Clase valida:**	**Clases válidas:**
	Valor entre 18 - 100. Caso único. P.e 30	Caso 1: 18 Caso 2: 100
	Clases no válidas:	**Clases no válidas:**
	Menor de 18. P.e 15 Mayor de 100. P.e. 110	Menor de 18. 17 Mayor de 100. 101
Una encuesta puede ser valorada con los valores 0, 1, 2, 3.	**Clase valida:**	**Clases válidas:**
	Cualquier de los valores 0,1,2,3 Caso único. P.e 2	Caso 1: 0 Caso 2: 3
	Clases no válidas:	**Clases no válidas:**
	Menor de 0. P.e -10 Mayor de 3. P.e 7	Valor -1 Valor 4

En las pruebas AVL también habría que generar casos de prueba atendiendo a clases de equivalencia de los datos de salida.

2.3. Conjetura de errores.

La experiencia en la fase de pruebas indica que existen ciertos valores de entrada que típicamente son generadores de errores, y qué en ocasiones, pasan desapercibidos en las técnicas de clases de equivalencia y de valores límite.

La conjetura de errores considera esos datos y define nuevos casos de prueba a los que someter a los programas. Se trata de una técnica menos metódica que las anteriores, tiene mucho más que ver con la intuición y experiencia del programador.

Una prueba típica en conjetura de errores es probar el valor de entrada 0 para datos numéricos por si pudiera participar como denominador en alguna división durante la ejecución del programa.

2.4. Casos de uso, resultados esperados y análisis.

Ahora toca definir los datos de entrada al programa. Al conjunto de entradas al programa utilizados para cada ejecución se le denomina caso de uso. Los casos de uso se generarán a partir de las clases de equivalencia, valores límite y conjeturas de errores obtenidos en los apartados anteriores.

Este proceso consta de las siguientes fases:

1. Numerar las clases de equivalencia.

2. Crear casos de uso que cubran todas las clases de equivalencia válidas. Se intentará agrupar en cada caso de uso tantas clases de equivalencia como sea posible.

3. Crear un caso de uso para cada clase de equivalencia no válida.

Además, toca definir los resultados previstos en cada ejecución. Cuando posteriormente se lance la ejecución del programa para cada caso de uso, los resultados obtenidos serán comparados con los esperados y así determinar la corrección del código.

3. Ejemplo Fibonacci

El siguiente programa java hace el cálculo de la serie fibonacci y muestra el resultado por pantalla. El número de dígitos de la serie mostrados serán introducidos por teclado y como máximo será de cinco dígitos.

El programa pedirá el nombre del usuario, debiendo tener un formato de entre 2 y 7 caracteres alfabéticos-no numéricos. Una vez calculada la serie para el número de dígitos solicitada, se mostrará por pantalla el mensaje: "nombre_cliente: serie_calculada".

El programa se ejecutará mientras no se introduzca por teclado una "S" o una "Q".

Nota: el código que se propone en el siguiente apartado no cumple con todos los requisitos planteados en este enunciado, será labor de las pruebas de caja negra detectar sus errores.

3.1. Código.

```
1  public class CalFibonacci {
2      public static void main(String [] args) {
3          CalFibonacci misCal = new CalFibonacci();
4          misCal.Fibonacci();
5      }
6      public void Fibonacci() {
7          Scanner miScan = new Scanner(System.in);
8          String sSalir=miScan.nextLine();
9          int iValor = 0;
10         String sResultado;
11         String sAux;
12         while(!(sSalir.equals("S") || sSalir.equals("s")))
13         {
14             System.out.println("¿Cuantos numeros de la serie deseas mostrar?");
15             sAux =  miScan.nextLine();
16             iValor = Integer.parseInt(sAux);
17             switch(iValor)
18             {
19                 case 3:
20                     sResultado = " 1";
21                 case 2:
22                     sResultado = " 1" + sResultado;
23                 case 1:
24                     sResultado = " 0" + sResultado;
25             }
26             System.out.println("Los " +  iValor + " numeros son: " +  sResultado);
27             System.out.println("Si deseas salir, pulsa: S o s");
28             sSalir = miScan.nextLine();
29         }
30     }
31 }
```

Nota: la serie de Fibonacci, comienza por el cero, sigue por el uno, y los siguientes números se van calculando como la suma de los dos anteriores, es decir: 0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55.

3.2. Clases de equivalencia.

Condición de entrada	Clases válidas		Clases no válidas	
Nombre	(1)	2 <= caracteres <= 7	(2)	caracteres < 2
			(3)	caracteres > 7
			(4)	Incluye números
Nº dígitos solicitados	(5)	1 <= num <=5	(6)	num < 1
			(7)	num > 5
			(8)	num una letra
Orden de finalizar el programa	(9)	Pulsar "S"	(11)	Pulsar "N"
	(10)	Pulsar "Q"		

3.3. Análisis de valores límite (AVL).

Los valores límite en este ejercicio son:

1. Para la información del nombre.
   - **Caso de prueba 1**. Determina que el nombre tendrá un número de caracteres entre 2 y 7. 4 por ejemplo. Al considerar AVL se divide en dos casos de prueba:

     (1.1) - nombre con 2 caracteres.

     (1.2) - nombre con 7 caracteres.

   - **Caso de prueba 2**. El número de caracteres del nombre deberá ser menor de 2 (1 o ninguno). Al considerar AVL se fija el valor a 1.
   - **Caso de prueba 3**. El número de caracteres del nombre deberá ser mayor de 7 (10 por ejemplo). Al considerar AVL el número de caracteres se ja a 8.

2. Para la información del número de dígitos de la serie.
   - **Caso de prueba 5**. El número de dígitos a solicitar está comprendido en el intervalo de 1 y 5 (3 por ejemplo). Al considerar AVL se divide en dos casos de prueba:

     (5.1) - 1

     (5.2) - 5

   - **Caso de prueba 6**. El número de dígitos a solicitar será menor de 1 (-3 por ejemplo). Al considerar AVL se ja el valor a 0.
   - **Caso de prueba 7**. El número de dígitos a solicitar será mayor de 5 (10 por ejemplo). Al considerar AVL el número de caracteres se ja a 6.

3.4. Conjetura de errores.

Dos posibles casos de pruebas en este código de acuerdo a la conjetura de errores podrían ser:

- Crear un caso de prueba 1.3 que considere en el nombre un espacio, por ejemplo, Noa Mor.
- En el caso de prueba (6) ya estamos considerando el valor 0 para el número de dígitos, éste podría también haber sido considerado como criterio para un caso de uso de la conjetura de errores.

3.5. Casos de uso, datos de entrada y resultados esperados.

Casos	Nombre	Nº Dígitos	Finalización	Resultado esperado
1.1;5.1;11	Va	2	<> S y Q	Va: 0,1. No finaliza el programa
1.2;5.2	Valeria	5	<> S y Q	Valeria: 0,1,1,2,3. No finaliza el programa
1.3	Vale M	2	<> S y Q	Vale M: 0,1. No finaliza el programa
9	Datos válidos		S	Finaliza el programa
10	Datos válidos		Q	Finaliza el programa
2	J	2	<> S y Q	Solicita el nombre de nuevo
3	Valeriaa	2	<> S y Q	Solicita el nombre de nuevo
4	Val3ria	2	<> S y Q	Solicita el nombre de nuevo
6	Valeria	1	<> S y Q	Solicita el número de dígitos de nuevo
7	Valeria	6	<> S y Q	Solicita el número de dígitos de nuevo
8	Valeria	P	<> S y Q	Solicita el número de dígitos de nuevo

Las pruebas de caja negra nos permiten detectar los siguientes errores en el código:

1. No existen validaciones en el formato y número de dígitos en el nombre.
2. No hay un control en el número máximo de dígitos de la serie Fibonacci. Además, al solicitar 4 o 5 dígitos el programa no funciona correctamente.
3. La opción de menú 'Q' no finaliza el programa.