Universidad Carlos III
Curso Desarrollo de Software 2024-25
Ejercicio Guiado 2

Desarrollo dirigido por pruebas

Grupo: **80**
Titulación: **Ingeniería Informática**
Grupo trabajo: **06**
Alumnos:

Alejandra de los Santos 100495843
100495843@alumnos.uc3m.es

Ana Grima Vázquez de Prada 100495785
100495785@alumnos.uc3m.es

Índice

1. Definición de los casos de prueba

Para realizar todas las funciones y comprobar que los resultados son válidos, también creamos las funciones: “validate_iban”, “validate_amount”, “validate_date”, “validate_type” y “validate_concept”. De esta forma nos aseguramos que se cumplen con los requisitos especificados para cada parámetro.

1.1. Función 1: Solicitar transferencia

En esta función se registran las solicitudes de transferencias entre dos cuentas bancarias. Esto se hace recibiendo un fichero JSON y comprobando si los datos dentro de él son válidos. Si son válidos, se devuelve un código correspondiente el Transfer Code y se almacenan las transferencias en un fichero JSON.

Identificamos las clases de equivalencia y valores límite de cada variable y de las diferentes salidas posibles.

from_iban

Clases de equivalencia válidas:

- CEV1 - string
- CEV2 - iban válido
- CEV9 - longitud 24
- CEV3 - dos primeros char son ES
- CEV28 - los 22 caracteres que siguen a ES son números

Clases de equivalencia inválidas:

- CENV1 - distinto de string
- CENV2 - iban no cumple algoritmo de validación (24 char)
- CENV3 - from iban < 24
- CENV4 - from iban > 24
- CENV5 - 2 primeros caracteres no son ES
- CENV13 - los caracteres que le siguen no son números

Valores límites válidos:

- VLV1 - longitud = 24

Valores límites inválidos:

- VLNV1 - 23 char
- VLNV2 - 25 char

to_iban

Clases de equivalencia válidas:

- CEV4 - string

- CEV5 - iban válido
- CEV29 - longitud 24
- CEV6- Dos primeros char son ES
- CEV30 - los 22 caracteres que siguen a ES son números
- CEV31 - from_iban y to_iban son diferentes

Clases de equivalencia inválidas:

- CENV6 - distinto de string
- CENV7 - iban no cumple algoritmo de validación
- CENV8 - from iban < 24
- CENV9 - from iban > 24
- CENV10 - 2 primeros caracteres no son ES
- CENV39 - los caracteres que le siguen no son números
- CENV40 - from_iban y to_iban no son diferentes

Valores límites válidos:

- VLV2 - longitud = 24

Valores límites inválidos:

- VLNV3 - 23 char
- VLNV 4 - 25 char

concept

Clases de equivalencia válidas:

- CEV7 - string
- CEV8 - concept válido (al menos dos cadenas separadas por espacio)
- CEV10 - caracteres [a-z A-Z]
- CEV11 - 10 <= concept <= 30

Clases de equivalencia inválidas:

- CENV11 - distinto de string
- CENV12 - concept no válido (menos de dos cadenas)
- CENV14- no caracteres [a-z A-Z]
- CENV15 - concept < 10
- CENV16 - concept > 30

Valores límites válidos:

- VLV3 - 2 palabras
- VLV4 - 3 palabras
- VLV5 - 10
- VLV6 - 11
- VLV7 - 30
- VLV8 - 29

Valores límites inválidos:

- VLNV5 - 1 palabra
- VLNV6 - 9
- VLNV7 - 31

type

Clases de equivalencia válidas:

- CEV12 - string
- CEV13 - "ORDINARY"
- CEV14 - "URGENT"
- CEV15 - "INMEDIATE"

Clases de equivalencia inválidas:

- CENV17 - distinto de string
- CENV18 - cualquier otro valor

Esta variable no tiene valores límites.

date

Clases de equivalencia válidas:

- CEV16 - "DD/MM/YY"
- CEV17 - 01 <= DD <= 31
- CEV18 - 01 <= MM <= 12
- CEV19 - 2025 <= YY <= 2050
- CEV20 - fecha generada correcta
- CEV21 - date >= fecha recibo orden
- CEV32 - longitud = 10

Clases de equivalencia inválidas:

- CENV19 - cualquier otro formato no string
- CENV22 - MM < 01
- CENV23 - MM> 12
- CENV24 - YY < 2025
- CENV25 - YY > 2050
- CENV26 - fecha generada incorrecta
- CENV27 - date < fecha recibo orden
- CENV41 - longitud < 10
- CENV42 - longitud > 10

Valores límites válidos:

- VLV9 - 01
- VLV10 - 02
- VLV11 - 31

- VLV12 - 30
- VLV13 - 01
- VLV14 - 02
- VLV15 - 12
- VLV16 - 11
- VLV17 - 2025
- VLV18 - 2026
- VLV19- 2050
- VLV20 - 2049
- VLV27 - 10

Valores límites inválidos:

- VLNV8 - 00
- VLNV9 - 32
- VLNV10 - 00
- VLNV11 - 13
- VLNV12 - 2024
- VLNV13 - 2051
- VLNV17 - 9
- VLNV18 - 11

amount

Clases de equivalencia válidas:

- CEV22 - float
- CEV23 - 10,00 <= amount <= 10000,00
- CEV24 - decimales <= 2

Clases de equivalencia inválidas:

- CENV28 - distinto de float
- CENV29 - amount < 10,00
- CENV30 - amount > 10000,00
- CENV31 - decimales > 2

Valores límites válidos:

- VLV21 - 10,00
- VLV22 - 10,01
- VLV23 - 10000,00
- VLV24 - 9999,99
- VLV25 - 2 decimales
- VLV26 - 1 decimal

Valores límites inválidos:

- VLNV14 - 9,99
- VLNV15 - 10000,01
- VLNV16 - 3 decimales

salidas

Clases de equivalencia válidas:

- CEV25 - cadena en MD5 correspondiente al Transfer Code
- CEV26 - fichero con los datos de la transferencia (se incluyen también las anteriores)

Clases de equivalencia inválidas:

- CENV32 - excepcion: número de cuenta no válido
- CENV33 - exception: el concepto no tiene un valor válido
- CENV34 - exception: el tipo de transferencia no es válido
- CENV35 - excepcion: la fecha de la transferencia no es válida
- CENV36 - excepcion: la cantidad no es válida
- CENV37 - excepcion: existe en el fichero de salida una transferencia con los mismos datos

Las salidas no tienen valores límites.

Casos de prueba

En los casos de prueba inválidos de esta función, vamos mirando los CEVN y VLNV. Los valores límites que estén dentro de los rangos de las clases de equivalencia los comprobamos en un mismo test. Para el resto hacemos un test por CENV o VLNV.

Hemos añadido los casos de prueba ya que en el excel no se especifica el nombre de la función que realiza cada uno.

TC1, TC2, TC3, TC4, TC5: Hay un test válido por cada caso válido que hay en el excel.

TC6 (test3_f1_not_valid) from: Iban no es string

- from_iban = 912514650100971756460833
- to_iban = "iban válido"
- concept = "esto es una prueba"
- type = "URGENT"
- date = "12/04/2025"
- amount = 250.14
- result: “Exception: from_iban no es válido”

TC7 (test7_f1_not_valid) from: Algoritmo no válido

- from_iban = "ES9121000418450200051333"
- to_iban = "iban válido"
- concept = "esto es una prueba"
- type = "URGENT"
- date = "12/04/2025"
- amount = 250.14
- result: “Exception: from_iban no es válido”

TC8 (test4_f1_not_valid) from: Iban con una longitud de 23

- from_iban = "ES9121000418450200051333"
- to_iban = "iban válido"
- concept = "esto es una prueba"
- type = "URGENT"
- date = "12/04/2025"
- amount = 250.14
- result: “Exception: from_iban no es válido”

TC9 (test5_f1_not_valid) from: Iban con una longitud de 25

- from_iban = "ES9121000418450200051332233"
- to_iban = "iban válido"
- concept = "esto es una prueba"
- type = "URGENT"
- date = "12/04/2025"
- amount = 250.14
- result: “Exception: from_iban no es válido”

TC10 (test2_f1_not_valid) from: Probar que en vez de ES sea DE

- from_iban = "DE9121000418450200051332"
- to_iban = "iban válido"
- concept = "esto es una prueba"
- type = "URGENT"
- date = "12/04/2025"
- amount = 250.14
- result: “Exception: from_iban no es válido”

TC11 (test6_f1_not_valid) from: Después de ES no hay 22 números

- from_iban = "ESDHFIEMANF"
- to_iban = "iban válido"
- concept = "esto es una prueba"
- type = "URGENT"
- date = "12/04/2025"
- amount = 250.14
- result: “Exception: from_iban no es válido”

TC12 (test8_f1_not_valid) to: Iban no es string

- from_iban = "ES9121000418450200051332"
- to_iban = 912514650100971756460833
- concept = "esto es una prueba"
- type = "URGENT"
- date = "12/04/2025"
- amount = 250.14
- result: “Exception: to_iban no es válido”

TC13 (test9_f1_not_valid) to: Algoritmo no válido

- from_iban = "ES9121000418450200051332"
- to_iban = "ES91210004184502000513333"
- concept = "esto es una prueba"
- type = "URGENT"
- date = "12/04/2025"
- amount = 250.14
- result: “Exception: to_iban no es válido”

TC14 (test10_f1_not_valid) to: Iban con una longitud de 25

- from_iban = "ES9121000418450200051332"
- to_iban = "ES912100041845020005133222"
- concept = "esto es una prueba"
- type = "URGENT"
- date = "12/04/2025"
- amount = 250.14
- result: “Exception: to_iban no es válido”

TC15 (test11_f1_not_valid) to: Iban con una longitud de 23

- from_iban = "ES9121000418450200051332"
- to_iban = "ES91210004184502000513"
- concept = "esto es una prueba"
- type = "URGENT"
- date = "12/04/2025"
- amount = 250.14
- result: “Exception: to_iban no es válido”

TC16 (test12_f1_not_valid) to: Probar que en vez de ES sea DE

- from_iban = "ES9121000418450200051332"
- to_iban = "DE9121000418450200051332"
- concept = "esto es una prueba"
- type = "URGENT"
- date = "12/04/2025"
- amount = 250.14
- result: “Exception: to_iban no es válido”

TC17 (test13_f1_not_valid) to: Despues de ES no hay 22 números

- from_iban = "ES9121000418450200051332"
- to_iban = "ESDHFIEMANF"
- concept = "esto es una prueba"
- type = "URGENT"
- date = "12/04/2025"
- amount = 250.14
- result: “Exception: to_iban no es válido”

TC18 (test14_f1_not_valid) to: from_iban y to_iban son iguales, lo tomamos como caso invalido ya que en el enunciado se dice que es entre dos cuentas bancarias.

- from_iban = "ES9121000418450200051332"
- to_iban = "ES9121000418450200051332"
- concept = "esto es una prueba"
- type = "URGENT"
- date = "12/04/2025"
- amount = 250.14
- result: “Exception: from_iban y to_iban son iguales”

TC19 (test15_f1_not_valid) concept: No es string

- from_iban = "ES9121000418450200051332"
- to_iban = "ES8658342044541216872704"
- concept = 112345
- type = "URGENT"
- date = "12/04/2025"
- amount = 250.14
- result: "Exception: el concepto no tiene un valor válido"

TC20 (test16_f1_not_valid) concept: solo una cadena

- from_iban = "ES9121000418450200051332"
- to_iban = "ES8658342044541216872704"
- concept = "cadena"
- type = "URGENT"
- date = "12/04/2025"
- amount = 250.14
- result: "Exception: el concepto no tiene un valor válido"

TC21 (test17_f1_not_valid) concept: string de números

- from_iban = "ES9121000418450200051332"
- to_iban = "ES8658342044541216872704"
- concept = "1234 45653"
- type = "URGENT"
- date = "12/04/2025"
- amount = 250.14
- result: "Exception: el concepto no tiene un valor válido"

TC22 (test18_f1_not_valid) concept: dos cadenas de longitud 9

- from_iban = "ES9121000418450200051332"
- to_iban = "ES8658342044541216872704"
- concept = "cadena de"
- type = "URGENT"
- date = "12/04/2025"
- amount = 250.14
- result: "Exception: el concepto no tiene un valor válido"

TC23 (test19_f1_not_valid) concept: dos cadenas de longitud 31

- from_iban = "ES9121000418450200051332"

- to_iban = "ES9121000418450200051332"
- concept = "cadena de longitud mayor que 31 caracteres"
- type = "URGENT"
- date = "12/04/2025"
- amount = 250.14
- result: "Exception: el concepto no tiene un valor válido"

TC24 (test20_f1_not_valid) Type: no es str

- from_iban = "ES9121000418450200051332"
- to_iban = "ES8658342044541216872704"
- concept = "esto es una prueba"
- type = 1234
- date = "12/04/2025"
- amount = 250.14
- result: "Exception: el tipo de transferencia no es válido"

TC25 (test21_f1_not_valid) Type: str no válido

- from_iban = "ES9121000418450200051332"
- to_iban = "ES8658342044541216872704"
- concept = "esto es una prueba"
- type = "hola"
- date = "12/04/2025"
- amount = 250.14
- result: "Exception: el tipo de transferencia no es válido"

TC26 (test22_f1_not_valid) date: no es str

- from_iban = "ES9121000418450200051332"
- to_iban = "ES8658342044541216872704"
- concept = "esto es una prueba"
- type = "URGENT"
- date = 12/11/2025
- amount = 250.14
- result: "Exception: la fecha de transferencia no es válida"

TC27 (test23_f1_not_valid) date: DD < 01

- from_iban = "ES9121000418450200051332"
- to_iban = "ES8658342044541216872704"
- concept = "esto es una prueba"
- type = "URGENT"
- date = "00/01/2025"
- amount = 250.14
- result: "Exception: la fecha de transferencia no es válida"

TC28 (test24_f1_not_valid) date: DD >31

- from_iban = "ES9121000418450200051332"
- to_iban = "ES8658342044541216872704"

- concept = "esto es una prueba"
- type = "URGENT"
- date = "32/01/2025"
- amount = 250.14
- result: "Exception: la fecha de transferencia no es válida"

TC29 (test25_f1_not_valid) date: < fecha recibo orden

- from_iban = "ES9121000418450200051332"
- to_iban = "ES8658342044541216872704"
- concept = "esto es una prueba"
- type = "URGENT"
- date = "01/01/2025"
- amount = 250.14
- result: "Exception: la fecha de transferencia no es válida"

TC30 (test28_f1_not_valid) date: no válida

- from_iban = "ES9121000418450200051332"
- to_iban = "ES8658342044541216872704"
- concept = "esto es una prueba"
- type = "URGENT"
- date = "31/02/2025"
- amount = 250.14
- result: "Exception: la fecha de transferencia no es válida"

TC31 (test26_f1_not_valid) date: longitud 9

- from_iban = "ES9121000418450200051332"
- to_iban = "ES8658342044541216872704"
- concept = "esto es una prueba"
- type = "URGENT"
- date = "1/01/2025"
- amount = 250.14
- result: "Exception: la fecha de transferencia no es válida"

TC32 (test27_f1_not_valid) date: longitud 11

- from_iban = "ES9121000418450200051332"
- to_iban = "ES8658342044541216872704"
- concept = "esto es una prueba"
- type = "URGENT"
- date = "001/01/2025"
- amount = 250.14
- result: "Exception: la fecha de transferencia no es válida"

TC33 (test29_f1_not_valid) amount: no es float

- from_iban = "ES9121000418450200051332"
- to_iban = "ES8658342044541216872704"
- concept = "esto es una prueba"

- type = "URGENT"
- date = "12/04/2025"
- amount = 1
- result: "Exception: la cantidad no es válida"

TC34 (test30_f1_not_valid) amount: < 10,00

- from_iban = "ES9121000418450200051332"
- to_iban = "ES8658342044541216872704"
- concept = "esto es una prueba"
- type = "URGENT"
- date = "12/04/2025"
- amount = 9.99
- result: "Exception: la cantidad no es válida"

TC35 (test31_f1_not_valid) amount: > 10000,00

- from_iban = "ES9121000418450200051332"
- to_iban = "ES8658342044541216872704"
- concept = "esto es una prueba"
- type = "URGENT"
- date = "12/04/2025"
- amount = 10000.01
- result: "Exception: la cantidad no es válida"

TC36 (test32_f1_not_valid) amount: tres decimales

- from_iban = "ES9121000418450200051332"
- to_iban = "ES8658342044541216872704"
- concept = "esto es una prueba"
- type = "URGENT"
- date = "12/04/2025"
- amount = 9.999
- result: "Exception: la cantidad no es válida"

TC37 (test33_f1_not_valid) todas las variables son válidas, pero el fichero ya existe

- from_iban = "ES9121000418450200051332"
- to_iban = "ES6000491500051234567892"
- concept = "esto es una prueba"
- type = "URGENT"
- date = "12/04/2025"
- amount = 200.34
- result: "Exception: existe en el archivo de salida una transferencia con los mismos datos"

TC38 (test34_f1_not_valid) date: MM < 01

- from_iban = "ES9121000418450200051332"
- to_iban = "ES8658342044541216872704"
- concept = "esto es una prueba"

- type = "URGENT"
- date = "01/00/2025"
- amount = 250.14
- result: "Exception: la fecha de transferencia no es válida"

TC39 (test35_f1_not_valid) date: MM > 12

- from_iban = "ES9121000418450200051332"
- to_iban = "ES8658342044541216872704"
- concept = "esto es una prueba"
- type = "URGENT"
- date = "01/13/2025"
- amount = 250.14
- result: "Exception: la fecha de transferencia no es válida"

TC40 (test36_f1_not_valid) date: YY < 2025

- from_iban = "ES9121000418450200051332"
- to_iban = "ES8658342044541216872704"
- concept = "esto es una prueba"
- type = "URGENT"
- date = "01/01/2024"
- amount = 250.14
- result: "Exception: la fecha de transferencia no es válida"

TC41 (test37_f1_not_valid) date: YY > 2051

- from_iban = "ES9121000418450200051332"
- to_iban = "ES8658342044541216872704"
- concept = "esto es una prueba"
- type = "URGENT"
- date = "01/01/2051"
- amount = 250.14
- result: "Exception: la fecha de transferencia no es válida"

1.2. Función 2: Ingreso en cuenta

Esta función tiene como objetivo procesar depósitos desde un archivo JSON, validando su estructura, generando una firma SHA-256 e introduciendo los datos correspondientes en el archivo “deposit_store.json”. Para esta función, creamos unos casos de prueba siguiendo un análisis sintáctico;

Gramática

ficheroJson ::= <inicioObjeto><deposit><finObjeto>

inicioObjeto ::= {

finObjeto ::= }

deposit ::= <iban_field> , <amount_field>

iban_field ::= "IBAN" : "<iban_value>"

iban_value ::= ES<iban_num>

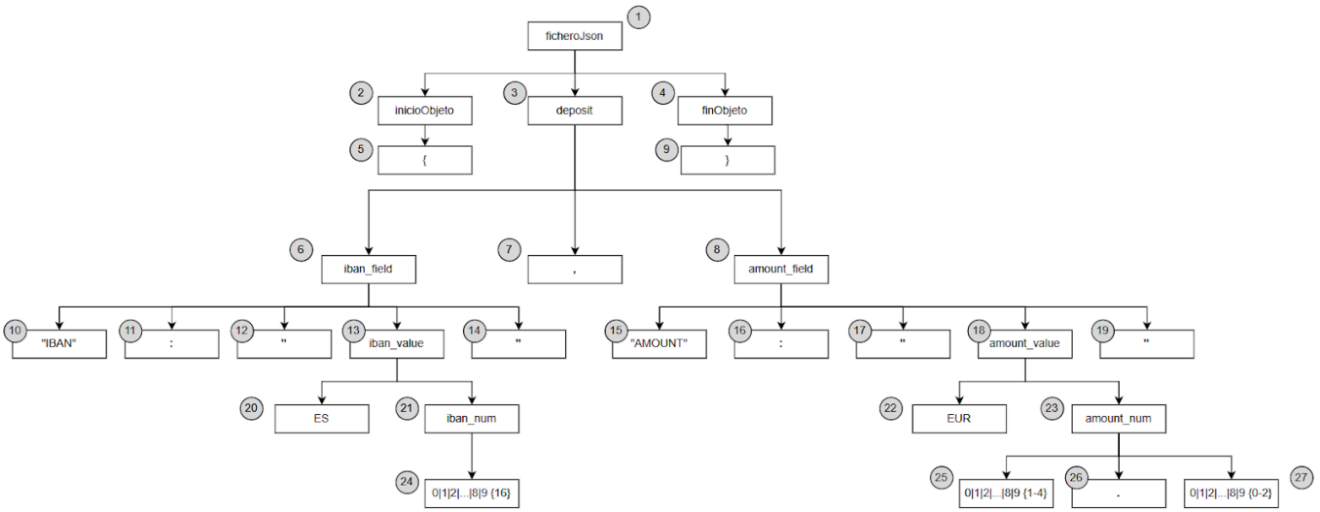
iban_num ::== 0|1|2|...|8|9 {16}

amount_field::= "AMOUNT" : "<amount_value>"

amount_value ::= EUR <amount_num>

amount_num ::= 0|1|2|...|8|9 {1-4} . 0|1|2|...|8|9 {0-2}"

Árbol de derivación



No terminales: 1, 2, 3, 4, 6, 8, 13,18, 21, 23

Terminales: 5, 7, 9, 10, 11, 12, 14, 15, 16, 17, 19, 20, 22, 24, 25, 26, 27

Casos de pruebas

Definimos las pruebas más relevantes centrándonos en entradas y salidas. Separamos los nodos terminales y los nodos no terminales para hacer varias pruebas en cada caso según corresponda, nodos no terminales con casos de eliminación y duplicación y los nodos terminales con casos de modificación, además de los dos mencionados anteriormente. Si alguna de las tres pruebas de los nodos terminales ya ha sido probada gracias a una prueba de un no terminal que lo contiene, entonces "juntamos" ambas pruebas como se menciona en el excel. Para las tres pruebas siempre seguimos los mismos pasos, eliminación; eliminamos el nodo, duplicación; duplicamos el nodo y modificación, lo modificamos de cualquier forma, buscando verificar si entonces sería correcto o no.

La separación de nodos es sencilla, siguiendo el ejemplo de nuestra gramática. Creemos que es importante destacar que los nodos 10 y 15 no se separaron en tres nodos (comillas iniciales, finales y clave separadas) ya que estos debían de ser siempre iguales, cualquier modificación conlleva a un error. Si es cierto que dependiendo de la modificación de estos, sería un error u otro según nuestro código, la diferencia de errores se puede comprobar entre el test 21 y el 67 para la modificación de "IBAN" solamente modificando las comillas entonces sería un error de formato (21), mientras que si se modifica el interior sería error de clave. Lo mismo sucede con el test 35, cuya relación estaría con el test 68 para la modificación de "AMOUNT".

Hablando de test adicionales que no se muestran en el excel (test 67 y 68), añadimos en nuestro código una sección para ellos, ya que son test que igual no llegan a ser necesarios pero vemos oportunos mencionarlos y comprobarlos, esta sección se encuentra a partir del test 66, en donde se encuentra también el test 69 para comprobar que, quitando el punto que separa la parte entera y decimal de amount, si este sigue siendo menor a la cantidad máxima establecida, entonces es válido.

En esta función, el nombre del fichero json es el mismo que el de la función que ejecuta el test. Esto está reflejado en el excel en la columna "FILE PATH".

En total contamos con 69 test que verifican el correcto funcionamiento de la función "deposit_into_account". Para los test en los que se espera una salida válida, primero comprobamos que el archivo de entrada contiene los datos requeridos. Luego, inicializamos un objeto "my_manager" de la clase "AccountManager()" y ejecutamos la función pasándole el archivo como parámetro. Finalmente, verificamos que devuelve la firma, generada con un hash, y que los datos son insertados correctamente en el archivo "deposit_store.json", con la estructura definida en el enunciado. Para ello, establecemos en los tests los valores esperados, recorremos "deposit_store.json" y, si los encontramos, significa que se ha añadido correctamente.

En el caso de los tests en los que se espera un mensaje de error, primero guardamos el contenido original de "deposit_store", para después verificar que no han habido cambios en el archivo, ya que, al producirse un error, no debería insertar ningún dato. Luego, inicializamos un objeto de la clase correspondiente y ejecutamos la función. Finalmente, igualamos el mensaje de error recibido con el mensaje de error esperado, si coincide, el test se ha ejecutado correctamente, validando así el caso no válido.

Cabe mencionar, que hay muchos errores de formato JSON ya que cualquier modificación en la estructura del archivo puede generar un fallo. Por ejemplo, si falta alguna de las comillas de los campos, si falta una coma para separar elementos, si hay una coma antes del primer elemento, o una coma después del último elemento, entonces, el formato se considera inválido.

1.3. Función 3: Calcular saldo

La tercera y última función tiene como objetivo calcular el saldo acumulado de una cuenta bancaria específica, identificada por un IBAN (el input en este caso), a partir del archivo “transactions.json” que contiene todas las transacciones asociadas a dicha cuenta. El proceso valida el IBAN, busca y suma todas las transacciones asociadas a ese IBAN en el archivo y devuelve True en caso de éxito. Además, escribe con la estructura especificada en el enunciado en el archivo “balance_store.json”, donde se almacenan todos los datos de balances de cada cuenta.

Para esta función seguimos un caso de pruebas estructurales, incluimos el gráfico de control de flujo, la definición de las rutas básicas y los casos adicionales necesarios para probar los bucles;

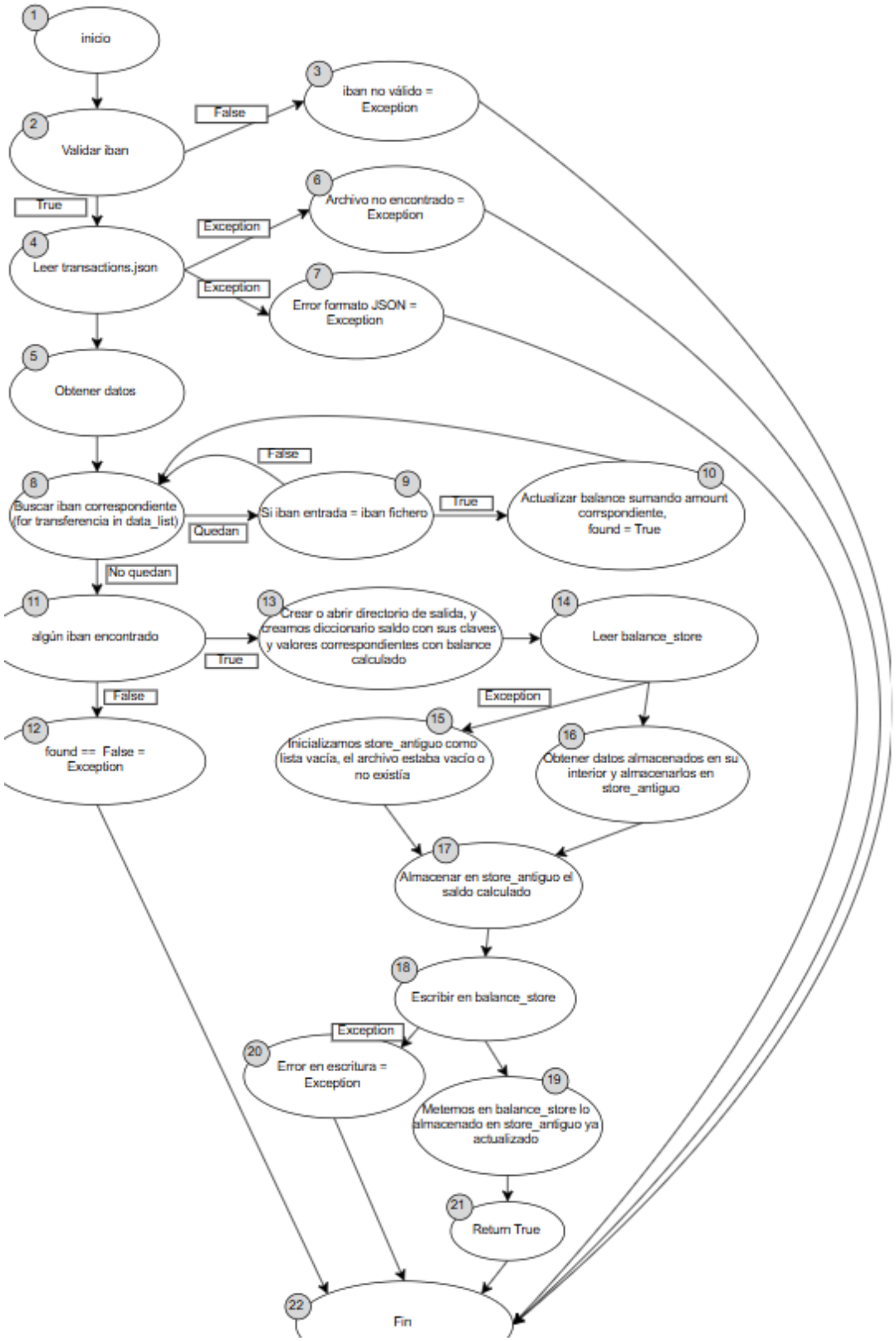
Pruebas adicionales de bucle

1. No entra, el fichero está vacío o no existe.
2. Entra max veces, transactions contiene un número x de transferencias y:

   2.1 Ninguna coincide

   2.2 Algunas coinciden pero no todas

Gráfico de control de flujo



Casos de pruebas

Para la definición de las pruebas, tuvimos en cuenta el archivo proporcionado en aula global “transactions.json”, ajustándonos a sus datos. No vimos necesario realizar la prueba de bucle cuando solo entra una vez o dos ya que el archivo contenía un número determinado de transacciones, donde no todas eran iguales, hay 4 IBANs distintos para ser exactos cada uno con distinto número de transacciones registradas.

Además, consideramos el método “validate_iban” al que se hace referencia dentro de “calculate_balance” como ya probado, por lo que no presentamos el diagrama de flujo ni sus casos de prueba específicos.

Calculamos el número de caminos con la complejidad de McCabe, lo que nos proporciona una estimación para asegurar cuántos casos de prueba se deben contemplar como máximo para garantizar una cobertura de decisiones.

Enlaces: 29

Nodos: 22

V(G) = Enlaces - Nodos + 2 = 29 - 22 + 2 = 9, en total sumando también las pruebas de bucle = 12, en nuestro código contamos con 9 test implementados.

Primero establecimos un camino de referencia, el que sería nuestro camino principal, o típico. Siendo este el camino que sigue los nodos: 1-2-4-5-8-9-10-8-11-13-14-16-17-18-19-21-22 pasando por el bucle tantas veces como transacciones haya en el archivo, y además probando el caso de prueba 2.2 del bucle, donde el IBAN coincide x número de veces. En base a este, establecemos los demás caminos, desviandonos hacia el segundo camino en el segundo nodo, dando como resultado una excepción por ejemplo. De esta forma, y como se puede ver en el excel con todos los caminos representados, observamos que tenemos un caso de prueba para cada camino posible. Entre otros casos, decidimos probar el caso en el que introducimos un iban no correspondiente y los demás un test para cada IBAN correspondiente, verificando que la suma de balances se hacía correctamente.

En el caso de los test con un resultado válido, inicializamos un iban válido y leemos el archivo “transactions.json”. Para cada iban coincidente con el inicializado, calculamos en el propio test el balance que se espera leer más adelante en el archivo donde se almacena los resultados para asegurarnos de que la suma se ha realizado correctamente además de inicializar también el date esperado. Llamamos a la función, y, verificamos que devuelve True y que se han introducido los datos de manera correcta comparándolos con los datos esperados mencionados anteriormente.

Para realizar la pruebas en donde el archivo “transactions.json” no se encuentra, en el propio test ‘bloqueamos’ la dirección de la ruta de archivo de forma que no se encuentre ningún archivo. En otro de los test, también modificamos el archivo, nos guardamos el original anteriormente para después modificarlo y corromper el formato JSON específico y una vez ejecutado el test, devolvemos el archivo a su estado original. Estas modificaciones también podíamos hacerlas para generar más casos de

prueba pero, consideramos que se comprueba el correcto funcionamiento de la función con las pruebas ya generadas.

Finalmente, en la ejecución de test inválidos seguimos la misma estructura, guardamos primero el contenido de “balance_store.json” original para luego verificar que no ha sido modificado y comprobamos que el error devuelto por la función coincide con el esperado.

Cabe mencionar que en esta función no incluimos la comprobación de si ya se ha registrado un iban con algún balance o no, ya que consideramos que se pueden hacer tantos balances como se quieran.

2. Corrección a nuestros compañeros del grupo 05

2.1. Tabla de clases de equivalencia y valores límites

- **Variable from_iban:**
  Todo bien.

- **Variable to_iban:**
  Todo bien.

- **Variable concept:**
  Debería de añadirse una CEV donde concept tiene dos o más palabras. Por tanto añadir también los VL para dos y tres palabras y VLNV de una palabra.

- **Variable type:**
  Todo bien.

- **Variable date:**
  Todo bien.

- **Variable amount:**
  La cantidad a introducir puede tener hasta dos decimales, por lo que faltaría incluir un VL para el caso en el que solo haya un decimal.

- **Salidas esperadas:**
  Se debería incluir una clase de equivalencia no válida CENV para incluir la excepción donde existe una transferencia en el fichero con los mismos datos que se han introducido.

2.2. Tabla casos de prueba para las clases de equivalencia y los valores límite

- **Variable from_iban:**
  - Separar los casos inválidos de from_iban y to_iban para que se vea de forma clara y asegurarse que las reglas se cumplen para ambas.

- **Variable to_iban:**
  - Separar from_iban y to_iban.

- **Variable concept:**
  - Separar el CVNV 10 y 12. Es decir, comprobar la longitud por una parte y por otra los caracteres del string con una longitud válida.

- **Variable type:**
  Todo bien.

- **Variable date:**
  - Faltan hacer casos de prueba para corroborar las CEV22 y CENV24.

- **Variable amount:**
  - Añadir las pruebas del nuevo VL.

- **Salidas esperadas:**
  - Añadir la prueba para la nueva salida CENV en la cual todas las variables son válidas, pero el fichero ya existe (excepción).

- **Globales:**
  - En el campo techine, especificar si es una clase de equivalencia, un valor límite o ambos.

  - Además, habría que añadir casos de prueba para las correcciones mencionadas anteriormente.

  - (Recomendación) En vez de poner datos numéricos, especificar qué tipo de dato se introduce, de forma que se entienda de forma clara el por qué es válido o inválido. Por ejemplo, en vez de "ES9923456789012345678 90" poner ‘string de longitud = 23’, de esta forma queda mucho más claro y visual.