Práctica 3

Desarrollo de Software

ID Grupo: 01

Alberto Penas Díaz
NIA: 100471939
Correo:100471939@alumnos.uc3m.es
Grupo: 80
Titulación: Ingeniería informática

Natalia Rodríguez Navarro
NIA: 100471976
Correo: 100471976@alumnos.uc3m.es
Grupo: 80
Titulación: Ingeniería informática

Índice

INTRODUCCIÓN

En este informe se detalla la **definición** de los tests proporcionados junto con el código que aborda el ejercicio guiado 3 para comprobar el correcto funcionamiento de las funciones pedidas. En el caso del RF1, se hace hincapié en los valores límite y clases de equivalencia, en el de RF2, en el análisis sintáctico junto con la gramática y el árbol de derivación generado por la misma, y en el caso del RF3, en las pruebas estructurales generadas por los posibles caminos que define el diagrama de flujo de nuestro código.

Para mayor eficiencia, en la implementación hay pruebas que se realizan junto a otras (indicado en el *excel* adjuntado en la carpeta *docs* del proyecto), por ello, aunque en este documento definamos cada una de ellas, realmente hay pruebas repetidas que ya no volvemos a generar.

Compilación de los test: En caso de eliminar *store_deliveries.json, store_order_request.json* y *store_shipping_order.json*, los tests que generan éstos deberán ser ejecutados en orden, ya que hay funciones (en concreto RF2 y RF3) que requieren la existencia de almacenes previos. Para su correcta creación se ejecutará primeramente el test de *register_order()*, después el de *send_product()* y finalmente el de *deliver_product()*.

Nota1: PyBuilder no pasa por el 100% del código porque hay excepciones que no contemplamos en los test de las funciones RF1 y RF2, ya que para ellas no utilizamos el método de pruebas estructurales en la realización de los casos de prueba.

Nota2: Debido a la extensión del diagrama de flujo, hemos decidido añadir una imagen de copia en la carpeta *docs* del proyecto para que éste pueda ser visto con mayor claridad.

RF1: Register Order

Siguiendo el proceso TDD, define los casos de prueba e implementa la primera función. Aplicar en este primer método el Análisis de Clases de Equivalencia y los Valores de Límite (cuando corresponda).

DEFINICIÓN DE LOS CASOS DE PRUEBA

Se han determinado las posibles clases de equivalencia y valores límite de todos los argumentos que posee la primera función. Todos ellos están numerados en el *excel* adjuntado en la carpeta *docs* del proyecto. A grandes rasgos, se pueden resumir en:

1. product_id: hemos definido 5 pruebas no repetidas que conforman la comprobación de la validez de este campo. Sus clases de equivalencia y valores límite son:

    a. El EAN13 es correcto — VÁLIDO
    b. El EAN13 no es un número. — INVÁLIDO
    c. El dígito de control no es coincidente. — INVÁLIDO
    d. El EAN13 tiene 12 dígitos — INVÁLIDO
    e. El EAN13 tiene 14 dígitos — INVÁLIDO

2. order_type: hemos definido 3 pruebas para verificar la validez de este campo. De éstas, la primera será comprobada junto con la prueba válida que hemos hecho para verificar el *product_id*, ya que este valor solo puede ser “Premium” o “Regular”. Sus clases de equivalencia:

    a. El order type es Premium — VÁLIDO
    b. El order type es Regular — VÁLIDO
    c. El order type es incorrecto — INVÁLIDO

3. delivery_address: hemos definido 10 pruebas para verificar la validez de este campo, las tres primeras ya son cubiertas en pruebas anteriores. Sus clases de equivalencia y valores límite son:

    a. Dirección correcta — VÁLIDO
    b. Dirección con un espacio blanco — VÁLIDO
    c. Dirección con dos espacios blancos — VÁLIDO
    d. Dirección con 20 caracteres — VÁLIDO
    e. Dirección con 21 caracteres — VÁLIDO
    f. Dirección con 99 caracteres — VÁLIDO
    g. Dirección con 100 caracteres — VÁLIDO
    h. Dirección con 19 caracteres — INVÁLIDO
    i. Dirección con 101 caracteres — INVÁLIDO
    j. Dirección sin espacios blancos — INVÁLIDO

4. phone_number: hemos definido 4 pruebas para verificar la validez de este campo, de las cuales la primera ya consideramos en pruebas anteriores. Sus clases de equivalencia y valores límite son:

a.	Número de teléfono correcto	VÁLIDO
b.	Número de teléfono con 8 dígitos	INVÁLIDO
c.	Número de teléfono con 10 dígitos	INVÁLIDO
d.	Número de teléfono con algún carácter	INVÁLIDO

5. zip_code: hemos definido 11 pruebas para verificar la validez de este campo. Aquí, comprobamos que los dos primeros dígitos son correctos, ya que corresponden a una numeración de las provincias del estado español que va del 01 al 52. Las dos primeras son consideradas la misma, pero la añadimos para mayor entendimiento. Sus clases de equivalencia y valores límite son:

a.	Código postal cualquiera correcto	VÁLIDO
b.	Código postal 5 dígitos	VÁLIDO
c.	Código postal primeros dos dígitos iguales a 01	VÁLIDO
d.	Código postal primeros dos dígitos iguales a 02	VÁLIDO
e.	Código postal primeros dos dígitos iguales a 51	VÁLIDO
f.	Código postal primeros dos dígitos iguales a 52	VÁLIDO
g.	Código postal primeros dos dígitos iguales a 00	INVÁLIDO
h.	Código postal primeros dos dígitos iguales a 53	INVÁLIDO
i.	Código postal 6 dígitos	INVÁLIDO
j.	Código postal 4 dígitos	INVÁLIDO
k.	Código postal con algún carácter	INVÁLIDO

6. output: por último, por cada test realizado comprobamos que se han registrado correctamente los datos del pedido verificando que se han introducido en el almacén de pedidos *store_order_request.json*. Respecto al MD5, consideramos que no hay que hacer comprobaciones ya que corresponde a una función interna.

RF2: Send Product

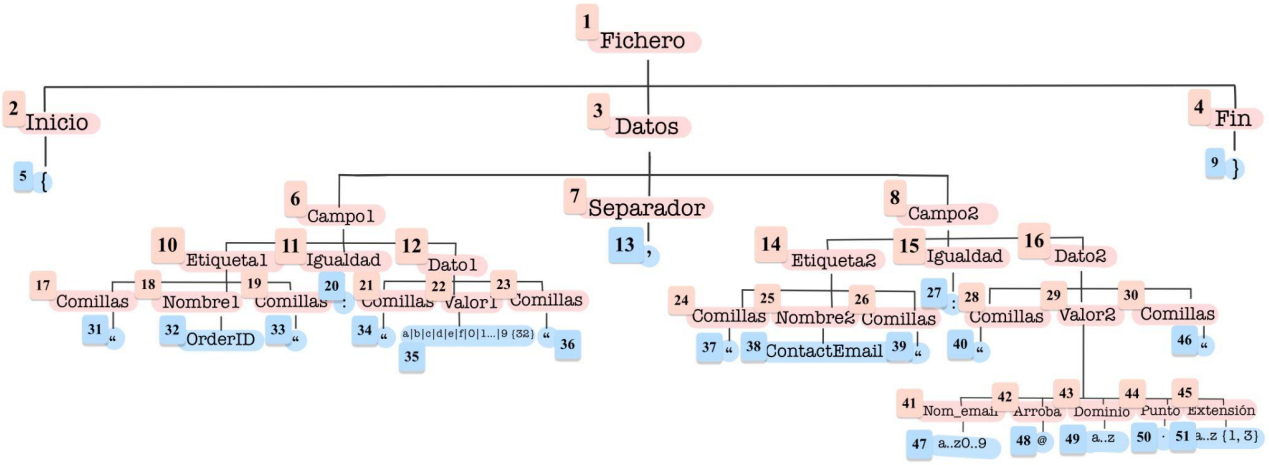
Define los casos de prueba e implementa la segunda función. En este caso, aplica la técnica de Análisis sintáctico. Si es necesario, complementa las pruebas aplicando clases de equivalencia y valores límite

A continuación, presentamos la gramática y el árbol de derivación correspondiente generado para seguir los pasos de la técnica de análisis sintáctico.

GRAMÁTICA:

Fichero ::= Inicio Datos Fin
Inicio ::= {
Fin ::= }
Datos ::= Campo1 Separador Campo2
Campo1 ::= Etiqueta1 Igualdad Dato1
Campo2 ::= Etiqueta2 Igualdad Dato2
Separador ::= ,
Igualdad ::= :
Etiqueta1 ::= Comillas Nombre1 Comillas
Nombre1 ::= OrderID
Dato1 ::= Comillas Valor1 Comillas
Valor1 ::= a|b|c|d|e|f|0|1…|9 {32}
Etiqueta2 ::= Comillas Nombre2 Comillas
Nombre2 ::= ContactEmail
Dato2 ::= Comillas Valor2 Comillas
Valor2 ::= Nom_email Arroba Dominio Punto Extensión
Nom_email ::= a..z0..9
Arroba ::= @
Dominio ::= a..z
Punto ::= .
Extensión ::= a..z {1,3}
Comillas ::= “

ÁRBOL DE DERIVACIÓN:



DEFINICIÓN DE LOS CASOS DE PRUEBA:

Hemos definido los casos de prueba para comprobar el correcto formato de la entrada siguiendo el algoritmo de análisis sintáctico. De esta forma, hemos generado en primer lugar, la gramática que permite la creación de un fichero *.json* válido como entrada para el método *send_product()*. En segundo lugar, hemos representado dicha gramática en un árbol de derivación con nodos numerados de izquierda a derecha y de arriba a abajo, que posteriormente nos servirá para definir los casos de prueba. Por último, hemos precisado los casos mediante la duplicación y eliminación de los nodos no terminales y la modificación de los nodos terminales.

Por esta razón, surgen estructuras ya contempladas en otros nodos lo que nos permite agrupar las pruebas. Es decir: el nodo 11 contempla las mismas pruebas que el 15, el nodo 17 contempla las mismas pruebas que el 19, 21, 23, 24, 26, 28 y 30, el nodo 20 cubre el nodo 27, y, el nodo 31 cubre el 33, 34, 36, 37, 39, 40 y 46. De esta forma, únicamente realizamos las pruebas en uno de ellos, como bien indicamos en el *excel*.

Por último, hemos añadido pruebas adicionales que contemplan las clases de equivalencia y valores límite de algunos valores. En concreto, es necesario contemplarlo en el nodo 35 (hash MD5) y en el nodo 47, 49 y 51 (email):

1. MD5: Como ya tenemos un caso válido, solamente necesitamos modificar este valor para los valores límite 31 DÍGITOS (inválido) y 33 DÍGITOS (inválido), y para la clase de equivalencia NO HEXADECIMAL (inválido).

2. Nombre del email: Como tenemos un caso válido y realizamos la modificación del nodo en pruebas anteriores ya cubrimos las clases de equivalencia, por lo que no

añadimos más. De todas formas, está indicado en el *excel* su definición y cuál es la prueba anterior que lo cubre.

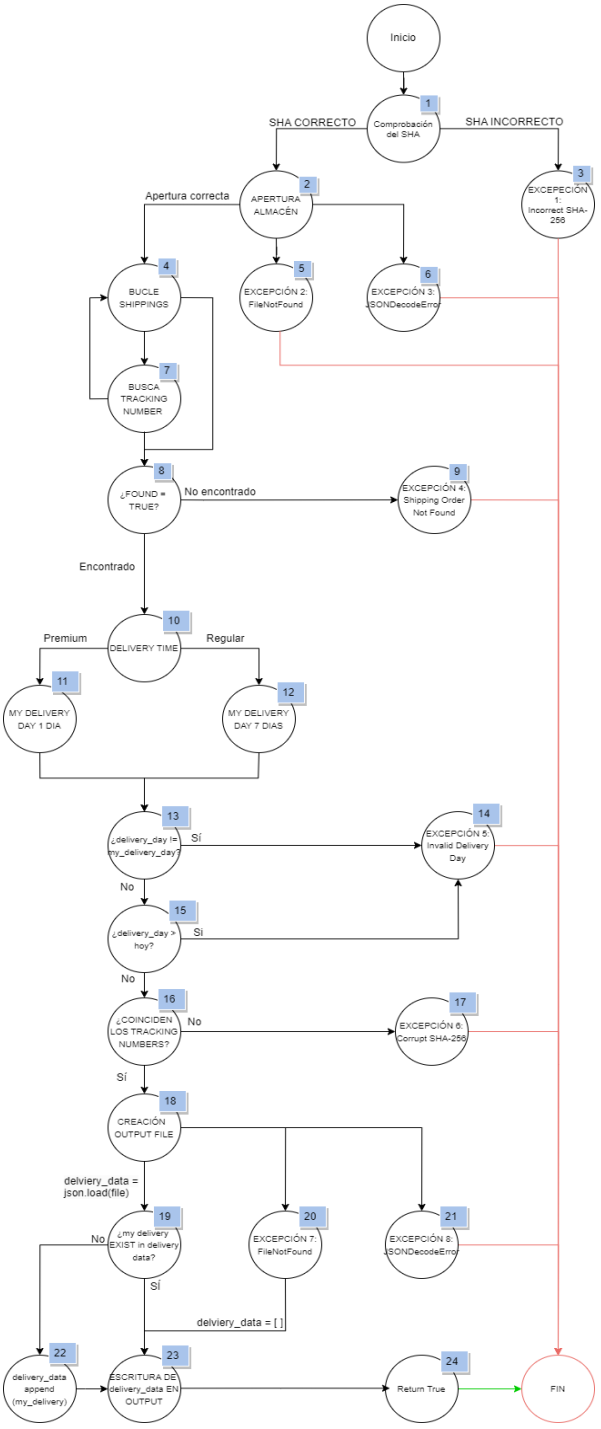
3. Dominio del email: De la misma manera que con el nombre, las clases de equivalencia ya son cubiertas en casos anteriores que siguen el método de análisis sintáctico.

4. Extensión del email: En este caso, se cubren prácticamente todas menos cuatro. Éstos son los valores límites: 1 LETRA (válido), 2 LETRAS (válido) y 4 LETRAS (inválido). Como bien pasa en las anteriores, 0 LETRAS, 3 LETRAS y NO LETRA ya se valora en otras pruebas del análisis sintáctico.

RF3: Deliver Product

Define los casos de prueba e implementa la tercera función. Aplicar en este tercer caso técnicas de prueba estructural.

GRÁFICO DE CONTROL DE FLUJO:



DEFINICIÓN DE LAS RUTAS BÁSICAS:

Hemos seguido el algoritmo de producción de las pruebas estructurales, creando en primer lugar un diagrama de flujo que identifique los posibles recorridos que tiene nuestra función internamente. De esta manera, y aplicando la fórmula de la complejidad de McCabe, hemos obtenido el número de rutas posibles que puede tener nuestro código al ejecutarse:

V(G) = Enlaces - Nodos + 2 = 38 - 26 + 2 = 14

Estas rutas las hemos obtenido siguiendo el algoritmo de definir el camino de referencia e ir cambiando la primera decisión para definir un camino, la segunda para definir otro y así sucesivamente hasta agotar las decisiones.

DEFINICIÓN DE LOS CASOS DE PRUEBA:

Por lo tanto, las pruebas a definir vienen dadas por los siguientes caminos:

1_2_4_7_8_10_11_13_15_16_18_19_22_23_24
1_3
1_2_5
1_2_6
1_2_4_8_9
1_2_4_8_10_11_13_15_16_18_19_22_23_24
1_2_4_7_8_9
1_2_4_7_8_10_12_13_15_16_18_19_22_23_24
1_2_4_7_8_10_11_13_14
1_2_4_7_8_10_11_13_15_14
1_2_4_7_8_10_11_13_15_16_17
1_2_4_7_8_10_11_13_15_16_18_20_23_24
1_2_4_7_8_10_11_13_15_16_18_21
1_2_4_7_8_10_11_13_15_16_18_19_23_24

Por esta razón, hemos creado las entradas correspondientes para emular un entorno en el que se generen las excepciones deseadas o se pase por la línea de código buscada según la ruta. Por ejemplo, en la prueba *test_deliver_product_nok_4()*. Lo que hubiera en el almacén sobre el que estamos trabajando (*store_shipping_order.json*) lo guardamos temporalmente en una variable (shippings).

```
with open(str(Path.home()) + "/PycharmProjects/G80.2023.T01.EG3/src/"
                             "json_files/store_shipping_order.json",
          "r", encoding="UTF-8", newline="") as file:
    shippings = json.load(file)
```

Después copiamos en el almacén un nuevo fichero *json* creado por nosotros con información incorrecta (en este caso, el *json* que copiamos solo contiene una llave " { ", ya que queremos que la excepción que genere sea "JSON Decode Error"). La librería que utilizamos para copiar información de un fichero a otro es shutil:

```
shutil.copy(str(Path.home()) + "/PycharmProjects/G80.2023.T01.EG3/src/"
                               "json_files/json_tests/mytest1_delivery.json",
            str(Path.home()) + "/PycharmProjects/G80.2023.T01.EG3/src/"
                               "json_files/store_shipping_order.json")
```

Una vez ya hemos configurado nuestro entorno, ejecutamos el respectivo test para que nos salte la excepción deseada:

```
my_delivery = OrderManager()

with self.assertRaises(OrderManagementException) as order_except:
    my_delivery.deliver_product("c7d0c3b0098a98d782981e6c5d7f5a28"
                                "08dd0c6841dd12c1932e9ad9499b243c")

self.assertEqual("Json Decode Error - Wrong Json format", order_except.exception.message)
```

Por último, restauramos el fichero anterior con la información que habíamos guardado:

```
if os.path.isfile(str(Path.home()) + "/PycharmProjects/G80.2023.T01.EG3/src/"
                                     "json_files/store_shipping_order.json"):
    os.remove(str(Path.home()) + "/PycharmProjects/G80.2023.T01.EG3/src"
                                 "/json_files/store_shipping_order.json")

# Volvemos a generar el fichero que existía antes
with open(str(Path.home()) + "/PycharmProjects/G80.2023.T01.EG3/src/"
                             "json_files/store_shipping_order.json", "w",
          encoding="UTF-8", newline="") as file:
    json.dump(shippings, file, indent=2)
```

Hemos seguido este procedimiento para prácticamente todos los tests que hemos realizado para comprobar que pasa por todas las líneas de código del RF3.

CASOS ADICIONALES PARA PROBAR EL BUCLE:

Hemos considerado los siguientes casos:

- No pasa por el bucle
- Pasa por el bucle 1 vez (el pedido que buscamos está el primero)
- Pasa por el bucle 2 veces (el pedido que buscamos está el segundo)
- Pasa por el bucle el número máximo de veces menos uno (es decir, el elemento que se quiere encontrar está en penúltima posición en el almacén de envíos)
- Pasa por el bucle el número máximo de veces (esta vez, el elemento está en última posición)
- Pasa por el bucle el número máximo de veces más uno (en este caso, no podemos generar el unittest que lo comprueba, ya que no es posible leer más del contenido almacenado en el almacén *store_shipping_order*).

CLASE ORDER DELIVERY:

Cabe destacar que para abordar esta función y seguir con la estructura del proyecto hemos generado el módulo order_delivery.py, que define la clase OrderDelivery.

CONCLUSIÓN

Habiendo realizado el proyecto con suficiente tiempo de antelación, nos hemos dado cuenta de la profundidad y complejidad que muchas veces hay a la hora de realizar las pruebas funcionales de un código para verificar que este se ejecute correctamente.

Podemos decir con bastante seguridad que no solamente hemos aprendido a realizar casos de prueba dependiendo de las entradas y complejidad de una función, si no que además, nos hemos dado cuenta de lo que se puede pasar por alto al realizar una función aparentemente simple.

La realización del proyecto ha sido tediosa pero fluida. Teniendo ya una base de conocimiento sólida sobre el funcionamiento de github, hemos implementado algunas funciones que nos han facilitado el desarrollo de código y, a nuestro parecer, han favorecido la programación en parejas. Un ejemplo de esto es la utilización continua de “issues”, que nos han permitido llevar un registro de los problemas encontrados en el trabajo y su correspondiente solución.

Por último, queríamos destacar y agradecer la rapidez con la que se nos han resuelto nuestras dudas en el foro de la asignatura y por el correo personal, lo que ha facilitado enormemente la rapidez con la que hemos avanzado en el ejercicio.