P2- Diseño de pruebas de caja blanca

Diseño de pruebas de caja blanca *(structural testing)*

El objetivo de esta práctica es aplicar el método de diseño de casos de prueba visto en clase para obtener un conjunto de casos de prueba de unidad (método java), a partir de la implementación de dicha unidad. Utilizaremos el método de caminos que hemos explicado en clase. Recuerda que no sólo se trata reproducir los pasos del método de forma mecánica, sino que, además, debes tener muy claro qué es lo que estás haciendo en cada momento, para así asimilar los conceptos explicados.

En esta sesión no utilizaremos ningún software, pero en la siguiente sesión automatizaremos la ejecución de los casos de prueba que hemos diseñado aquí, por lo que necesitarás tus soluciones de esta práctica para poder trabajar en la próxima clase.

Ejercicios

A continuación proporcionamos el código de las unidades a probar, así como sus especificaciones.

1. **Especificación 1.** Se trata de probar el siguiente método **`calculaTasaMatricula()`** que devuelve el valor de las tasas de matriculación en función de la edad, de si es familia numerosa, y si es o no repetidor, de acuerdo con la siguiente tabla (asumiendo que se aplican sobre un valor inicial de tasa=500 euros) :

	Edad < 25	Edad > 25	Edad 25..50	Edad 51..64	Edad ≥ 65
Edad	SI	SI	SI	SI	SI
Familia Numerosa	NO	SI	SI		
Repetidor	SI				
Valor tasa-total	tasa + 1500	tasa/2	tasa/2	tasa -100	tasa/2

Obtén una tabla de casos de prueba utilizando el método de diseño que hemos visto en clase, a partir del código siguiente:

```
1. public float calculaTasaMatricula(int edad, boolean familiaNumerosa,
2.                                    boolean repetidor) {
3.   float tasa = 500.00f;
4.
5.   if ((edad < 25) && (!familiaNumerosa) && (repetidor)) {
6.     tasa = tasa + 1500.00f;
7.   } else {
8.      if ((familiaNumerosa) || (edad >= 65)) {
9.        tasa = tasa / 2;
10.     }
11.     if ((edad > 50) && (edad < 65)) {
12.       tasa = tasa - 100.00f;
13.     }
14.  }
15.  return tasa;
16.}
```

2. **Especificación 2.** Se trata de probar el método `validaNif()`, cuya especificación es la siguiente: dada una cadena de caracteres que representa un NIF, devuelve cierto o falso en función de que efectivamente se corresponda con un NIF válido o no. El algoritmo para determinar si un NIF es válido o no lo podéis consultar en http://es.wikibooks.org/wiki/Algoritmo_para_obtener_la_letra_del_NIF.

   Obtén una tabla de casos de prueba utilizando el método de diseño que hemos visto en clase, a partir del código siguiente

```
1.public boolean validaNif(String nif) {
2.  if(nif.length()!=9 || nif==null) {
3.      return false;
4.  }
5.
6.  String dni = nif.substring(0, 8);
7.  char letra = nif.charAt(8);
8.
9.  //A partir de una expresión regular, obtenemos una representación
10.  //compilada (objeto de tipo Pattern)
11.  Pattern pattern = Pattern.compile("[0-9]{8,8}");
12.  //El patrón obtenido se utiliza para crear un objeto Matcher
13.  Matcher matcher = pattern.matcher(dni);
14.  String letras = "TRWAGMYFPDXBNJZSQVHLCKE";
15.
16.  long ldni = 0;
17.  try {
18.     ldni = Long.parseLong(dni);
19.  } catch(NumberFormatException e) {
20.       return false;
21.  }
22.
23.  int indice = (int)(ldni % 23);
24.  char letraEsperada = letras.charAt(indice);
25.
26.  return matcher.matches() && letra==letraEsperada;
27.}
```

> **Nota**: El método *Pattern.compile( )*, recibe como entrada una expresión regular **X** y devuelve una instancia de la clase *Pattern*. El método *Pattern.matcher( )*, recibe como entrada una secuencia de caracteres **S**, sobre la que quiere hacer "*matching*" con **X**. El resultado del *matching* de **X** con **S** se obtiene invocando al método Matcher.matches( ). (Para más información consultar https://docs.oracle.com/javase/7/docs/api/java/util/regex/Pattern.html).
>
> El método *Long.parseLong( )* admite como válidos números negativos
>
> (se puede consultar en: http://docs.oracle.com/javase/7/docs/api/java/lang/Long.html#parseLong(java.lang.String)).

3. **Especificación 3.** Se trata de probar el método `realizaReserva()`. Dicho método realiza una reserva de una lista de libros de una biblioteca a uno de los socios de la misma. Solamente el usuario con el rol de bibliotecario puede realizar la reserva,. El método compruebaPermisos(), devuelve cierto si la persona que hace la reserva es el bibliotecario y falso en caso contrario. La clase `FactoriaBOs` devuelve instancias de objetos de negocio de tipo `IOperacionBO`, a partir de los cuales, podemos hacer efectiva la reserva utilizando el método `getOperacionBO()`. Para hacer efectiva la reserva es necesario introducir el identificador del socio en la base de datos, así como el `isbn` del libro que quiere reservar. El método `realizaReserva()` puede lanzar la excepción `IsbnInvalidoException` (si el isbn

del libro no existe en la BD, con el mensaje "ISBN invalido:isbn"). Por otro lado, si el socio no existe en la BD, el método de reserva devolverá la excepción `SocioInvalidoException` (con el mensaje "`SOCIO invalido`"). Finalmente, si no puede acceder a la BD, se generará la excepción `JDBCException`, con el mensaje "`CONEXION invalida`". Todos los mensajes de las excepciones generadas, se almacenan en una lista de mensajes de error. Si dicha lista contiene uno o más elementos, el método `realizaReserva()`, a su vez, generará la excepción `ReservaException`, con un mensaje formado por los elementos de la lista de mensajes de error, separados por "; ". Obtén una tabla de casos de prueba utilizando el método de diseño que hemos visto en clase, a partir del código siguiente:

```
1.public void realizaReserva(String login, String password,
2.                        String socio, String [] isbns) throws Exception {
3.
4.  ArrayList<String> errores = new ArrayList<String>();
5.  if(!compruebaPermisos(login, password,
6.                         Usuario.BIBLIOTECARIO)) {
7.    errores.add("ERROR de permisos");
8.  } else {
9.    FactoriaBOs fd = FactoriaBOs.getInstance();
10.    IOperacionBO io = fd.getOperacionBO();
11.    try {
12.      for(String isbn: isbns) {
13.        try {
14.          io.realizaReserva(socio, isbn);
15.        } catch (IsbnInvalidoException iie) {
16.          errores.add("ISBN invalido" + ":" +  isbn);
17.        }
18.      }
19.    } catch (SocioInvalidoException sie) {
20.      errores.add("SOCIO invalido");
21.    } catch (JDBCException je) {
22.      errores.add("CONEXION invalida");
23.    }
24.  }
25.  if (errores.size() > 0) {
26.    String mensajeError = "";
27.    for(String error: errores) {
28.      mensajeError += error + "; ";
29.    }
30.    throw new ReservaException(mensajeError);
31.  }
32.}
```

Bitbucket

El trabajo de esta sesión también debes subirlo a Bitbucket. Todo el trabajo de esta práctica, tanto las tablas, como cualquier otro documento con vuestras notas de trabajo, deberán estar en la carpeta **P2.** El formato de los ficheros que contienen las tablas debe ser preferiblemente pdf o formato de imagen (png, jpg). Recuerda que necesitas utilizar las tablas obtenidas para la siguiente práctica.

Asimismo, todo el trabajo de la práctica anterior debéis reorganizarlo para que resida en la carpeta **P1** (esta carpeta contendrá todos los ficheros con vuestras notas sobre la práctica, así como el directorio P1-netbeans, con el código del proyecto).