P6- Dependencias externas (2)

Verificación basada en el comportamiento. Librería EasyMock.

El objetivo de esta práctica es automatizar pruebas unitarias, implementando drivers que utilicen verificación basada en el comportamiento. Usaremos la librería EasyMock para implementar los tests unitarios.

Bitbucket

El trabajo de esta sesión también debes subirlo a Bitbucket. Todo el trabajo de esta práctica, tanto los proyectos maven que vamos a crear, como cualquier otro documento con vuestras notas de trabajo, deberán estar en la carpeta **P6** de vuestro repositorio**.**

Ejercicios

1. Crea un nuevo proyecto Maven "**gestorllamadas**" (recuerda que debes crearlo en el directorio **P6**), con valores para ArtifactId y GroupId **gestorllamadas** y **ppss**, respectivamente. En el campo de texto Package que nos muestra Netbeans pondremos como valor **ppss**.

   Dado el siguiente código: , se trata de automatizar las pruebas unitarias sobre el método **ppss.ejercicio1.GestorLlamadas.calculaConsumo()**, según la tabla de casos de prueba proporcionada, de forma que:

```
public class GestorLlamadas {
  static double TARIFA_NOCTURNA=10.5;
  static double TARIFA_DIURNA=20.8;

  public Calendario getCalendario() {
    Calendario c = new Calendario();
    return c;
  }

  public double calculaConsumo(int minutos) {
    Calendario c = getCalendario();
    int hora = c.getHoraActual();
    if(hora < 8 || hora > 20) {
      return minutos * TARIFA_NOCTURNA;
    } else {
       return minutos * TARIFA_DIURNA;
    }
  }
}
```

	minutos	hora	Resultado esperado
C1	10	15	208
C2	10	22	105

```
public class Calendario {
  public int getHoraActual() {
    throw new
      UnsupportedOperationException
           ("Not yet implemented");
  }
}
```

   A) Implementa los drivers utilizando verificación basada en el comportamiento sin utilizar un mocking parcial.

      La clase que contiene los tests se llamará *GestorLlamadasMockTest.java*

   B) Implementa los drivers utilizando verificación basada en el comportamiento utilizando un mocking parcial. En este caso la clase que contiene los tests se llamará *GestorLlamadasPartialMockTest.java* (recuerda que debe pertenecer al mismo paquete que la clase a probar)

2. Crea un nuevo proyecto Maven "**premio**" (recuerda que debes crearlo en el directorio P6), con valores para ArtifactId y GroupId **premio** y **ppss**, respectivamente. En el campo de texto Package que nos muestra Netbeans pondremos como valor **ppss**.

   Para este ejercicio proporcionamos el código del método **ppss.ejercicio2.Premio.compruebaPremio()**

```
public class Premio {
    private static final float PROBABILIDAD_PREMIO = 0.1f;
    public Random generador = new Random(System.currentTimeMillis());
    public ClienteWebService cliente = new ClienteWebService();

    public String compruebaPremio() {
        if(generaNumero() <  PROBABILIDAD_PREMIO) {
            try {
                String premio = cliente.obtenerPremio();
                return "Premiado con " + premio;
            } catch(ClienteWebServiceException e) {
                return "Premiado";
            }
        } else {
            return "Sin premio";
        }
    }

    // Genera numero aleatorio entre 0 y 1
    public float generaNumero() {
        return generador.nextFloat();
    }
}
```

   Se trata de implementar los siguientes tests unitarios sobre el método anterior, utilizando verificación basada en el comportamiento:

   A) el número aleatorio generado es de 0,01, el servicio de consulta del premio (método obtenerPremio) devuelve “pez de goma”, y el resultado esperado es “Premiado con pez de goma”

   B) el número aleatorio generado es de 0,01, el servicio de consulta del premio (método obtenerPremio) devuelve una excepción de tipo ClientWebServiceException, y el resultado esperado es “Premiado”

3. Crea un nuevo proyecto Maven "**webClient**", con valores para ArtifactId y GroupId **webClient** y **ppss**, respectivamente. En el campo de texto Package que nos muestra Netbeans pondremos como valor **ppss**.

   Para este ejercicio proporcionamos el código del método **ppss.ejercicio3.Webclient.getContent()**, así como la tabla de casos de prueba correspondiente. Dicho método abre una conexión http (de tipo HttpURLConnection) contenida en la url que se pasa como parámetro del método, y lee el contenido de dicha conexión, devolviendo como resultado la cadena de caracteres que representa dicho contenido, o bien el valor null, en el caso de que haya habido algún error, bien en la conexión, o porque la url no es válida.

```
import java.io.IOException;
import java.io.InputStream;
import java.net.HttpURLConnection;
import java.net.URL;

public class WebClient {
  public String getContent(URL url) {
    StringBuilder content = new StringBuilder();
    try {
      //creamos una conexión http
      HttpURLConnection connection = createHttpURLConnection(url);
      //abrimos la conexión http, puede lanzar la excepción IOException
      InputStream is = connection.getInputStream();

      //leemos el contenido de la conexión
      byte[] buffer = new byte[2048];
      int count;
      while (-1 != (count = is.read(buffer))) {
       content.append(new String(buffer, 0, count));
      }
     }
     catch (IOException e){
           return null;
     }
     return content.toString();
    }

  protected HttpURLConnection createHttpURLConnection(URL url)throws IOException {
      return (HttpURLConnection) url.openConnection();
  }
}
```

	url	fallo al abrir conexión	Resultado esperado
C1	http://www.ua.es	NO	"Funciona"
C2	http://www.ua.es	SI	null

Nota: Para crear una nueva instancia de un objeto de tipo URL puedes utilizar el método `new URL(String direccionHttp)`. Para convertir un *String* a un objeto de tipo *InputStream* puedes hacerlo con: `new ByteArrayInputStream("Cadena_a_convertir".getBytes())`

Teniendo en cuenta que NO estamos interesados en probar la conexión http real, se pide implementar los drivers para la tabla anterior utilizando verificación basada en el comportamiento. La clase que contiene los tests se llamará, por ejemplo, WebClientMockTest.java

4. Crea un nuevo proyecto Maven "**reserva**" (recuerda que debes crearlo en el directorio P6), con valores para ArtifactId y GroupId **reserva** y **ppss**, respectivamente. En el campo de texto Package que nos muestra Netbeans pondremos como valor **ppss**.

   Para este ejercicio proporcionamos el código del método **ppss.ejercicio4.Reserva.realizaReserva()**, así como la tabla de casos de prueba correspondiente.

```
public class Reserva {
  private List<String> errores = new ArrayList<String>();

  protected FactoriaBOs getFactoriaBOs() {
     return new FactoriaBOs();
  }

  public boolean compruebaPermisos(String login, String password, Usuario tipoUsu) {
        throw new UnsupportedOperationException("Not yet implemented");
  }

  public void realizaReserva(String login, String password,
                          String socio, String [] isbns) throws ReservaException {

    ArrayList<String> errores = new ArrayList<String>();
    if(!compruebaPermisos(login, password, Usuario.BIBLIOTECARIO)) {
         errores.add("ERROR de permisos");
    } else {
        FactoriaBOs fd = getFactoriaBOs();
        IOperacionBO io = fd.getOperacionBO();
        try {
           for(String isbn: isbns) {
              try {
                 io.operacionReserva(socio, isbn);
               } catch (IsbnInvalidoException iie) {
                   errores.add("ISBN invalido" + ":" +  isbn);
                }
            }
         } catch (SocioInvalidoException sie) {
                errores.add("SOCIO invalido");
         } catch (JDBCException je) {
             errores.add("CONEXION invalida");
        }
     }
     if (errores.size() > 0) {
        String mensajeError = "";
        for(String error: errores) {
           mensajeError += error + "; ";
         }
        throw new ReservaException(mensajeError);
     }
  }
}
```

Las excepciones debes implementarlas en el paquete **ppss.ejercicio4.excepciones**

La definición de la interfaz **IOperacionBO** y del tipo enumerado son las mismas que en la práctica anterior.

La definición de la clase **FactoriaBOs** es la siguiente:

```
public class FactoriaBOs {
   public IOperacionBO getOperacionBO(){
        throw new UnsupportedOperationException("Not yet implemented");
    }
}
```

La tabla de casos de prueba es la siguiente:

	login	password	ident. socio	Acceso BD	isbns	Resultado esperado
C1	"xxxx"	"xxxx"	"Luis"	OK	{"11111"}	ReservaException1
C2	"ppss"	"ppss"	"Luis"	OK	{"11111", "22222"},	No se lanza excep.
C3	"ppss"	"ppss"	"Luis"	OK	{"33333"}	ReservaException2
C4	"ppss"	"ppss"	"Pepe"	OK	["11111"}	ReservaException3
C5	"ppss"	"ppss"	"Luis"	fallo	{"11111"}	ReservaException4

ReservaException1: Excepción de tipo ReservaException con el mensaje: "ERROR de permisos; "

ReservaException2: Excepción de tipo ReservaException con el mensaje: "ISBN invalido:33333; "

ReservaException3: Excepción de tipo ReservaException con el mensaje: "SOCIO invalido; "

ReservaException4: Excepción de tipo ReservaException con el mensaje: "CONEXION invalida; "

Nota: Suponemos que el login/password del bibliotecario es "ppss"/"ppss"; que "Luis" es un socio y "Pepe" no lo es; y que los isbn registrados en la base de datos son "11111", "22222".

Utilizando la librería EasyMock, se pide lo siguiente:

A) Implementa los drivers utilizando verificación basada en el comportamiento. La clase que contiene los tests se llamará, por ejemplo, ReservaMockTest.java

B) Implementa los drivers utilizando verificación basada en el estado. La clase que contiene los tests se llamará, por ejemplo, ReservaStubTest.java

5. Crea un nuevo proyecto Maven "**listados**", con valores para ArtifactId y GroupId **listados** y **ppss**, respectivamente. En el campo de texto Package que nos muestra Netbeans pondremos como valor **ppss**.

   Para este ejercicio proporcionamos el código del método **ppss.ejercicio5.porApellidos()**, así como información sobre el estado de la tabla alumnos antes de ejecutar el método. El método en cuestión obtiene un listado de alumnos después de acceder a una base de datos (tabla alumnos)

```
public class Listados {
  public String porApellidos(Connection con, String tableName) throws SQLException {
   Statement stm = con.createStatement();
  //realizamos la consulta y almacenamos el resultado en un ResultSet
   ResultSet rs =
      stm.executeQuery("SELECT apellido1, apellido2, nombre FROM " + tableName);
   String result = "";
   //recorremos el ResulSet
    while (rs.next()) {
      String ap1 = rs.getString("apellido1");
      String ap2 = rs.getString("apellido2");
      String nom= rs.getString("nombre");
     result += ap1 + ", " + ap2 + ", " + nom + "\n";
   }
   return result;
  }
}
```

Nota: Connection, Statement y ResultSet son Interfaces Java, cuyos métodos pueden devolver una excepción de tipo SQLException, al igual que el método a probar.

tabla alumnos

Apellido1	Apellido2	Nombre
Garcia	Molina	Ana
Lacalle	Verna	Jose Luis

Implementa los drivers para automatizar los siguientes casos de pruebas unitarias utilizando verificación basada en el comportamiento. No queremos acceder a la base de datos real. Asumiremos en todos los casos que solicitamos un listado de la tabla alumnos, cuyo estado inicial es el que hemos indicado, y que proporcionamos una instancia de la clase Connection como entrada.

A) Suponemos que no se lanza ninguna excepción durante el acceso a la base de datos, ni al recorrer el ResultSet. El resultado esperado será: *"Garcia, Molina, Ana\nLacalle, Verna, Jose Luis\n"*

B) Suponemos que se lanza una excepción al recorrer el ResultSet (en la primera invocación del método next()). El resultado esperado será que se lanza una excepción de tipo SQLException.