P5- Dependencias externas (1)

Control de dependencias externas y verificación basada en el estado

El objetivo de esta práctica es automatizar pruebas unitarias. En este caso, las unidades a probar tienen dependencias externas que necesitaremos controlar aplicando lo que hemos explicado en clase. Recuerda que cuando realizamos pruebas unitarias, tenemos que aislar la ejecución de nuestra SUT, de esta forma nos aseguramos de que cualquier defecto que detectemos estará exclusivamente en el código de nuestro SUT, y no en cualquier otra unidad de la que dependa.

Para controlar las dependencias externas implementaremos stubs, como hemos visto en clase.

Para automatizar las pruebas utilizaremos una verificación basada en el estado, tal y como hemos explicado en clase. Además, tendremos que implementar los drivers para realizar las correspondientes pruebas de integración.

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 **P5** de vuestro repositorio**.**

Ejercicios

1. Crea un nuevo proyecto Maven "**gestorllamadas**" (recuerda que debes crearlo en el directorio P5), con valores para ArtifactId y GroupId **gestorllamadas** y **ppss**, respectivamente. En el campo de texto Package que nos muestra Netbeans pondremos como valor **ppss**.

   Se trata de automatizar las pruebas unitarias sobre el método **GestorLlamadas.calculaConsumo()** (que pertenece al paquete **ppss.ejercicio1**) utilizando verificación basada en el estado.

```
public class GestorLlamadas {
  static double TARIFA_NOCTURNA=10.5;
  static double TARIFA_DIURNA=20.8;
  public int getHoraActual() {
    Calendar c = Calendar.getInstance();
    int hora = c.get(Calendar.HOUR);
        return hora;
    }

  public double calculaConsumo(int minutos) {
    int hora = 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

Otra forma de crear los tests desde Netbeans

Para implementar los drivers, podemos utilizar la opción "Tools->Create/Update Tests", desde el menú contextual de la clase sobre la que se quieren implementar los drivers

2. Añade en el proyecto **gestorllamadas** el paquete **ppss.ejercicio2**, que contiene el código de las clases **GestorLlamadas** y **Calendario**, tal y como se muestra a continuación. Implementa los casos de prueba de la tabla del ejercicio anterior para probar el método **GestorLlamadas.calculaConsumo()** utilizando verificación basada en el estado

```
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;
    }
  }
}
```

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

3. Crea un nuevo proyecto Maven "**reserva**" (recuerda que debes crearlo en el directorio P5), con valores para ArtifactId y GroupId **reserva** y **ppss**, respectivamente. En el campo de texto Package que nos muestra Netbeans pondremos como valor **ppss**.

   Se trata de automatizar las pruebas unitarias sobre el método **Reserva.realizaReserva()** (que pertenece al paquete **ppss.ejercicio3**), cuyo código se muestra más adelante.

   También proporcionamos la tabla de casos de prueba que debéis utilizar para poder implementar los drivers correspondientes, realizando una verificación basada en el estado. Utiliza una clase factoría si necesitas refactorizar la SUT.

	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

   Implementa, además, los drivers para realizar tests de integración, utilizando la tabla proporcionada y tests parametrizados.

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; "

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”.

A continuación mostramos la implementación sobre la que automatizaremos las pruebas.

```
public class Reserva {

  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 Exception {

    ArrayList<String> errores = new ArrayList<String>();
    if(!compruebaPermisos(login, password, Usuario.BIBLIOTECARIO)) {
        errores.add("ERROR de permisos");
    } else {
        IOperacionBO io = new Operacion();
      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.ejercicio3.excepciones**".

```
public class JDBCException extends Exception {
  //no es necesario el constructor, excepto para ReservaException
}
```

```
public class ReservaException extends Exception {
  public ReservaException(String message) {super(message);}
}
```

Definición de la interfaz (paquete: **ppss.ejercicio3**):

```
public interface IOperacionBO {
  public void operacionReserva(String socio, String isbn)
              throws IsbnInvalidoException, JDBCException, SocioInvalidoException;
}
```

Definición del tipo enumerado (paquete: **ppss.ejercicio3**):

```
public enum Usuario {
  BIBLIOTECARIO, ALUMNO, PROFESOR
}
```