P06- Dependencias externas 2: mocks

OBSERVACIONES SOBRE LA IMPLEMENTACIÓN: (recordatorio extraído del enunciado de la práctica)

- Si realizamos una verificación basada en el ***comportamiento***, SIEMPRE nos va a importar el orden en el que los dobles van a ser invocados desde la SUT (independientemente de si dichos dobles pertenecen a la misma clase o no).

  Si necesitas incluir un "mock parcial" en un "StrictControl", incluye el "StrictControl" como parámetro del método para crear la clase que contiene nuestros dobles.

```
ctrl = EasyMock.createStrictControl();
partialMock = EasyMock.partialMockBuilder(...)
                .addMockedMethod(...)
                .mock(ctrl);
```

  Sólo debes usar un StrictControl y/o un PartialMock si es IMPRESCINDIBLE!!!

- Si al programar ***varias expectativas de un doble*** necesitas capturar una excepción, usa un único *assertDoesNotThrow*, e incluye las expectativas para ese doble en la expresión lambda:

```
//doble() es un método void que puede lanzar una excepción
assertDoesNotThrow(()-> {
    mock1.doble(param1,param2);
    EasyMock.expectLastCall().andThrow(...);
    mock1.doble(param3,param4);
    mock1.doble(param5,param6);
    EasyMock.expectLastCall().andThrow(...);
});
```

- Al programar ***varias expectativas de un doble*** encadena las expectativas (NO uses bucles!!):

```
//doble() es un método void que puede lanzar una excepción
assertDoesNotThrow(()->clase.doble(anyString(), anyString()));
EasyMock.expectLastCall()
        .andThrow(...)
        .andVoid()
        .andThrow(..)
        .andVoid().anyTimes();
```

➪ ➪ ANEXO 2: Observaciones sobre los ejercicios de P06

SUT: ***compruebaPremio()***

- ***Dependencias que requieren un doble***: *Premio.generaNumero(), ClienteWebService.obtenerPremio()*
- *La sut es testable*
- ***Dobles***: *generaNumero() requiere un mock parcial. Necesitamos un StrictControl*

SUT: ***contarCaracteres()***

- ***Dependencias que requieren un doble***: *FileReader.read(), FileReader.close()*
- *La sut NO es testable.* El punto de inyección para ambos dobles será una factoría local, que será una nueva dependencia externa para la que necesitamos un doble, el cual pertenece a un objeto de tipo *FicheroTexto*.
- ***Dobles***: el doble de la factoria local requiere un mock parcial. *Necesitamos un StrictControl*

SUT: ***notifyUsers()***

- ***Dependencias que requieren un doble***: *(a) NotifyCenter.getServer(),*
  (b) NotiifyCenter.sendNotify()
  (c)LocalDate.now()
  (d) MailServer.findMailItemsWithDate()
- *La sut NO es testable.* Necesitamos un punto de inyección para *LocalDate.now()* que será una factoria local a *NotifyCenter*. Este punto de inyección es una nueva dependencia externa *(e)*.
- ***Dobles***: *Las dependencias (a),(b) y (e) requieren un mock parcial*
  Necesitamos un StrictControl

SUT: ***reserva()***

- ***Dependencias que requieren un doble***: *(a) Reserva.compruebaPermisos(),*
  (b) FactoriaBOs.getOperacionBO()
  (c) IOperacionBO.operacionReserva()
- *La sut NO es testable.* Necesitamos un punto de inyección para (b): *setter*
- ***Dobles***: *La dependencia (a) requiere un mock parcial*
  Necesitamos un StrictControl,

Recuerda que para implementar mocks con EasyMock necesitas realizar **necesariamente** los **4** pasos indicados en la **tr. 13 de S06**)

➡ No puedes usar un objeto de tipo *NiceMock* para implemetar un mock

Recuerda que para implementar stubs con EasyMock necesitas realizar **necesariamente** los **3** pasos indicados en la **tr. 6 de S06**)

➡ No puedes usar un objeto de tipo *Mock* o *StrictMock* para implemetar un stub

Recuerda también que TODAS las normas explicadas en la sesión S03 (Drivers) se aplican con independencia de que estemos usando dobles o no en el driver. Por lo tanto, es obligatorio intentar reducir la duplicación de código usando un @BeforeEach siempre que se pueda.