Orientación a Objetos 2 – Práctica 5

Ejercicio 1: ToDoList

Se desea definir un sistema de seguimiento de tareas similar a Jira[1].
En este sistema hay tareas en las que se puede definir el nombre y una serie de comentarios. Las tareas atraviesan diferentes etapas a lo largo de su ciclo de vida y ellas son: *pending*, *in-progress*, *paused* y *finished*. Cada tarea debe estar modelada mediante la clase ToDoItem con el siguiente protocolo:

```
ToDoItem  class>>  name: aName
"Instancia un ToDoItem nuevo en estado pending con aName como
nombre"

ToDoItem>> start
"Pasa el ToDoItem a in-progress (siempre y cuando su estado actual
sea pending, si se encuentra en otro estado, no hace nada)".

ToDoItem>>togglePause
"Pasa la tarea a paused si su estado es in-progress, o a
in-progress si su estado es paused. Caso contrario (pending o
finished) genera un error informando la causa específica del
mismo"

ToDoItem>> finish
"Pasa el ToDoItem a finished (siempre y cuando su estado actual
sea in-progress o pausada, si se encuentra en otro estado, no hace
nada)"

ToDoItem>>workedTime
"Retorna el tiempo que transcurrió desde que se inició la tarea
(start) hasta que se finalizó. En caso de que no esté finalizada,
hasta el actual. Si la tarea no se inició genera un error
informando la causa específica del mismo."

ToDoItem>>addComment: aComment
"Agrega un comentario a la tarea siempre y cuando no haya
finalizado. Caso contrario no hace nada."
```

[1] https://es.atlassian.com/software/jira

Nota: para generar o levantar un error debe utilizar la expresión

```
self error: 'Este es mi mensaje de error'
```

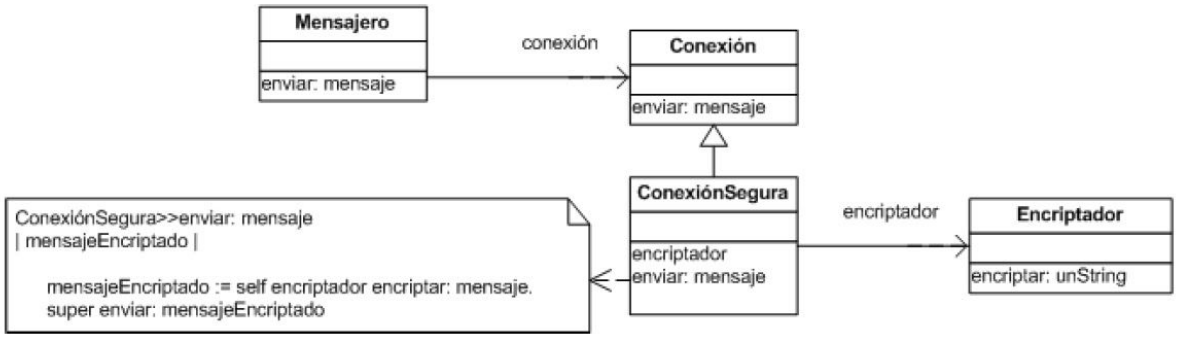
El mensaje de error específico que se espera en este ejercicio debe ser descriptivo del caso. Por ejemplo, para el #togglePause, el mensaje de error debe indicar que el ToDoItem no se encuentra en in-progress o paused: `self error:'el objeto ToDoItem no se encuentra en pause o in-progress'`

Tareas:

1. Modele una solución orientada a objetos para el problema planteado utilizando un diagrama UML de clases. Si utilizó algún patrón de diseño indique cuáles son los participantes en su modelo de acuerdo a Gamma et al.
2. Implemente su solución en Pharo, para comprobar cómo funciona recomendamos usar test cases.

Ejercicio 2: Encriptador

En un sistema de mensajes instantáneos (como Hangouts) se envían mensajes de una máquina a otra a través de una red. Para asegurar que la información que pasa por la red no es espiada, el sistema utiliza una conexión segura. Este tipo de conexión encripta la información antes de enviarla y la desencripta al recibirla. La siguiente figura ilustra un posible diseño para este enunciado.



El encriptador utiliza el algoritmo RSA. Sin embargo, se desea agregar otros algoritmos (diferentes algoritmos ofrecen distintos niveles de seguridad, overhead en la transmisión, etc.).

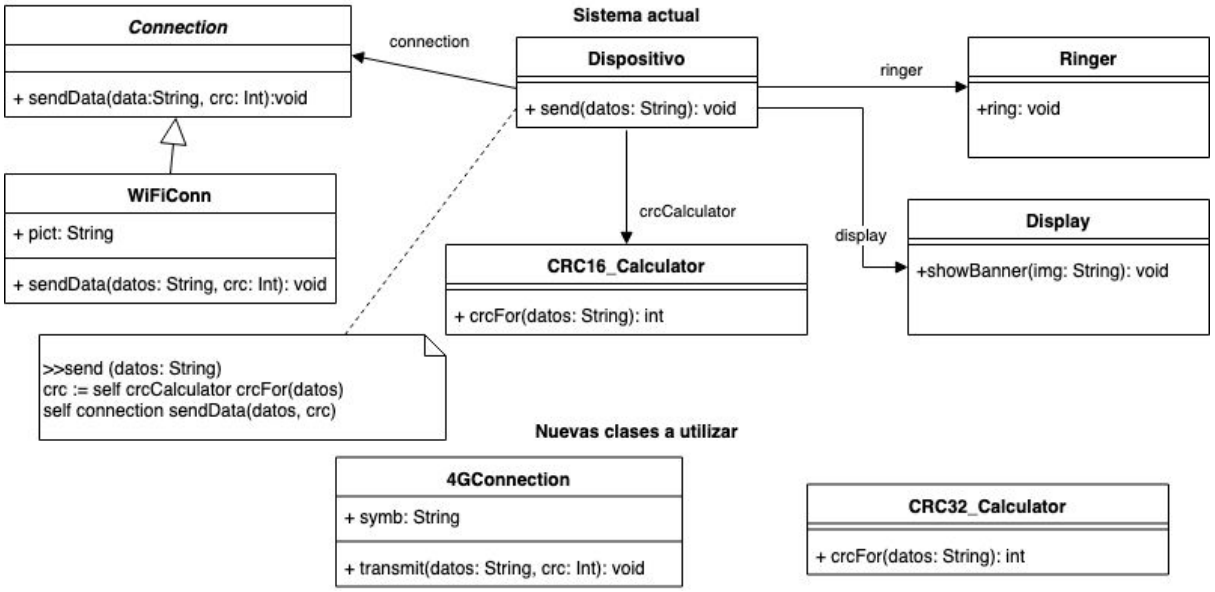
Tareas:

1. Modifique el diseño para que el objeto Encriptador pueda encriptar mensajes usando los algoritmos Blowfish y RC4, además del ya soportado RSA.

2. Documente mediante un diagrama de clases UML indicando los roles de cada clase.

Ejercicio 3 - Dispositivo móvil y conexiones

Sea el software de un dispositivo móvil que utiliza una conexión WiFi para transmitir datos. La figura muestra parte de su diseño:



El dispositivo utiliza, para asegurar la integridad de los datos emitidos, el mecanismo de cálculo de redundancia cíclica que le provee la clase CRC16_Calculator que recibe el mensaje #crcFor: con los datos a enviar y devuelve un valor entero. Luego el dispositivo envía a la conexión el mensaje #sendData: crc: con ambos parámetros (los datos y el entero calculado).

Se desea hacer dos cambios en el software. En primer lugar, se quiere que el dispositivo tenga capacidad de ser configurado para utilizar conexiones 4G. Para este cambio se debe utilizar la clase 4GConnection.
Además se desea poder configurar el dispositivo para que utilice en distintos momentos un cálculo de CRC de 16 o de 32 bits. Es decir que en algún momento el dispositivo seguirá utilizando CRC16_Calculator y en otros podrá ser configurado para utilizar la clase CRC32_Calculator. Se desea permitir que en el futuro se puedan utilizar otros algoritmos de cálculo de CRC.

Cuando se cambia de conexión, el dispositivo muestra en pantalla el símbolo correspondiente (que se obtiene con el getter #pict para el caso de WiFiConn y #symb de 4GConnection) y se utiliza el objeto Ringer para emitir un #ring.

Tanto las clases existentes como las nuevas a utilizar pueden ser ubicadas en las jerarquías que corresponda y se les pueden agregar mensajes, pero no se pueden modificar los mensajes que ya existen porque otras partes del sistema podrían dejar de funcionar.

Dado que esto es una simulación y no dispone del hardware ni emulador para esto ud debe loggear al Transcript cada cosa que ocurre en el sistema, dato que se envíe o qué está mostrando la pantalla en cada momento.

Tareas:
Modele los cambios necesarios para poder agregar al protocolo de la clase Dispositivo los mensajes

```
Dispositivo>>conectarCon: unaConexion
"Pasa a utilizar unaConexion, muestra en display su símbolo y genera el sonido"

Dispositivo>>configurarCRC: unCRCcalculator
"Configura el uso de unCRCcalculator para validar la integridad de los datos a enviar"
```

Entregables:

1. Diagrama UML de clases para su solución al problema planteado. Indique claramente el o los patrones de diseño que utiliza en el modelo y el rol que cada clase cumple en cada uno.
2. Implemente en Pharo todo lo necesario para asegurar el envío de datos por cualquiera de las conexiones y el cálculo adecuado del índice de redundancia cíclica.
3. Implemente test cases al menos para los siguientes métodos:
    a. Dispositivo>>send:
    b. `Dispositivo>>conectarCon:`
    c. `Dispositivo>>configurarCRC:`

Nota: para implementar las CRC16_Calculator y CRC32_Calculator utilice la clase CRC (provista por Pharo) de la siguiente manera:

```
CRC crc16FromCollection: 'message'.  " retorna el crc16 para el string 'message' "
CRC crc32FromCollection: 'message'.  " retorna el crc32 para el string 'message' "
```