Orientación a Objetos 2 – Curso 2022
Práctica 3

Fecha de última edición: 4 abril 2022

Ejercicio 1: ToDoItem

Se desea definir un sistema de seguimiento de tareas similar a Jira[1].
En este sistema hay tareas en las cuales 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:

```
public class ToDoItem {
	/**
	* Instancia un ToDoItem nuevo en estado pending con <name> como nombre.
	*/
	 public ToDoItem(String name)

	/**
	* Pasa el ToDoItem a in-progress (siempre y cuando su estado actual sea
	* pending, si se encuentra en otro estado, no hace nada)
	*/
	 public void start()

	/**
	* 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
	*/
	 public void togglePause()

	/**
	* 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)
	*/
	 public void finish()

	/**
	* Retorna el tiempo que transcurrió desde que se inició la tarea (start)
```

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

```
        * hasta que se finalizó. En caso de que no esté finalizada, el tiempo que
        * haya transcurrido hasta el momento actual. Si la tarea no se inició,
        * genera un error informando la causa específica del mismo.
        */
         public Duration workedTime()


        /**
        * Agrega un comentario a la tarea siempre y cuando no haya finalizado. Caso
        * contrario no hace nada."
        */
         public void addComment(String comment)
}
```

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

```
throw new RuntimeException("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 método togglePause() , el mensaje de error debe indicar que el ToDoItem no se encuentra en in-progress o paused:

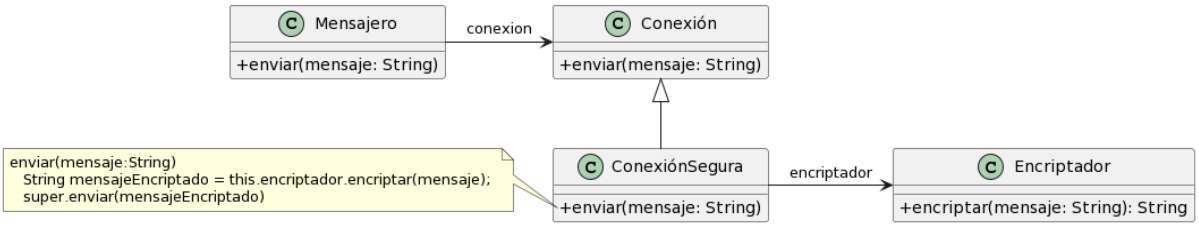
```
throw new RuntimeException("El objeto ToDoItem no se encuentra en pause o
in-progress");
```

Tareas:

1. Modele una solución orientada a objetos para el problema planteado utilizando un diagrama de clases UML. 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 Java. 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.



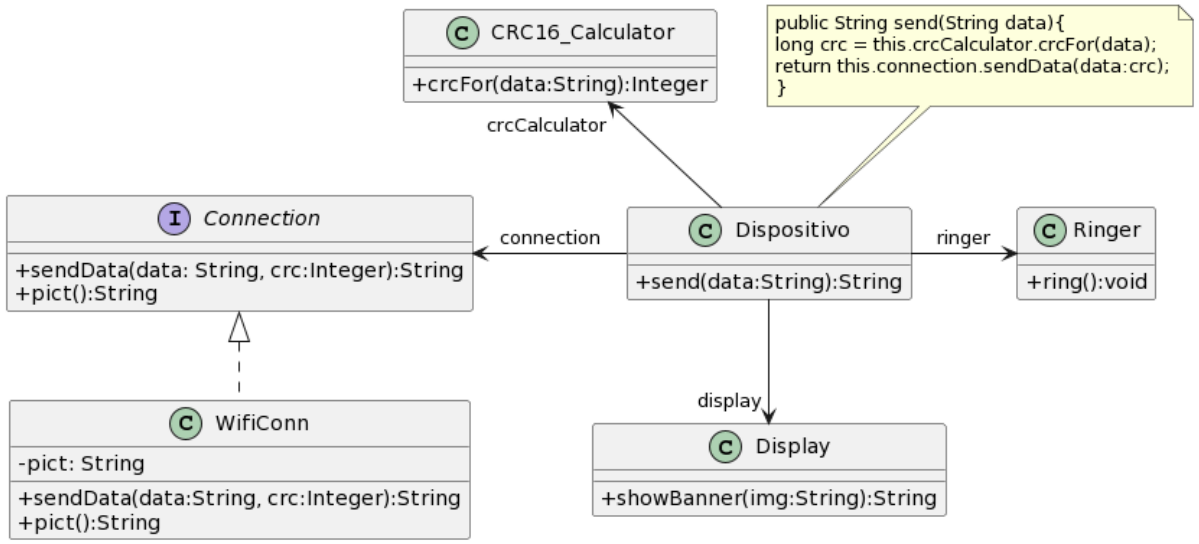
El encriptador utiliza el algoritmo RSA. Sin embargo, se desea agregar otros algoritmos (diferentes algoritmos ofrecen distintos niveles de seguridad, overhead en la transmisión, etc.).

Tareas:

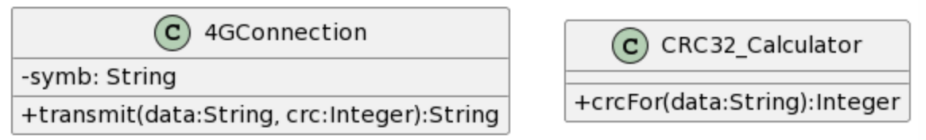
1. Modifique el diseño para que el objeto Encriptador pueda encriptar mensajes usando los algoritmos Blowfish y RC4, además del ya soportado RSA.
2. Documente mediante un diagrama de clases UML indicando los roles de cada clase.

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:



Nuevas clases a utilizar:



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(data: String)` con los datos a enviar y devuelve un valor numérico. Luego el dispositivo envía a la conexión el mensaje `sendData` con ambos parámetros (los datos y el valor numérico 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 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 (modificar la clase de la que extienden o la interfaz que implementan) 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 de hardware ni emulador para esto, la signatura de los mensajes se ha simplificado para que se retorne un String descriptivo de los eventos que suceden en el dispositivo y permitir de esta forma simplificar la escritura de los tests.

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

- cambiar la conexión, ya sea la 4GConnection o la WifiConn. En este método se espera que se pase a utilizar la conexión recibida, muestre en el display su símbolo y genere el sonido.
- poder configurar el calculador de CRC, que puede ser el CRC16_Calculator, el CRC32_Calculator, o pueden ser nuevos a futuro.

Tareas:

1. Realice un 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 Java 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 para los siguientes métodos de la clase Dispositivo:
    a. `send`
    b. `conectarCon`
    c. `configurarCRC`

En cuanto a CRC16_Calculator, puede utilizar la siguiente implementación:

```
public long crcFor(String datos) {
	int crc = 0xFFFF;
	for (int j = 0; j < datos.getBytes().length; j++) {
		crc = ((crc >>> 8) | (crc << 8)) & 0xffff;
		crc ^= (datos.getBytes()[j] & 0xff);
		crc ^= ((crc & 0xff) >> 4);
		crc ^= (crc << 12) & 0xffff;
		crc ^= ((crc & 0xFF) << 5) & 0xffff;
	}
	crc &= 0xffff;
	return crc;
}
```

Nota: para implementar CRC32_Calculator utilice la clase java.util.zip.CRC32 de la siguiente manera:

```
CRC32 crc = new CRC32();
String datos = "un mensaje";
crc.update(datos.getBytes());
long result = crc.getValue();
```

Ejercicio 4 - Decodificador de películas

Sea una empresa de cable *on demand* que entrega decodificadores a sus clientes para que miren las películas que ofrece. El decodificador muestra la grilla de películas y también sugiere películas.

Usted debe implementar la aplicación para que el decodificador sugiera películas. El decodificador conoce la grilla de películas (lista completa que ofrece la empresa), como así también las películas que reproduce. De cada película se conoce título, año de estreno, películas similares y puntaje. La similaridad establece una relación recíproca entre dos películas, por lo que si A es similar a B entonces también B es similar a A.

Cada decodificador puede ser configurado para que sugiera 3 películas (que no haya reproducido) por alguno de los siguientes criterios:

(i) novedad: las películas más recientes.
(ii) similaridad: las películas más nuevas son similares a alguna película que reprodujo.
(iii) puntaje: las películas de mayor puntaje, para igual puntaje considera las más recientes.

Tenga en cuenta que la configuración del criterio de sugerencia del decodificador no es fija, sino que el usuario la debe poder cambiar en cualquier momento. El sistema debe soportar agregar nuevos tipos de sugerencias aparte de las tres mencionadas.

Sea un decodificador que reprodujo Thor y Rocky, y posee la siguiente lista de películas:

Thor, 7.9, 2007 (Similar a Capitan America, Iron Man)
Capitan America, 7.8, 2016 (Similar a Thor, Iron Man)
Iron man, 7.9, 2010 (Similar a Thor, Capitan America)
Dunkirk, 7.9, 2017
Rocky, 8.1, 1976 (Similar a Rambo)
Rambo, 7.8, 1979 (Similar a Rocky)

Las películas que debería sugerir son:

(i) Dunkirk, Capitan America, Iron man

(ii) Capitán América, Iron man, Rambo

(iii) Dunkirk, Iron man, Capitan America

Nota: si existen más de 3 películas con el mismo criterio, retorna 3 de ellas sin importar cuales. Por ejemplo, si las 6 películas son del 2018, el criterio (i) retorna 3 cualquiera.

Tareas:

1. Realice el diseño de una correcta solución orientada a objetos con un diagrama UML de clases.
2. Si utiliza patrones de diseño indique cuáles y también indique los participantes de esos patrones en su solución según el libro de Gamma et al.
3. Escriba un test case que incluya estos pasos, con los ejemplos mencionados anteriormente:
    - configure al decodificador para que sugiera por similaridad (ii)
    - solicite al mismo decodificador las sugerencias
    - configure al mismo decodificador para que sugiera por puntaje (iii)
    - solicite al mismo decodificador las sugerencias
4. Programe su solución en Java. Debe implementarse respetando todas las buenas prácticas de diseño y programación de POO.