**Universidad ORT Uruguay**

**Facultad de Ingeniería**

Aspectos de Seguridad - Obligatorio

Entregado como requisito para la obtención del crédito Aspectos de Seguridad

Segundo Semestre 2019

Marcela Ferraz - 200112

Mauricio Zito - 111725

Docentes: Santiago Paz, Felipe Sotuyo

Tabla de contenido

[Introducción 3](#_Toc26827780)

[Tecnología utilizada 3](#_Toc26827781)

[Parte 1 – Sistema seguro Bell Lapadula 4](#_Toc26827782)

[1.1 Desarrollo 4](#_Toc26827783)

[1.2 Instalación 6](#_Toc26827784)

[1.3 Prueba 9](#_Toc26827785)

[1.4 Conclusión 10](#_Toc26827786)

[Parte 2 – Canal Encubierto 11](#_Toc26827787)

[Introducción 11](#_Toc26827788)

[2.1 Desarrollo 11](#_Toc26827789)

[2.2 Instalación 15](#_Toc26827790)

[2.3 Prueba 15](#_Toc26827791)

[2.4 Conclusión 17](#_Toc26827792)

[Conclusión Final 18](#_Toc26827793)

# Introducción

En el marco de la materia Aspectos de seguridad, se solicita un trabajo práctico en caracter de obligatorio donde se implementará un modelo de seguridad basado en el planteado por la dupla de Billy Elliott Bell y Len LaPadula, consiste en dividir el permiso de [acceso](https://es.wikipedia.org/wiki/Acceso) de los usuarios a la información en función de etiquetas de seguridad. Por ejemplo, en sistemas militares norteamericanos, categorizándola en 4 niveles: no clasificado, confidencial, secreto y ultrasecreto.

El modelo define 2 reglas de control de acceso mandatorio (MAC):

Propiedad de seguridad simple: Un sujeto de un determinado nivel de seguridad no puede leer un objeto perteneciente a un nivel de seguridad más alto.

Propiedad \*: Un sujeto de un determinado nivel de seguridad no puede escribir un objeto perteneciente a un nivel de seguridad más bajo. (También llamada propiedad de confinamiento).

## Tecnología utilizada

El obligatorio fue implementado utilizando:

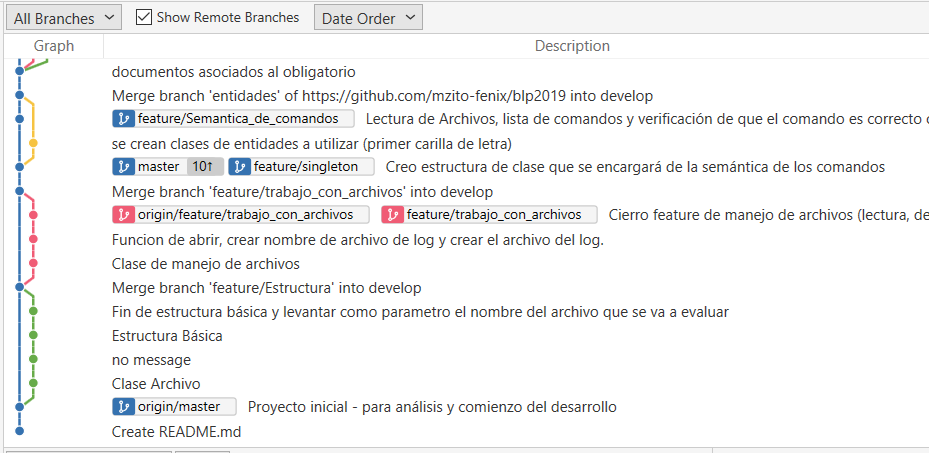
* Lenguaje Java
* IDE Netbeans versión 8.0.2

Version de Java

* JDK 1.8

Control de versiones, desarrollo compartido y organización de la tarea

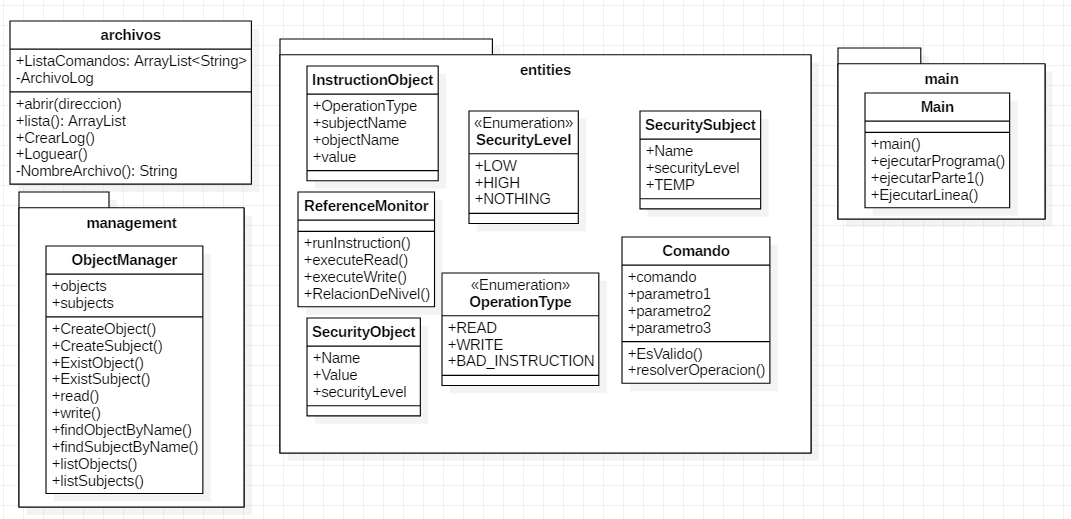
* Utilizamos github y sourcetree para poder trabajar en conjunto



## Parte 1 – Sistema seguro Bell Lapadula

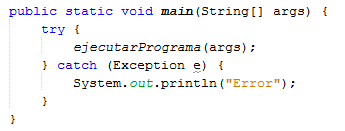
### Desarrollo

Vista de descomposición



|  |  |  |  |
| --- | --- | --- | --- |
|  | archivos | Archivos | Clase que controla la lectura y escritura de archivos (lectura de instrucciones y escritura de log), interacción con el sistema operativo. |
| Entities | InstructionObject | Clase que resume información para la ejecución del comando |
| OperationType | Enumeración de tipos de operaciones.  READ, WRITE, BAD\_INSTRUCTION |
| ReferenceMonitor | Control de instrucciones, control de relación de permiso y ejecución. |
| SecurityLevel | Enumeración de tipos de niveles de seguridad  LOW, HIGH, NOTHING |
| SecurityObject | Representación de un Objeto del modelo Bell Lapadula |
| SecuritySubject | Representación de un Sujeto del modelo Bell Lapadula |
| Comando | Encargado de la verificación de correctitud y separación de las operaciones y parámetros para su posterior ejecución |
| Main | Main | Programa principal, funcion Main. |
| Management | ObjectManager | Controlador que contiene el comportamiento de una base de datos para registrar el listado de sujetos, objetos y sus estados |

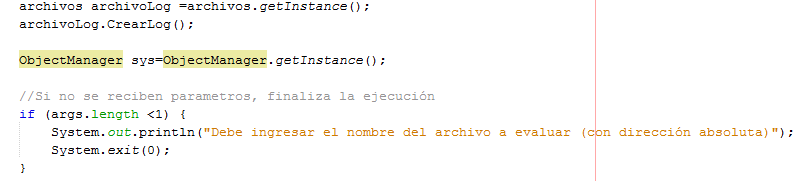
**La función principal**



La función “ejecutarPrograma” recibe los parámetros recibidos por linea de comandos directo del main.



Varias de las funciones fueron implementadas con el patrón “Singleton”, de ésta forma es posible utilizarlas de forma transversal desde diferentes secciones del proyecto.





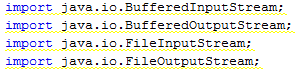
### Instalación

Para realizar la prueba, solo hay que copiar la carpeta “test” que se encuentra dentro del proyecto a cualquier carpeta del equipo destino.

|  |  |
| --- | --- |
|  | En la misma carpeta debe estar el archivo “InstructionList.txt” que es donde se encuentran las instrucciones a ejecutar. |

#### Sobre el Log

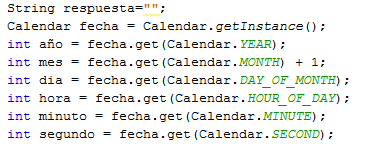
Se implementó una clase para realizar el log del sistema, utilizando las siguientes librerías.



Se compone de 2 partes

1. La creación del archivo de log

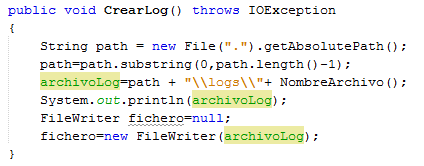
En la subcarpeta “logs” del proyecto se genera un archivo bajo el siguiente patrón de nombre



Se compone de la palabra “Log\_” seguido de Año, Mes, Dia, Hora, minuto y segundo y le asigna la extensión “.log” de ésta forma nos aseguramos que los logs se puedan acumular y no se pierdan registros de cada ejecución independiente.

**Crear Log**

La función CrearLog se encarga de crear un archivo de registro con nombre único y por única vez para toda la ejecución.

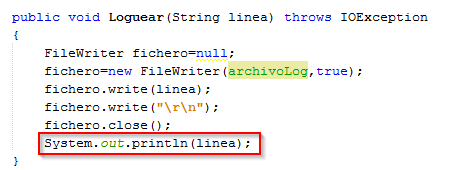




1. El Append de nueva linea al archivo de log

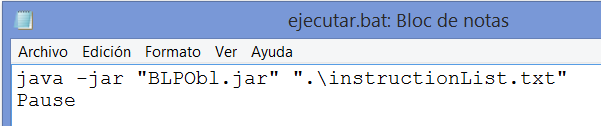
Es la función de la clase que se encarga de hacer “Append” al archivo utilizado y designado por la instancia de ejecución actual.

Tiene la doble funcionalidad de dejar registrado en un archivo la salida y tambien de mostrar en pantalla (por consola) lo que el programa va realizando.



#### 1.2.2 Ejecutar

Para correr el programa se debe ejecutar utilizando el archivo “Parte1.bat” que tiene el siguiente contenido.



### Prueba

#### Prueba 1 – Verificación según Instruction list de letra del obligatorio

|  |  |
| --- | --- |
| Archivo | Resultado de cada linea esperada |
|  |  |

|  |  |
| --- | --- |
| Resultado esperado | Resultado tras corrida |
|  |  |

#### Prueba 2 – Verificación según Instruction list de Aulas

|  |  |
| --- | --- |
| Archivo | Resultado de cada linea esperada |
|  |  |

|  |  |
| --- | --- |
| Resultado esperado | Resultado tras corrida |
|  |  |

### 1.4 Conclusión

La propuesta de la implementación de un sistema seguro con el concepto básico planteado por Bell y Lapadula fomenta la comprensión del método y las decisiones que se deben tomar para que la información vaya en un sentido seguro.

Si bien en ésta ocasión se trabajaba con números enteros, es posible implementar con documentos completos e incluso encriptados.

La implementación del modelo propuesto originalmente por sus creadores nos ofrecieron una mayor comprensión sobre la potencia que le ofrece éste modelo al concepto de confidencialidad.

## Parte 2 – Canal Encubierto

### Introducción

Un canal encubierto (del inglés covert channel), es un canal que puede ser usado para transferir información desde un usuario de un sistema a otro, usando medios no destinados para este propósito por los desarrolladores del sistema.

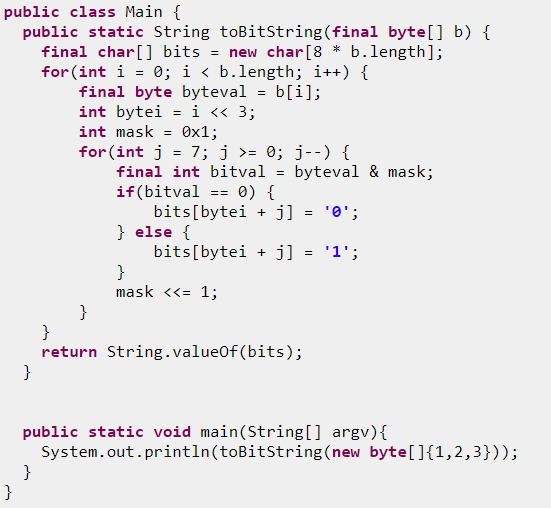
En ésta segunda parte del obligatorio se solicita el desarrollo de un canal encubierto para intentar transferir información entre 2 sujetos de diferentes niveles sorteando las reglas del modelo de Bell Lapadula.

### 2.1 Desarrollo

El mensaje recibido originalmente mediante el archivo XXXXXXXX y por parámetro del sistema, se debe descomponer en caracteres (bytes) y luego en bits (1 y 0) ya que la transferencia se hace en “goteos de bits” para rearmar le mensaje en el destinatario.

**Investigación:**

Como pasar un array de bytes en un array de bits:



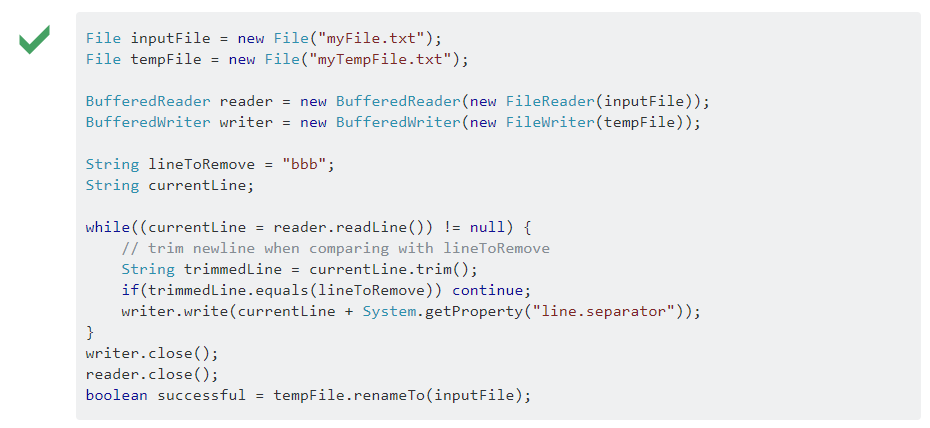
Será necesario confeccionar una función para tales efectos, logrando una cadena de bits desde el texto original.

Fuente:

<http://www.java2s.com/Tutorials/Java/Data_Type/Array_Convert/Convert_byte_array_to_bit_string_in_Java.htm>

Vaciar contenido de archivo:

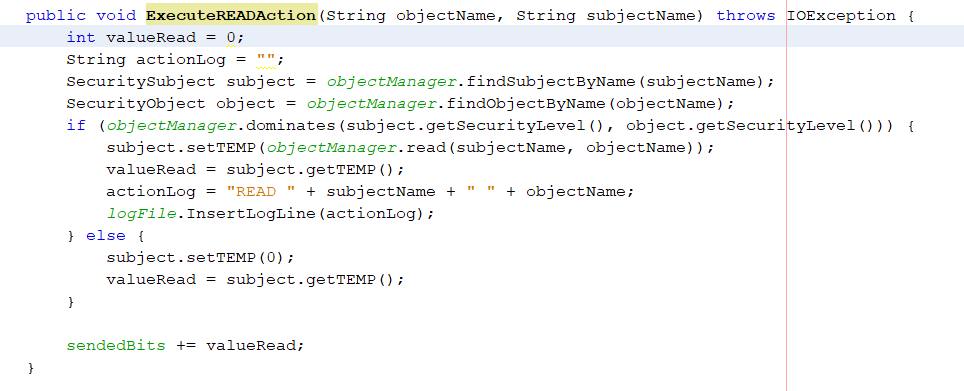
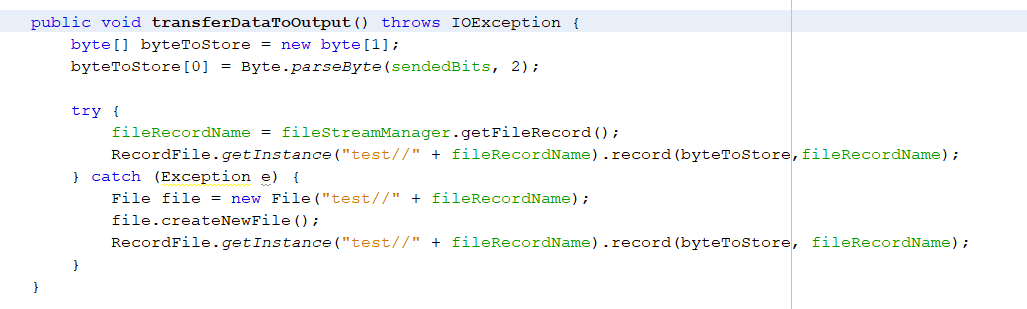
Cuando vamos a escribir en un archivo, primero procedemos a eliminar el contenido “viejo” del mismo para luego escribir los resultados de la nueva ejecucion.

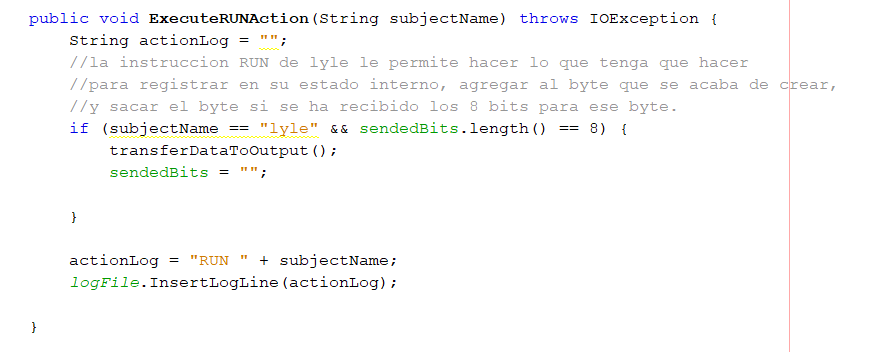
En el caso de no existir el archivo no hace nada, de lo contrario elimina su contenido dejandolo en blanco.

Fuente:

<https://stackoverflow.com/questions/1377279/find-a-line-in-a-file-and-remove-it>

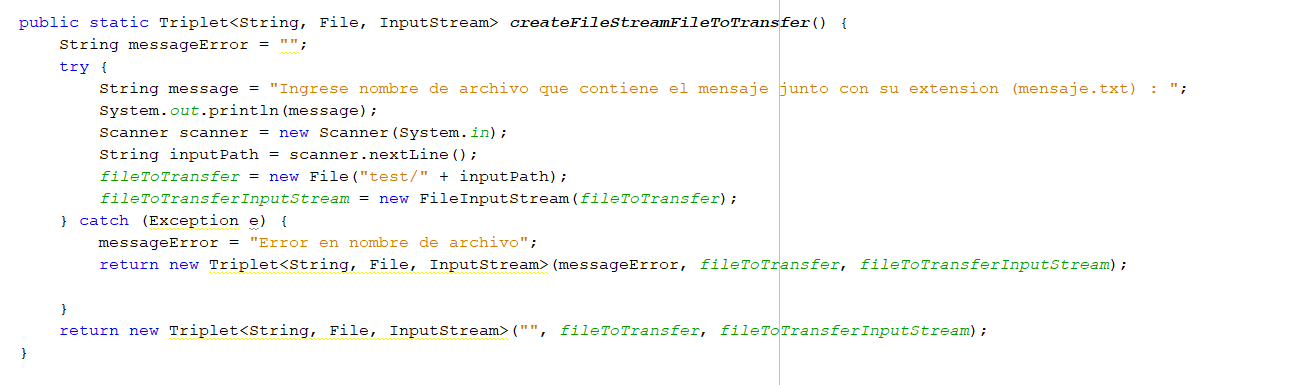
El proceso de resolución del problema es el siguiente:

1. Se crean los sujetos involucrados
2. Se lee el archivo que se va a transferir
3. Se lee la secuencia de lectura
4. Se traduce el archivo a transferir a bytes y luego a bits
5. La lectura del archivo de secuencia indica en que orden se deben ejecutar las intervenciones de los diferentes sujetos.
6. Si es el turno de HAL y el objeto existe, debe eliminarlo y ejecutar el proceso de “dejar” el dato creado y con un estado legible por LYLE.
7. Si es el turno de LYLE, se ejecuta el proceso de “leer” el dato creado y con un estado legible
8. LYLE va armando el mensaje en base a los bits recibidos que va leyendo. En caso de no poder leerlo (por su nivel de seguridad) se acumula en la variable “sendedBits” un 0, en caso de poder leerlo se concatena al mismo un 1. Generando de ese modo una acumulacion de bits que luego se traducen a un char de ASCII.
9. Por ultimo lyle siempre ejecuta la secuencia RUN. Una vez que el proceso de secuencia finalice, se arman paquetes de 8 bits, se transforma a bytes y se pasa a Ascii nuevamente creándose el archivo mensaje.salida.txt y escribiendo en el mismo los correspondientes caracteres.

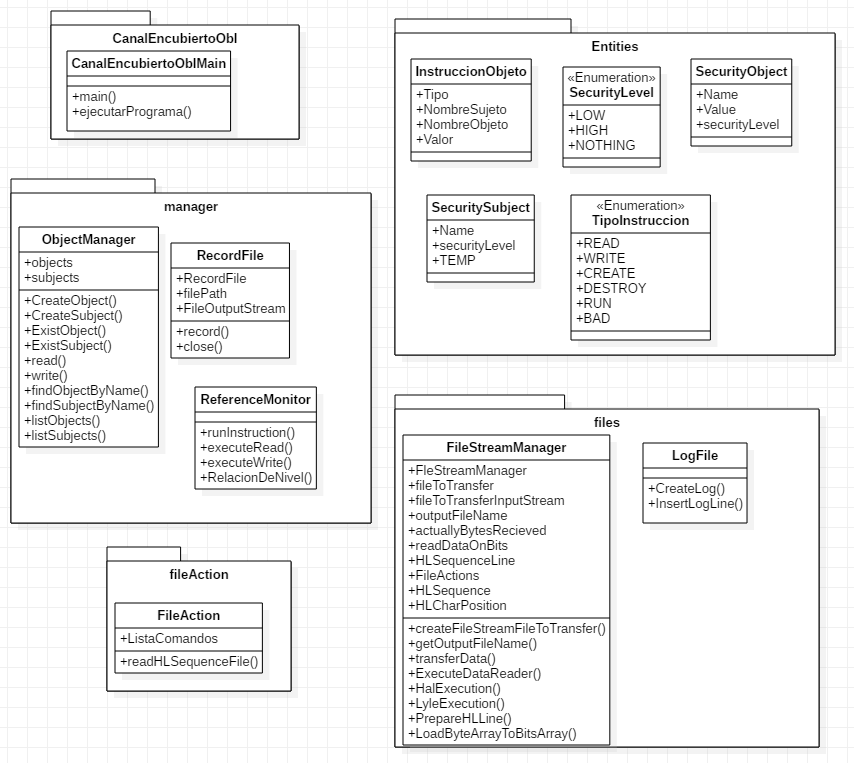
En el caso de completar los bits necesarios, entonces procede a grabar el carácter en el nuevo archivo output. Pot ultimo, vuelve a dejar la variable sendedBits vacia para poder hacer el siguiente procesamiento del proximo carácter que se desee transmitir.

**Librerías adicionales**

En ésta ocasión fue neceario utilizar la librería javatuples-1.2.jar (<http://www.java2s.com/Code/Jar/j/Downloadjavatuples12jar.htm>), la cual fue utilizada para apoyar el analisis del archivo importado, tercerizando dicho control y separación. En caso de que ocurra algun inconveniente, dicha herramienta se encarga de cargar el error y ofrecer información sobre el mismo, nombre del archivo desde donde se va a leer el mensaje.

Se adjunta una imagen donde se utilizo la herramienta:

Vista de descomposición



Detalle de clases, agrupación y función

|  |  |  |  |
| --- | --- | --- | --- |
|  | canalencubiertoobl | CanalEncubiertoOblMain | Clase Main, principal, inicial. |
| Entities | InstruccionObjeto | Clase que resume información para la ejecución del comando |
| SecurityLevel | Enumeración de tipos de niveles de seguridad  LOW, HIGH, NOTHING |
| SecurityObject | Representación de un Objeto del modelo Bell Lapadula |
| TipoInstruccion | Enumeración de tipos de operaciones.  READ, WRITE, CREATE, DESTROY, RUN, BAD\_INSTRUCTION |
| fileAction | FileAction | Encargada de consultar al usuario, leer y cargar la secuencia de ejcución en un buffer. |
| files | FileStreamManager | Organiza y administra la ejecución dependiendo del turno en el archivo de secuencia |
| LogFile | Manager del log de la aplicación, crea y adiciona lineas. |
| manager | ObjectManager | Controlador que contiene el comportamiento de una base de datos para registrar el listado de sujetos, objetos y sus estados |
| RecordFile | Encargado de la creación del archivo de salida |
| ReferenceMonitor | Control de instrucciones, control de relación de permiso y ejecución. |

### 2.2 Instalación

Para realizar la prueba, solo hay que copiar la carpeta “test” que se encuentra dentro del proyecto a cualquier carpeta del equipo destino.

|  |  |
| --- | --- |
|  | En la misma carpeta debe estar el archivo “mensaje.txt” que es donde se encuentra el mensaje para transmitir y “secuencia.txt” que es donde se encuentra la secuencia de ejecución por usuario H=Hal, L=Lyle. |

**\*\* MARCELA**, CREO QUE DEBERIAMOS HACER QUE FUNCIONE DESDE CONSOLA COMO LA PARTE 1, COMO SI NO TUVIERA LA IDE DE NETBEANS, DENTRO DE UNA CARPETA TEST COMO LA DEL EJEMPLO VOS QUE DECIS?

### 2.3 Prueba

A Continuación se muestran 3 pruebas realizadas de transferencia de datos, utilizando la metodología de canal encubierto propuesta por el obligatorio y que fue implementada.

#### Prueba 1 – Verificación según el mensaje y la secuencia ofrecida por el obligatorio

|  |  |
| --- | --- |
| Mensaje enviado | Secuencia |
|  |  |
| Mensaje recibido | |
|  | |

Resultado: OK

#### Prueba 2 – Como la prueba 1, pero cambiando el archivo de secuencia

|  |  |
| --- | --- |
| Mensaje enviado | Secuencia |
|  |  |
| Mensaje recibido | |
|  | |

Resultado: OK

#### Prueba 3 – Como la prueba 1, pero cambiando el mensaje a transmitir

|  |  |
| --- | --- |
| Mensaje enviado | Secuencia |
|  |  |
| Mensaje recibido | |
|  | |

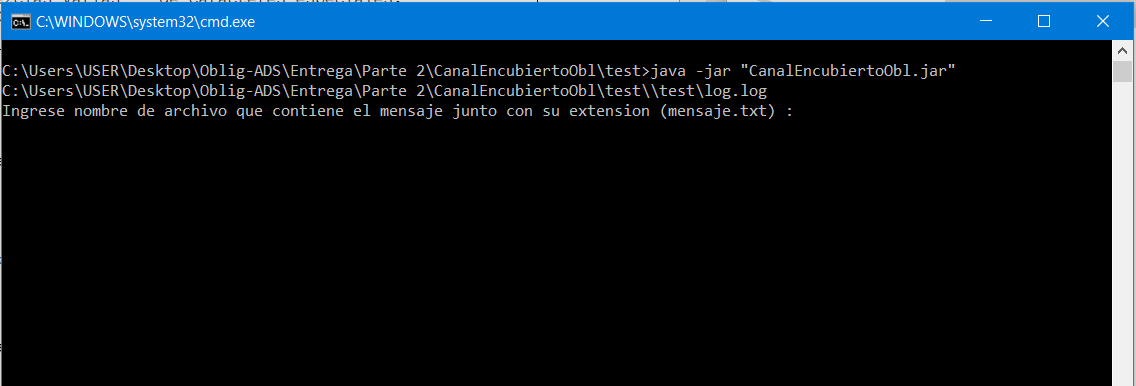
Resultado: OK

#### Bug encontrado:

Cuando se intenta pasar un mensaje con un carácter que contiene un tilde, se produce un error. Finalizando con la ejecucion.

#### Pruebas desde ejecutable:

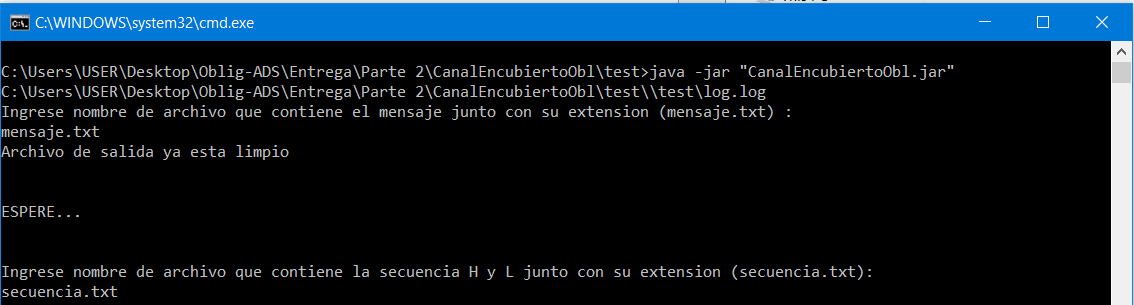
**Instalacion**:

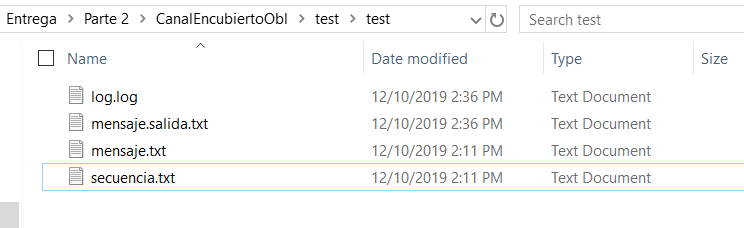
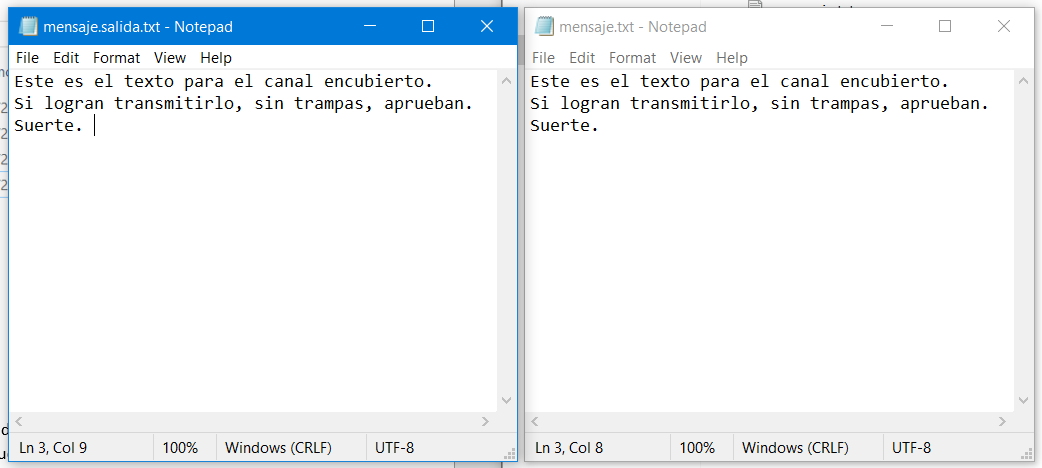
* Acceder a la carpeta Entrega/Parte 2/CanalEncubiertoObl/test/  
  Abrir el ejecutable “Parte2.bat”, donde se abrera la consola para ejecutar el proyecto.

**Prueba:**

Ingresar el nombre del archivo que contiene el mensaje que se desea transmitir, junto con su extension.

Luego colocar el nombre del archivo que contiene la secuencia H y L junto con su extension:



Una vez ejecutado estos pasos, se comienza a correr el programa, imprimiendo un log por consola de las instrucciones que se van realizando. En paralelo las mismas se van registrando en el log, en el archivo “log.log” ubicado en Entrega\Parte 2\CanalEncubiertoObl\test\test.  
  
Por ultimo, se puede ver como se crea un archivo de salida con el mismo nombre del archivo que contiene el mensaje + “.salida”.   
Se lo realizo de esta manera porque dio problemas para crearlo en una nueva carpeta aparte.

Evidencia del mensaje de salida junto con el archivo que contiene el mensaje principal a transmitir.

Nota: Para crear pruebas adicionales, se crean archivos .txt en la carpeta “\Entrega\Parte 2\CanalEncubiertoObl\test\test”.

Luego, se ejecuta nuevamente la consola y cuando pide el nombre del archivo que contiene el mensaje, se coloca alli el mismo.

Una vez ejecutado, deberia aparecer en la misma carpeta, un nuevo archivo con el nombre “[pruebaAdicional].salida.txt”.

### 2.4 Conclusión

El canal encubierto se basa en la transmisión de mensajes entre 2 sujetos que pertenecen a niveles de acceso diferentes, es decir que un sujeto con un nivel de acceso superior pueda transmitir un mensaje a un sujeto de nivel de acceso inferior (contrario a la regla estrella de Bell Lapadulla).

Mediante la utilización de la máxima granularidad posible de información (bits) es posible desarmar un mensaje en letras, éstas en bytes y luego en bits, transmitirlo mediante cambios de estados de objetos (respetando las reglas de bell-lapadula) y volver a armarlo tomando nota de los cambios de estado por parte del receptor.

El mensaje al ser granulado en bits de 0 y 1. Se logra desarmar cada carácter en bits, entonces dependiendo si el sujeto lyle puede leer o no el archivo, se va produciendo en un string “sendedBits” un conjunto de 0s y 1s que al alcanzar 8 bits (si lyle puede leer el archivo entonces se concatena un 1 en “sendedBits”, de lo contrario se concatena un 0) se los convierte a un correspondiente numero, ejemplo: si al completar los 8 bits tengo un “01000101” se convierte en un char “69” que corresponde a la letra E en la tabla ASCII y de esta manera (ya convertido en caracter) se escribe en un archivo de salida el mensaje.

## Conclusión Final

En éste trabajo obligatorio, pudimos desarrollar y probar los conceptos del modelo de Bell y Lapadulla así también como el camino para poder transmitir mensajes rompiendo las reglas básicas del mismo.

No hay dudas que es un modelo sólido de seguridad, si es que se tienen precauciones a nivel logístico para evitar la transferencia de información entre sujetos de diferentes niveles.

Mediante el análisis de la solución, la discusión entre los miembros del equipo y la implementación y testing cruzado, pudimos entender el funcionamiento del mismo y su aplicación.